**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«АВТОМАТИЗОВАНА СИСТЕМА УПРАВЛІННЯ ВИРОБНИЧИМИ ПРОЦЕСАМИ НА ОСНОВІ МОДЕЛІ RETRIEVAL-AUGMENTED GENERATION (RAG)»**

на здобуття освітнього ступеня бакалавр
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

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

_____ Арсеній КОРОВКІН

Виконав: здобувач вищої освіти групи ШІД-41

Арсеній КОРОВКІН

Керівник: *доц каф к.т.н* Антон ШАНТИР

Рецензент: _____
*(науковий ступінь, вчене звання, прізвище, ім'я)*

Київ 2025

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

Кафедра Штучного інтелекту
Ступінь вищої освіти Бакалавр
Спеціальність 122 Комп'ютерні науки
Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту
_____ Ольга ЗІНЧЕНКО
«_____» _____ 2025 р.

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коровкіну Арсенію Станіславовичу

1. Тема кваліфікаційної роботи: «Автоматизована система управління виробничими процесами на основі моделі retrieval-augmented generation (RAG)» керівник кваліфікаційної роботи Антон ШАНТИР Доц.каф к.т.н.,
*(прізвище, ім'я,науковий ступінь, вчене звання)*
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24»02.2025р. № 56

2. Строк подання кваліфікаційної роботи «23» травня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, дані інтернет мережі, мова програмування Python, середовище розроблення Visual Studio Code
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Провести аналіз існуючих систем автоматизації виробництва звикористанням ШІ
Обрати технології, інструменти для реалізації системи розпізнавання виробничих креслень
Розробити алгоритм роботи системи з урахуванням особливостей виробничих креслень

5. Перелік графічного матеріалу: презентація
6. Дата видачі завдання «24» лютого 2025 р.

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Визначення типів небажаних повідомлень у соціальних мережах | 15.03.2025 | Виконано |
| 2. | Аналіз наукової та технічної літератури з машинного навчання та користування LLM | 31.03.2025 | Виконано |
| 3. | Вибір алгоритмів та інструментів для побудови моделі класифікації | 10.04.2025 | Виконано |
| 4. | Формування датасету та підготовка навчальної вибірки | 25.04.2025 | Виконано |
| 5. | Розробка та тренування моделі на виявлення і запис інформації з конструкторської/бухгалтерської документації | 10.05.2025 | Виконано |
| 6. | Тестування системи та аналіз результатів | 25.05.2025 | Виконано |
| 7. | Підготовка звіту та доповіді до захисту | 31.05.2025 | Виконано |

Здобувач вищої освіти _____ Арсеній КОРОВКІН
(підпис)

Керівник кваліфікаційної роботи _____ Антон ШАНТИР
(підпис)

# РЕФЕРАТ

Текстова частина бакалаврської роботи на здобуття освітнього ступеня бакалавра: 52стор, 4 рис., 2 табл, 19 джерел.

*Мета роботи* - Розробити автоматизовану систему керування виробничими процесами включаючи технологію штучного інтелекту, для розпізнавання конструкторських розрахунків та бухгалтерських наряд-рахунків с записом у файл для бази даних.

*Об'єкт дослідження* - Автоматизація процесів виробництва, збільшення продуктивності підприємства

*Предмет дослідження* – методи та алгоритми розпізнавання малюнків/креслень

*Галузь використання* – Малі, середні та великі підприємства з виробництва продукції

*Короткий зміст роботи* – Кваліфікаційна робота присвячена розробці автоматизованої системи управління виробничими процесами, що забезпечує доступне використання ШІ у галузі виробництва механічних та не механічних деталей, покращує швидкість і якість виготовлення деталей

КЛЮЧОВІ СЛОВА: LLM, РОЗПІЗНАВАННЯ ДОКУМЕНТАЦІЇ, АВТОМАТИЗАЦІЯ, ВИРОБНИЦТВО, RAG, ПРОДУКТИВНІСТЬ, НАЛАГОДЖЕННЯ ПРОЦЕСІВ.

## ABSTRACT

Text part of the bachelor's thesis for the bachelor's degree: 52 pages, 4 figures, 2 tables, 19 sources.

Purpose of the work - To develop an automated production process control system including artificial intelligence technology, for recognizing design calculations and accounting orders-accounts with recording in a file for the database.

Object of research - Automation of production processes, increasing the productivity of the enterprise

Subject of research - methods and algorithms for recognizing drawings/drawings

Industry of use - Small, medium and large enterprises for the production of products

Summary of the work - The qualification work is devoted to the development of an automated production process control system, which provides affordable use of AI in the field of production of mechanical and non-mechanical parts, improves the speed and quality of manufacturing parts



**KEYWORDS: LLM, DOCUMENTATION RECOGNITION, AUTOMATION, PRODUCTION, RAG, PRODUCTIVITY, PROCESS ADJUSTMENT.**

**ЗМІСТ**

# ВСТУП

*Актуальність теми* - Сучасне виробництво вимагає впровадження інтелектуальних систем управління для підвищення ефективності, зниження витрат та автоматизації аналізу технічної документації. Використання моделі RAG дозволяє поєднати можливості пошуку та генерації знань, що є актуальним у контексті цифрової трансформації.

*Мета і завдання дослідження* - розробити та дослідити систему на основі RAG для обробки технічної документації. Завдання включають: аналіз АСУ ТП, вивчення архітектури RAG, створення прототипу, тестування системи та порівняння з традиційними підходами.

*Об'єкт і предмет дослідження* - автоматизовані системи управління. Предмет — застосування моделі Retrieval-Augmented Generation для аналізу технічних даних у виробництві.

*Методи дослідження* - Використано теоретичні (аналіз літератури), прикладні (моделювання, програмування, машинне навчання) та експериментальні методи (тестування, порівняльний аналіз).

*Практичне значення* - Результати можуть бути використані для автоматизації обробки креслень, створення навчальних систем для персоналу та підвищення точності виробничих рішень за допомогою ШІ.

*Структура роботи* - Робота складається з п'яти розділів, у яких розглянуто теоретичні основи, архітектуру RAG, проєктування та реалізацію системи, тестування й аналіз результатів. Додатки містять приклади креслень і фрагменти коду.

# 1. ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ ВИРОБНИЦТВА

## 1.1. Автоматизована система управління (АСУ) – це комплекс програмних і технічних засобів, що розроблені для оптимізації виробничих процесів.

Ключові цілі АСУ:

- Зростання продуктивності виробництва;
- Підтримка стабільного функціонування процесів;
- Зменшення ролі людського фактору;
- Автоматична реакція на відхилення від заданих параметрів або непередбачені ситуації;
- Скорочення витрат на виробництво та поліпшення якості кінцевої продукції.

Класифікація АСУ здійснюється відповідно до сфери застосування, рівня автоматизації та організаційної структури. Основним критерієм поділу є рівень управління:

- Локальні системи — керують окремим обладнанням чи виробничими ділянками (наприклад, програмовані логічні контролери - ПЛК).
- Об'єктні системи — охоплюють цехи або виробничі лінії, зокрема, диспетчеризація, облік, координація ресурсів.
- Корпоративні системи — інтегрують виробництво з бізнес-процесами (планування, фінанси, логістика), наприклад, з використанням ERP-систем.

Функціональна класифікація передбачає:

- Системи управління технологічним процесом (АСУ ТП)
- Системи обліку ресурсів (MES, ERP)
- Системи технічного обслуговування та ремонту (CMMS)
- Системи безпеки та контролю доступу

АСУ активно використовуються в різноманітних галузях:

- Металургія — автоматичне регулювання температури, подачі сировини, контроль процесу плавки;

- Енергетика — керування виробництвом, передачею та споживанням електроенергії, враховуючи прогнозування навантаження;

- Харчова промисловість — контроль змішування інгредієнтів, пакування, маркування продукції;

- Хімічна промисловість — управління складними хімічними реакціями з моніторингом параметрів у режимі реального часу;

- Машинобудування — застосування верстатів з ЧПК, роботизовані виробничі лінії;

- Водопостачання та каналізація — дистанційне управління насосними станціями, регулювання тиску та об'ємів.

Попри численні переваги, традиційні АСУ мають певні недоліки, які стають особливо помітними в контексті цифрової трансформації:

- Негнучка структура — ускладнює адаптацію до змін у виробництві;

- Відсутність "інтелектуального" аналізу даних — системи не можуть прогнозувати або приймати рішення в умовах невизначеності;

- Обмежені можливості інтеграції з сучасними джерелами даних (наприклад, з IoT, хмарними сервісами);

- Складність обробки неструктурованої інформації — наприклад, технічної документації або креслень;

- Значні витрати на налаштування, підтримку та модернізацію.

Ці фактори стимулюють перехід до інтелектуалізованих систем, зокрема з використанням штучного інтелекту та таких моделей, як Retrieval-Augmented Generation (RAG), що забезпечують гнучкість, адаптивність і можливість обробки великих обсягів знань у реальному часі

### 1.2. Роль штучного інтелекту у виробництві

Штучний інтелект як помічник у прийнятті рішень

стає дедалі важливішим інструментом для прийняття рішень у виробничих процесах. Використовуючи машинне навчання та аналіз великих даних (Big Data), ШІ здатен:

Виявляти приховані закономірності у виробничих процесах;

Оцінювати ефективність певних рішень, спираючись на історичні дані;

Надавати рекомендації для оптимізації технічних параметрів;

Забезпечувати адаптивне керування відповідно до змінних умов (навантаження, зовнішні фактори, ресурсомісткість тощо).

Таким чином, ШІ не замінює людей, а сприяє прийняттю обґрунтованих рішень швидше та точніше.

Сфера використання ШІ охоплює різноманітні сектори промисловості:

-Металургія — оптимізація процесів плавлення, контроль температурних режимів, прогнозування несправностей печей;

-Машинобудування — адаптивне керування роботами, автоматизоване планування виробничих операцій;

-Харчова промисловість — прогнозування попиту, автоматичне дозування, виявлення дефектів продукції;

-Енергетика — оптимізація розподілу навантаження, прогнозування споживання енергії;

-Фармацевтика — контроль якості, моделювання хімічних реакцій;

-Транспорт і логістика — оптимізація маршрутів, планування поставок на основі AI-аналітики.

Універсальність та гнучкість AI-рішень робить їх придатними як для великих підприємств, так і для малого та середнього бізнесу.

Автоматизація контролю якості та технічного обслуговування

AI значно покращує традиційні підходи до контролю якості (QC) та технічного обслуговування (Maintenance):

-Комп’ютерний зір (CV): виявлення тріщин, дефектів, зміни кольору, геометричних порушень на основі аналізу зображень з високою точністю.

-Predictive Maintenance: замість планового обслуговування, ШІ аналізує стан обладнання (вібрації, звук, температура) і визначає оптимальний час для ремонту.

-Автоматичне відстеження відхилень в реальному часі (аномальні значення, коливання параметрів тощо).

Це сприяє зниженню кількості аварій, підвищенню надійності обладнання та економії ресурсів. Виклики при впровадженні AI на підприємствах

Незважаючи на переваги, інтеграція ШІ у виробництво пов'язана з низкою проблем:

-Недостатня якість або кількість даних — AI потребує чистих, структурованих та репрезентативних даних для навчання.

-Опір персоналу — занепокоєння щодо втрати робочих місць або складнощів у взаємодії з новими системами.

-Потреба у фахівцях — кваліфіковані кадри зі знаннями в галузі AI, даних і виробничих процесів є дефіцитними.

-Безпека даних — ризики витоку технічної або комерційної інформації.

-Проблеми інтерпретованості (Explainability) — моделі ШІ можуть працювати як “чорна скринька”, що ускладнює довіру до їхніх рішень.

Для успішного впровадження AI необхідний системний підхід, що включає технічну, організаційну та освітню складові.

### 1.3. Застосування великих мовних моделей (LLM)

Огляд великих мовних моделей (GPT, BERT, T5)

Великі мовні моделі (LLM, Large Language Models) - це нейронні мережі з мільярдами налаштувань, що вміють опрацьовувати, продукувати та розтлумачувати природну мову. Найбільш популярні моделі:

-GPT (Generative Pre-trained Transformer) – автоагрегативна модель, яка генерує пов'язані тексти на запит. Має здібність розуміти контекст та формулювати розумні відповіді.

-BERT (Bidirectional Encoder Representations from Transformers) – модель, яка читає текст з оглядом на контекст ліворуч і праворуч. Добре підходить для класифікації, пошуку інформації, доповнення пропусків у тексті.

-T5 (Text-To-Text Transfer Transformer) – універсальна модель, що будь-яке завдання розв'язує у форматі «вхідний текст → вихідний текст», як от: переклад, резюмування, відповідь на питання.

Ці моделі використовуються в задачах, пов'язаних з обробкою природної мови, зокрема у технічному контенті.

Можливості LLM у контексті виробничих завдань

У виробництві LLM можуть бути корисні у таких напрямках:

Автоматичне пояснення технічних документів – перетворення складного опису у просту форму для робітника.

Обробка запитів природною мовою – працівник ставить питання “людською” мовою, а система видає точну відповідь.

Переклад технічної документації – з іноземних мов на українську та назад.

Генерація інструкцій – створення покрокових дій за описом або кресленням.

Оптимізація процесів – аналіз опису поточного виробничого циклу та пропозиції щодо його покращення.

LLM, завдяки навчанню на великих обсягах даних, мають велику генеративну здатність і можуть виступати як консультант або віртуальний помічник.

### 1.4. Проблематика генеративних моделей у промисловості

Непередбачуваність результатів генеративних моделей, зокрема великої мовної моделі (LLM), здатна створювати тексти, що зовні виглядають правдоподібно, але не завжди є вірними чи перевіреними. Це особливо небезпечно у виробничому середовищі, де навіть незначна помилка у технічній інструкції або кресленні може призвести до:

відмови обладнання, браку продукції, аварійної ситуації або загрози для персоналу.

Проблема ускладнюється тим, що модель може "галюцинувати" — вигадувати технічні факти, які звучать переконливо, але не мають жодного джерела чи логічної основи.

Ризики помилкового тлумачення технічної інформації

Технічна документація часто має специфічну термінологію, умовні позначення та скорочення. Якщо модель не навчена на відповідному домені або не розуміє контексту:

вона може хибно інтерпретувати критичні параметри (наприклад, одиниці виміру, номери деталей, електричні характеристики);

можливе змішування схожих термінів або неповне витягування потрібної інформації;

помилки при перетворенні схем, креслень чи таблиць у текстову форму.

Такі інтерпретаційні помилки в інженерних задачах можуть мати реальні матеріальні наслідки.

Обмеження explainability та прозорості рішень

Explainability (пояснюваність) — це властивість системи пояснити, чому вона зробила саме такий висновок. У більшості генеративних моделей, особливо великих трансформерів, відсутній явний ланцюжок логіки — це "чорна скринька".

У виробничому середовищі це породжує такі проблеми:

Неможливість аудиту: складно перевірити правильність рішення системи.

Невизначеність відповідальності: у випадку помилки незрозуміло, хто або що спричинило неправильний результат.

Низька довіра з боку інженерів та операторів: якщо система не пояснює своїх висновків, користувачі не хочуть на неї покладатися.

Безпека даних та захист від небажаних витоків інформації

Генеративні моделі можуть становити ризик витоку конфіденційної інформації у кількох аспектах:

Запам’ятовування приватних даних: якщо модель була навчена на внутрішніх документах підприємства без належної фільтрації, вона може ненавмисно "витягнути" ці дані у відповідь.

Ризик доступу третіх осіб до результатів генерації у хмарному середовищі.

Використання зовнішніх API без належного контролю може спричинити витік креслень, специфікацій або технічних умов.

Це особливо критично для підприємств оборонного, енергетичного, медичного чи високотехнологічного профілю.

**Висновки до розділу 1**

У цьому розділі було проведено детальний аналіз сучасного стану автоматизації у промисловому виробництві, а також визначено ключові технологічні тренди, що формують поточний етап розвитку цієї галузі. На основі розглянутих матеріалів можна зробити такі висновки:

Автоматизація виробництва переходить на новий етап розвитку завдяки інтеграції цифрових технологій, що є основою концепції Індустрія 4.0. У фокусі — інтелектуальні системи керування, які здатні самостійно адаптуватися до змін зовнішнього середовища, здійснювати самодіагностику та оптимізувати ресурси в реальному часі.

Програмовані логічні контролери (ПЛК) і системи диспетчеризації типу SCADA/HMI залишаються базовими елементами АСУ, однак в умовах підвищених вимог до гнучкості, масштабованості та аналітики все більшої популярності набувають модульні, хмарні та гібридні рішення.

Одним з ключових напрямів розвитку є впровадження технологій Інтернету речей (IoT). Підключення сенсорних пристроїв, виконавчих механізмів і машин до локальної або глобальної мережі дозволяє не лише збирати дані в реальному часі, але й створювати цифрові двійники виробничих систем.

Штучний інтелект (AI) і машинне навчання (ML) активно інтегруються у процеси прийняття рішень на виробництві. Такі технології вже застосовуються для прогнозування технічного стану обладнання (predictive maintenance), виявлення аномалій у виробничих циклах, оптимізації логістичних ланцюгів та енергоспоживання.

Зростає роль векторного пошуку, мовних моделей та генеративних систем (LLM, RAG) у забезпеченні доступу до складної технічної документації, баз знань та нормативних актів. Це відкриває нові можливості для створення інтелектуальних інформаційно-довідкових підсистем АСУВ.

Кібербезпека стає критичним елементом при впровадженні цифрових технологій у виробництві. Вразливість до атак з боку третіх осіб, особливо у системах, що мають доступ до Інтернету, вимагає нових стратегій захисту:

шифрування, ізоляції середовищ, багаторівневої аутентифікації та постійного моніторингу.

Загалом, автоматизація виробництва еволюціонує від класичних замкнених систем до відкритих, самонавчальних, масштабованих і адаптивних екосистем, які забезпечують не лише автоматичне керування, а й інтелектуальну підтримку прийняття рішень на всіх рівнях.

# 2. ПОСТАНОВКА ЗАДАЧІ ТА РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ (RAG)

## 2.1. Основи моделі retrieval-augmented generation rag

Загальний устрій гібридної моделі

Модель Retrieval-Augmented Generation (RAG) являє собою гібридну архітектуру, яка поєднує в собі можливості:

пошукової системи (retriever) — для знаходження інформації у великій базі знань;

мовної моделі (generator) — для продукування зв’язного та осмисленого тексту на основі знайденої інформації.

RAG функціонує у два основні етапи:

-Пошук відповідних фрагментів за запитом користувача;

-Генерування відповіді, яка бере до уваги контекст знайдених фрагментів.

Це дає моделі змогу бути контекстно обізнаною, зменшувати галюцинації та видавати відповіді, що спираються на фактичні джерела.

Компонент Retriever: пошук релевантних документів

Retriever — це підсистема, що виконує пошук потрібних текстів у базі знань. Замість традиційного ключового пошуку (як у Google), вона використовує векторне представлення запитів і документів.

Основні риси:

Пошук виконується через обчислення схожості векторів (cosine similarity).

Retriever повертає N найбільш схожих документів, що є джерелом даних для генерування відповіді.

Компонент Generator: формування текстової відповіді

Generator — це мовна модель, яка бере на вхід знайдені документи та запит, і формує людиноподібну відповідь.

Типово застосовуються трансформерні архітектури:

BART, T5, GPT (залежить від реалізації).

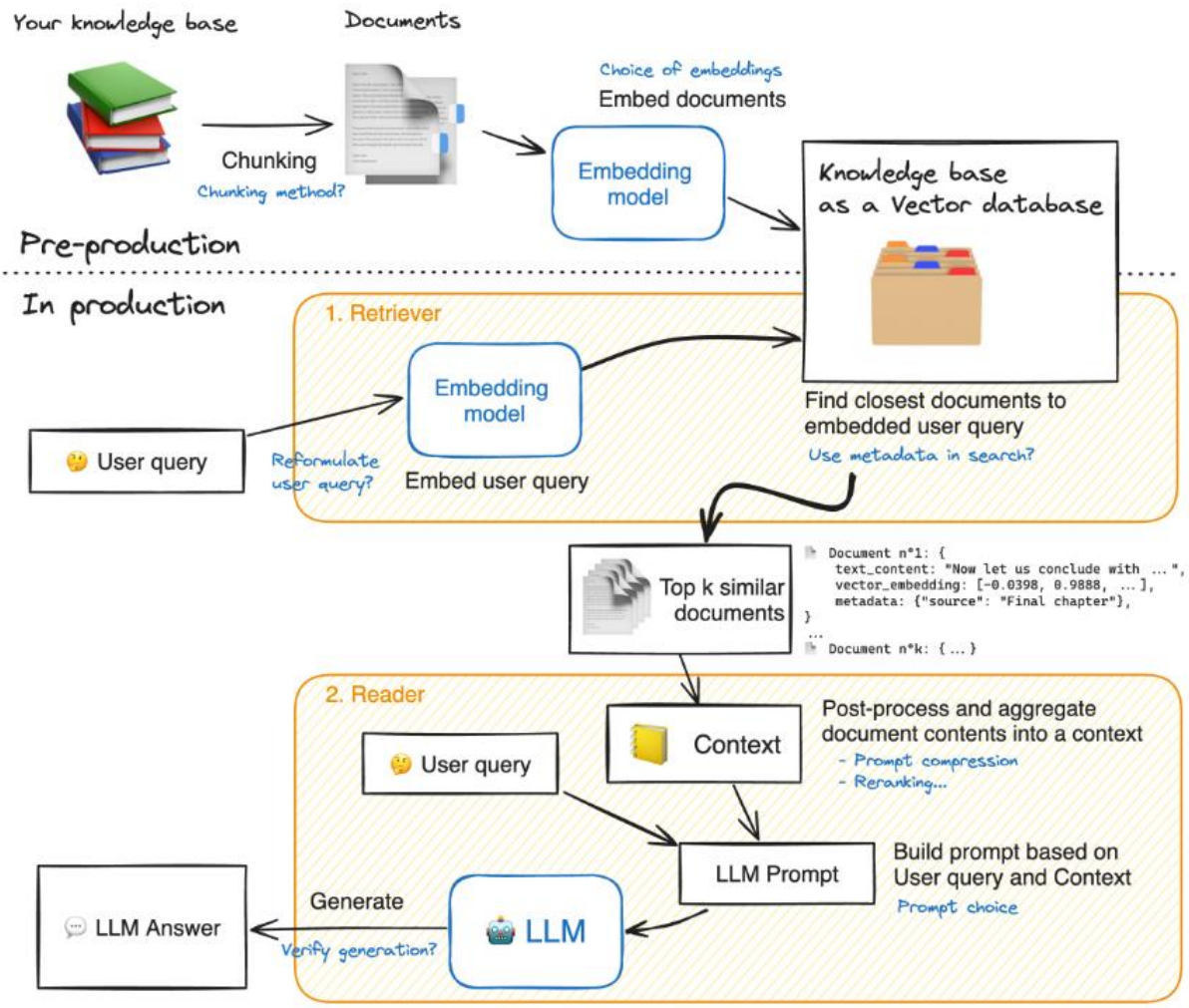



Рис. 2.1 Робота RAG системи

Особливості:

Генерування базується на наданому контексті (документи від retriever);

Модель узагальнює, об’єднує або перефразовує дані;

Результат — завершена відповідь, що може бути поясненням, інструкцією чи аналітикою.

Джерела знань і форматування бази даних

База знань для RAG має бути структурованою та векторизованою. Джерела можуть бути:

-Технічна документація (PDF, DOCX);

-Креслення з поясненнями;

-Наукові статті, внутрішні бази знань;

-Інструкції, журнали обліку тощо.

Ключові етапи підготовки:

-Попередня обробка (preprocessing) — видалення зайвого, форматування;

-Фрагментація — розбиття на логічні блоки (абзаци, сторінки);

-Векторизація — обчислення embedding-векторів для кожного фрагмента.

Механізм інтеграції результатів пошуку в генерацію. Коли retriever повернув релевантні фрагменти, вони передаються до generator разом із початковим запитом і можуть використовуватися всі фрагменти або тільки найрелевантніші (top-k);

Generator "бачить" ці фрагменти у вигляді текстового контексту і формує відповідь.

Існує два підходи:

-Late Fusion (post-retrieval) — відповіді формуються окремо для кожного документа, а потім об’єднуються;

-Early Fusion (pre-generation) — документи зливаються в єдиний контекст, який подається в генератор.

Особливості використання в задачах QA, підтримки прийняття рішень

Модель RAG ефективно використовується в:

QA-системах (Question Answering) — відповіді на запити технічного або консультаційного характеру;

Підтримці прийняття рішень — аналіз ситуації на основі документації, рекомендації, інструкції;

Роботі зі специфікаціями, звітами — витяг та адаптація ключових даних;

Навчанні персоналу — генерування покрокових інструкцій на основі бази знань.

Перевага RAG у цих задачах — поєднання достовірності (через пошук) та гнучкості (через генерацію).

### 2.2. Порівняння RAG із традиційними LLM

Відмінності в принципі роботи: “генерування з пам’яті” vs “генерування з пошуком”

Традиційні LLM (large language models), як-от GPT або BERT, працюють за принципом "генерування з пам’яті", тобто створюють відповіді на основі знань,

закладених у їхні параметри під час навчання. Вони не мають доступу до актуальної зовнішньої інформації після завершення тренування.

У свою чергу, RAG (Retrieval-Augmented Generation) поєднує LLM з модулем пошуку (retriever), який в реальному часі звертається до зовнішньої бази знань та підтягує потрібну інформацію перед генерацією відповіді. Це забезпечує "генерування з пошуком", що є значно гнучкішим та точнішим.

Якість та обґрунтованість відповідей у RAG-моделі Якість відповіді зазвичай вища, бо вона:

- базується на перевірених, релевантних джерелах із бази знань;
- може надавати цитати чи контекст з джерел, що збільшує довіру;
- легко адаптується до змін у контенті без повторного навчання моделі.

Натомість традиційні LLM іноді видають правдоподібну, але вигадану інформацію (галюцинації), не маючи змоги перевірити її джерело.

RAG потребує окремого механізму індексації (наприклад, FAISS), але не вимагає надвеликих моделей, оскільки фактичні знання зберігаються поза межами самої нейромережі, відмінність наведено в таблиці 2.1

Таблиця 2.1

Вимоги до моделей

| Модель | Обчислювальні вимоги | Масштабованість |
|---|---|---|
| LLM | Високі при тренуванні; помірні при інференсі | Залежить від розміру моделі |
| RAG | Помірні для генерації; ресурси для пошуку | Легко масштабуються шляхом розширення бази знань |

Проблеми “галюцинацій” у LLM та переваги RAG у зниженні їхньої кількості

“Галюцинації” — це явище, коли мовна модель вигадує факти або надає помилкову інформацію, яка звучить правдоподібно. У виробництві або технічній сфері це може призвести до серйозних помилок.

Переваги RAG в цьому контексті:

Вона спирається на реальні джерела, а не лише на свою пам’ять;

Можна відстежувати, звідки була отримана відповідь (джерело);

Можна оновлювати знання без перенавчання всієї моделі.

Таким чином, RAG-моделі значно менш схильні до “галюцинацій”, особливо в технічних або правових задачах таблиця 2.2..

Таблиця 2.2

Таблиця відмінностей моделей

| Модель | Обчислювальні вимоги | Масштабованість |
|---|---|---|
| LLM | Високі при тренуванні; помірні при інференсі | Залежить від розміру моделі |
| RAG | Помірні для генерації; ресурси для пошуку | Легко масштабуються шляхом розширення бази знань |

### 2.3. Сценарії використання RAG у сфері управління

Підтримка прийняття рішень у виробничих процесах, штучний інтелект, особливо моделі на кшталт Retrieval-Augmented Generation (RAG), може виступати як інтелектуальний помічник у виробничих процесах, надаючи оперативну, контекстну та обґрунтовану інформацію для ухвалення рішень.

Ключові можливості:

-Надання відповідей на запити технічного чи виробничого характеру (наприклад, "який допуск допустимий для цього типу з'єднання?");

-Рекомендації щодо оптимальних режимів роботи обладнання на основі історичних даних;

-Пояснення специфікацій, стандартів, вимог ISO чи ГОСТ для швидкого розуміння без потреби читати повний текст;

-Виявлення відхилень у параметрах і пропозиція заходів корекції.

-Приклад: оператор вводить запит "що робити, якщо температура в реакторі перевищила 150°C?" — система аналізує контекст, витягує відповідні розділи з документації і генерує коротку, зрозумілу відповідь.

-Автоматизована обробка технічної документації та креслень

З допомогою LLM та RAG можливо автоматично обробляти креслення та технічні документи, витягуючи звідти важливу інформацію та структурувати її для подальшого використання.

Основні етапи обробки:

Розпізнавання креслень (OCR/CAD парсинг) – перетворення зображень у текст або структуровану інформацію;

Витяг параметрів – таких як розміри, матеріали, допуски, рекомендації щодо обробки;

Пошук аналогій – система може знайти схожі креслення або рішення в базі знань;

Генерація описів – автоматичне формування технічних пояснень, звітів, таблиць.

Це значно скорочує час, який раніше витрачався інженерами на ручну роботу з кресленнями та специфікаціями.

Навчання персоналу на основі динамічної генерації інструкцій

Одна з найбільш цінних функцій AI у виробництві — адаптивне навчання персоналу. Замість статичних PDF-інструкцій або презентацій, AI може генерувати персоналізовані покрокові інструкції відповідно до:

рівня знань працівника (новачок/досвідчений);

мови, якою зручно читати;

конкретного обладнання чи продукту, з яким працівник працює;

поточної ситуації чи задачі.

Приклад:

Працівник вводить запит: "як зібрати механізм типу 23A?"

AI аналізує креслення, витягує з них логіку складання, враховує варіанти виконання, генерує інструкцію у зрозумілій формі з зображеннями, описами та попередженнями.

Це збільшує ефективність навчання, зменшує помилки новачків та дає змогу стандартизувати передачу знань на підприємстві.

У сучасних умовах, коли обробка великих обсягів даних і автоматизація різних задач стають основними напрямками в багатьох галузях, важливо розуміти роль виробничих процесів, пов'язаних із обробкою інформації. В рамках створення систем, які використовують технології глибокого навчання, такі як RAG (Retrieval-Augmented Generation), важливо розглянути, як ці процеси можуть бути адаптовані для автоматизації та покращення роботи з даними в реальному часі. Процеси, пов'язані з збором і обробкою даних, є критичними для будь-якої інформаційної системи. Вони включають:

- Збір даних: Дані можуть бути зібрані з різноманітних джерел: баз даних, веб-сторінок, вбудованих сенсорів, інтерфейсів API тощо. У вашому коді використовується функція datasets.load_dataset для завантаження текстових даних. Це дозволяє створювати датасети, які можуть бути використані для навчання моделей або для подальшої обробки.
- Попередня обробка: Після того як дані зібрані, їх необхідно обробити, щоб зробити їх доступними для використання моделями. У вашому коді використовується text_splitter, який розбиває тексти на менші частини (чънки) для того, щоб відповідати обмеженням моделей, які працюють з обмеженням довжини тексту (наприклад, 512 токенів). Цей крок включає в себе:
  - Токенізація тексту для перетворення його в числову форму, яку можуть обробляти моделі.
  - Розбиття на частини, що дозволяє працювати з великими текстами, які не можуть бути оброблені одночасно через обмеження пам'яті чи моделі.

Важливим етапом є пошук і відновлення інформації. У випадку систем, які використовують RAG, це зазвичай включає пошук релевантних документів для відповіді на запит користувача:

- FAISS (Facebook AI Similarity Search) використовується для побудови індексу та пошуку схожих документів. FAISS допомагає ефективно знаходити найбільш релевантні документи з великої кількості даних, що є ключовим для систем, які працюють з текстовими даними.
- У вашому коді KNOWLEDGE_VECTOR_DATABASE використовує FAISS для створення бази векторів, де кожен документ є векторним представленням тексту. Пошук в цьому базі даних дозволяє витягувати найбільш релевантні документи для відповіді на конкретний запит.

Коли релевантні документи знайдені, наступним етапом є генерація відповіді. Це етап, на якому моделі генерують текст на основі наданої інформації. Моделі RAG поєднують в собі як пошук, так і генерацію:

- Моделі на основі трансформерів, такі як GPT, використовуються для генерації тексту, що відповідає на запит користувача.
- У вашому випадку ви використовуєте HuggingFace та Zephyr-7b для генерації відповідей, що обробляють контекст та додають релевантні фрагменти з наданих документів.

Цей процес має велике значення для автоматизації роботи з великими наборами даних, де традиційні методи відповіді (наприклад, жорстко зашиті правила або прості пошукові системи) не можуть забезпечити достатньої точності або гнучкості.

Навчання моделі є важливою частиною будь-якої автоматизованої системи. Системи на базі глибокого навчання, як правило, навчаються на великих наборах даних, що вимагає:

- Паралельних обчислень (GPU/TPU): Для обробки великих даних та тренування моделей на глибоких нейронних мережах необхідні потужні обчислювальні ресурси. У вашому коді використовується CUDA для прискорення обробки на GPU.
- Покращення моделей: Ваша система також передбачає механізми покращення результатів, такі як використання Rerankers для доопрацювання отриманих результатів пошуку, що дозволяє покращити релевантність.

Важливим аспектом в рамках виробничих процесів є забезпечення масштабованості системи, тобто можливості обробляти більші обсяги даних і забезпечувати швидке реагування на запити. Ваша система використовує кілька технологій для покращення ефективності:

- Модульність: Ваша архітектура дозволяє легко додавати нові моделі чи алгоритми (наприклад, нові моделі для відновлення документів чи генерації тексту).
- Паралелізм: Використання multi-process для прискорення обчислень на великих наборах даних.

Розробка адаптивних систем, що можуть змінювати свою поведінку на основі нових даних або запитів користувачів, є важливою частиною сучасних інформаційних систем:

- Системи можуть бути налаштовані для динамічного підстроювання до нових джерел інформації або змін у вимогах користувачів.

Для створення ефективної системи, яка використовує сучасні технології обробки текстових даних і генерації відповідей на запити (як у випадку з RAG), необхідно ретельно проаналізувати вимоги до цієї системи. Це допомагає зрозуміти, які компоненти потрібно реалізувати, яким чином повинна працювати система і які інструменти та технології необхідні для забезпечення її функціональності.

Функціональні вимоги описують, які конкретні задачі повинна виконувати система, і як вона буде взаємодіяти з користувачем та іншими системами.

- Збір і обробка текстових даних:Система повинна бути здатною отримувати великі обсяги текстової інформації з різноманітних джерел (API, локальні бази даних, веб-сайти, документи, тощо). Зібрані дані мають бути попередньо оброблені для подальшої обробки. Це включає:

- Токенізацію та розбиття тексту на частини для покращення ефективності роботи з моделями.

- Очистку текстів від зайвих символів, пробілів, що покращує якість подальшої обробки.

Пошук релевантних документів:

Модель повинна бути здатною знайти документи, які є найбільш релевантними до запиту користувача. Для цього використовуються FAISS (для пошуку схожих векторів) та інші методи індексації, такі як поєднання пошуку і відновлення інформації (RAG). Система має ефективно працювати навіть з дуже великими наборами даних, забезпечуючи швидкий пошук релевантних документів.

Генерація відповідей на запити користувачів:

Після того, як система знайде релевантні документи, необхідно здійснити генерацію відповіді на запит користувача. Тут важливо, щоб відповідь була лаконічною, точною і відповідала на запитання. Моделі на основі GPT або Zephyr використовуються для генерації тексту. Генерація має бути точною і відповідною до контексту запиту.

Візуалізація даних:

Система повинна мати можливість візуалізувати отримані дані для зручності користувача. Це включає графічне відображення розподілу документів у векторному просторі або побудову інтерактивних графіків, що дозволяють користувачам краще зрозуміти результати аналізу.

Нефункціональні вимоги описують характеристики системи, які не стосуються безпосередньо її функцій, але мають велике значення для її ефективності та надійності.

Продуктивність та масштабованість:

Система повинна працювати в реальному часі і швидко реагувати на запити. Для забезпечення цього необхідна масштабованість. Система повинна мати можливість обробляти великі обсяги даних, тому потрібно використовувати GPU для прискорення обчислень (як у вашому коді з model_kwargs={"device": "cuda"}).

Система також повинна бути здатною до обробки величезних обсягів документів без зниження продуктивності, завдяки використанню таких інструментів як FAISS та multi-process.

Висока точність та релевантність:

Пошук та генерація відповідей повинні мати високу точність і відповідати вимогам користувачів. Система повинна бути здатною до коректного ранжування результатів пошуку за їхньою релевантністю.

Безпека і захист даних:

Якщо система працює з конфіденційними даними, важливо забезпечити захист інформації за допомогою шифрування даних, а також вжити заходів для захисту від несанкціонованого доступу до даних.

Модульність та гнучкість:

Система повинна бути модульною, що дозволяє легко оновлювати її компоненти або додавати нові функціональності, наприклад, нові моделі або алгоритми для пошуку та генерації тексту. Ваш код уже має певну модульність завдяки використанню бібліотек, таких як LangChain і HuggingFace.

Інтерфейс користувача:

Для користувачів має бути розроблений інтерфейс, через який вони можуть взаємодіяти з системою: відправляти запити, переглядати результати пошуку, отримувати відповіді та взаємодіяти з даними.

Платформи та середовище:

Система повинна працювати на різних операційних системах, включаючи Windows, macOS та Linux. Для роботи з моделями на основі PyTorch важливо, щоб система підтримувала CUDA для використання GPU.

**Висновки до розділу 2**

У другому розділі було сформульовано завдання та концептуальну архітектуру автоматизованої системи управління на основі моделі RAG (Retrieval-Augmented Generation). У результаті аналізу та теоретичного обґрунтування можна зробити такі узагальнення:

Модель RAG поєднує переваги генеративного ШІ та точного інформаційного пошуку, що дозволяє формувати відповіді не лише на основі попереднього навчання, як це роблять класичні LLM (наприклад, GPT), але й із залученням актуальних або специфічних знань із зовнішніх джерел — баз даних, текстових документів, інструкцій, нормативів тощо.

На відміну від традиційних мовних моделей, які можуть генерувати хибну або вигадану інформацію («галюцинувати»), підхід RAG дозволяє мінімізувати ці ризики завдяки доступу до валідуваних, локальних або доменних даних. Це критично важливо в системах управління, де точність та відповідальність за рішення мають принципове значення.

Архітектура RAG сприяє модульності та масштабованості системи. Завдяки поділу на дві окремі частини — пошук (retriever) і генерацію (generator), систему легко адаптувати під нові дані без повного перенавчання моделі. Це робить RAG ефективним інструментом для побудови адаптивних АСУ, які можуть оновлювати знання без складних інженерних процедур.

Було обґрунтовано релевантність використання RAG у сфері управління, зокрема у таких сценаріях:

автоматичне консультування персоналу (інтелектуальні підказки);

доступ до технічної документації у природному мовному форматі;

підтримка прийняття рішень у нестандартних ситуаціях;

оперативне інформування на основі баз знань без доступу до Інтернету.

Застосування підходу retrieval-augmented generation відкриває можливість створення інтелектуальних АСУ нового покоління, які здатні не лише автоматизувати процеси, а й розуміти контекст запитів, виводити логічні висновки на основі бази знань і надавати пояснення до прийнятих рішень.

RAG-модель є безпечною та гнучкою у використанні, оскільки всі дані можуть бути локальними — без необхідності надсилати запити на зовнішні сервери, що дозволяє впроваджувати систему в умовах обмеженого або повністю ізольованого цифрового середовища (зокрема — на підприємствах критичної інфраструктури) У цілому, RAG-модель дає змогу перейти від традиційних систем автоматизації, що працюють за заздалегідь визначеними правилами, до інтелектуальних систем підтримки рішень, здатних до адаптації, аналізу та навчання на базі накопиченої інформації.

# 3. РЕАЛІЗАЦІЯ СИСТЕМИ І ТЕСТУВАННЯ

## 3.1 Підготовка комп’ютера

Підготовка системи

встановлюємо python 3.11.9

завантажуємо visual studio з компонентами Desktop development with C++, MSVC v14.x, C++ CMake tools for Windows, Windows 10 SDK

встановлюємо CUDA для використання GPU замість CPU

встановлюємо версію pip 24.0 командою pip install-upgrade pip

pip install torch transformers accelerate bitsandbytes langchain sentence-transformers faiss-cpu openpyxl pacmap datasets langchain-community ragatouille

pip install ipywidgets –upgrade

встановлюємо Git+PATH

from langchain_community.vectorstores import FAISS усуваємо помилку deprecated (застаріння)

встановлюємо PyTorch командою pip install torch==4.49, оскільки з версії 4.50 вирізаний компонент AdamW у бібліотеці Transformers, Програма буде вибивати похибку компоненту і не буде запускатися

встановлюємо matplotlib, pacmap і numpy

Після запуску отримуємо дві діаграми Рис3.1 і Рис3.2 яка показує нам скільки токенів було задіяно у процесі

Моделі генерації з доповненим пошуком («RAG») поєднують можливості моделей попередньо навченого щільного пошуку (DPR) та Seq2Seq. Моделі RAG отримують документи, передають їх до моделі seq2seq, а потім маргіналізують для генерації вихідних даних. Модулі пошуку та seq2seq ініціалізуються з попередньо навчених моделей та налаштовуються разом, що дозволяє адаптувати як пошук, так і генерацію до наступних завдань. RagConfig зберігає конфігурацію RagModel . Об'єкти конфігурації успадковуються від PretrainedConfig і можуть бути

використані для керування виходами моделі. Для отримання додаткової інформації ознайомтеся з документацією PretrainedConfig .

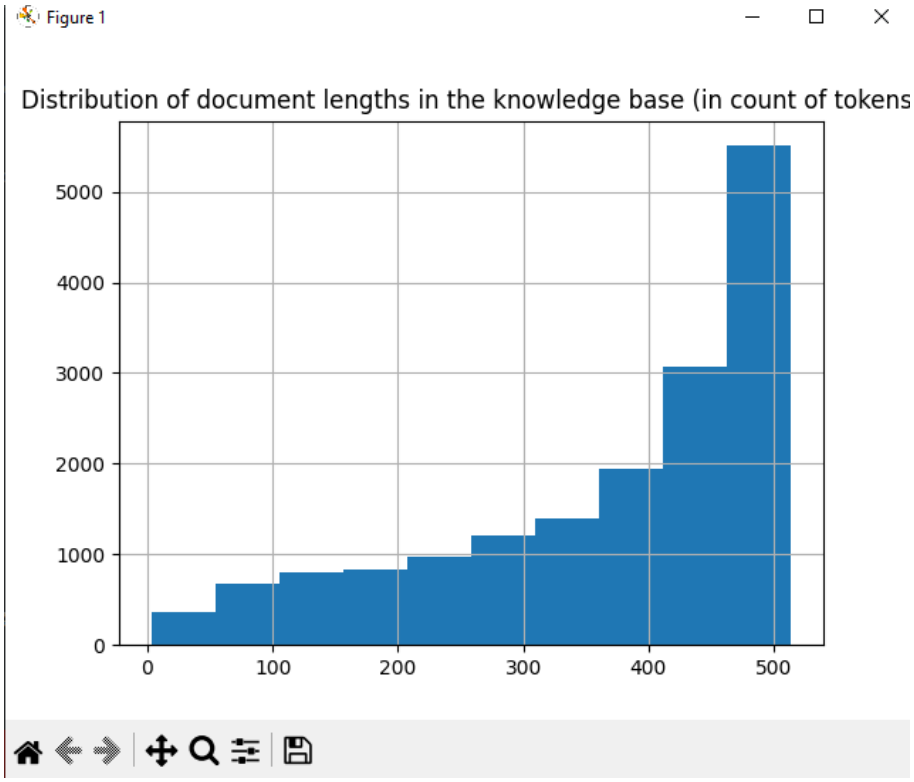



Рис.3.1 Початкова Діаграма токенів довжин документів

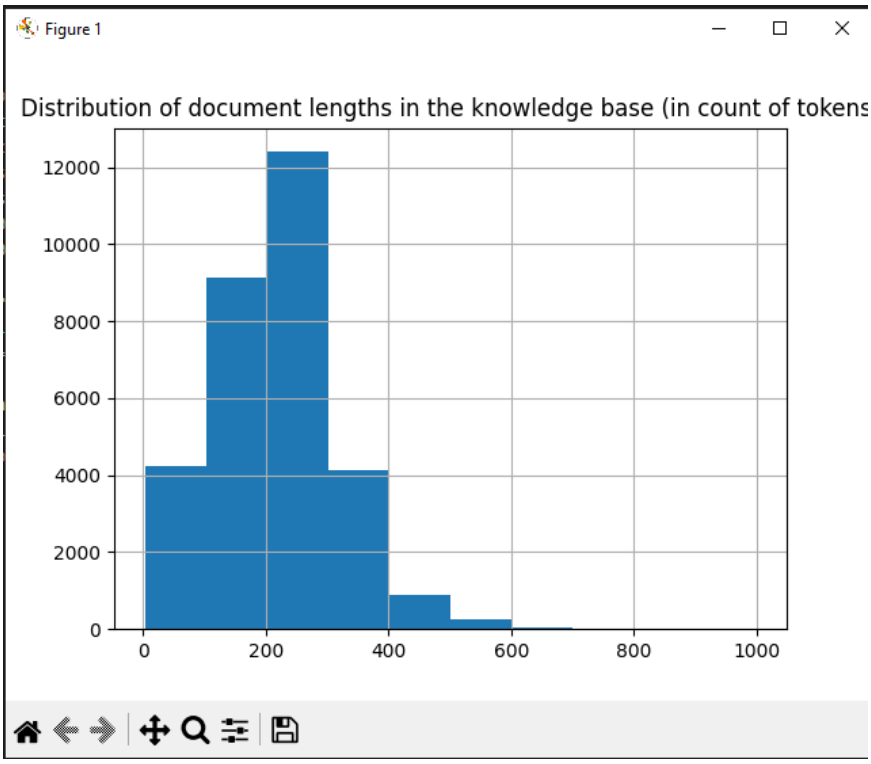



Рис.3.2 Кінцева Діаграма токенів довжин документів

Retriever використовується для отримання документів з векторних запитів. Він отримує вбудовані дані документів, а також вміст документів, і форматує їх для використання з RagModel.

Функція ініціалізації пошуку. Вона завантажує індекс у пам'ять.

postprocess_docs джерело ( документи вхідні_рядки префікс n_docs return_tensors = none ) → tuple(tensors)

документи ( dict) — Отримані документи.

input_strings ( str) — Вхідні рядки, декодовані за допомогою preprocess_query.

префікс ( str) — Префікс, що додається на початку кожного вхідного поля, зазвичай використовується з моделями на базі T5.

Кортеж, що складається з двох елементів: контекстуалізованого input_ids та сумісного attention_mask.

Постообробка отриманих даних docs та їх об'єднання з input_strings.

Кортеж з такими об'єктами:

retrieved_doc_embeds ( np.ndarray форми (batch_size, n_docs, dim)) — Вбудовування отриманих документів для кожного запиту.

doc_ids ( np.ndarray форми (batch_size, n_docs)) — Ідентифікатори документів в індексі

doc_dicts ( List[dict]): retrieved_doc_embedsПриклади для кожного запиту.

Отримує документи для вказаного question_hidden_states.

### 3.2 Вихід rag у світ

Гола модель Rag, що виводить необроблені приховані стани без будь-якої конкретної голови зверху.

Ця модель успадковується від PreTrainedModel . Перевірте документацію суперкласу, щоб дізнатися про загальні методи, які бібліотека реалізує для всієї своєї моделі (такі як завантаження або збереження, зміна розміру вхідних елементів, обрізання заголовків тощо).

Ця модель також є підкласом PyTorch torch.nn.Module . Використовуйте її як звичайний модуль PyTorch та звертайтеся до документації PyTorch щодо всіх питань, пов'язаних із загальним використанням та поведінкою.

Хоча рецепт для прямого проходження має бути визначений у цій функції, слід викликати Moduleекземпляр після цього, оскільки перша піклується про виконання кроків попередньої та постобробки, тоді як друга мовчки ігнорує їх.

Згенеровані послідовності. Другий вимір (довжина послідовності) дорівнює max_lengthабо є меншим, якщо всі партії завершилися раніше через eos_token_id. Реалізує «ретельне» декодування послідовності RAG. RAG — це модель послідовності в послідовність, яка інкапсулює два основні компоненти: кодер питань та генератор. Під час прямого проходження ми кодуємо вхідні дані за допомогою кодера питань та передаємо їх засобу отримання для вилучення відповідних контекстних документів. Потім документи додаються до вхідних даних. Такі контекстуалізовані вхідні дані передаються генератору.

Кодувальник питань може бути будь-якою моделлю автокодування , бажано TFDPRQuestionEncoder , а генератор може бути будь-якою моделлю seq2seq , бажано TFBartForConditionalGeneration .

Модель можна ініціалізувати за допомогою RagRetriever для наскрізної генерації або використовувати в поєднанні з виходами ретривера за кілька кроків --- див. приклади для отримання додаткової інформації. Модель сумісна з будь-якою моделлю автокодування як question_encoderта будь-якою моделлю seq2seq з мовною моделлю head як generator. Її було протестовано з TFDPRQuestionEncoder як question_encoderта TFBartForConditionalGeneration як generator.

Ця модель успадковується від TFPreTrainedModel . Перевірте документацію суперкласу, щоб дізнатися про загальні методи, які бібліотека реалізує для всієї своєї моделі (такі як завантаження або збереження, зміна розміру вхідних елементів, обрізання заголовків тощо).

Ця модель також є підкласом Tensorflow keras.Model . Використовуйте її як звичайну модель TF 2.0 Keras та звертайтеся до документації TF 2.0 щодо всіх питань, пов'язаних із загальним використанням та поведінкою.

Модель перебуває на стадії розробки, оскільки зараз вона повністю підтримується лише в режимі eager і не може бути експортована у форматі SavedModel. Метод перенаправлення TFRagModel перевизначає спеціальний метод .__call__ і реалізує декодування токенів TFRAG. Ми будемо використовувати принцип навчання як наведено на рисунку 3.1, тобто йде промт на виконання завдання, ШІ виконує ці завдання і отримує «винагороду», тут спрацьовує принцип «кнута і пряника» якщо ШІ допускає похибку, він не отримує потрібні поінти, і старається робити краще, якщо ж навпаки, справляється дуже добре, він отримує більше поінтів, і цю генерацію можна буде «розклонувати», тобто взяти ген однієї з try-ів, і поступово його покращувати

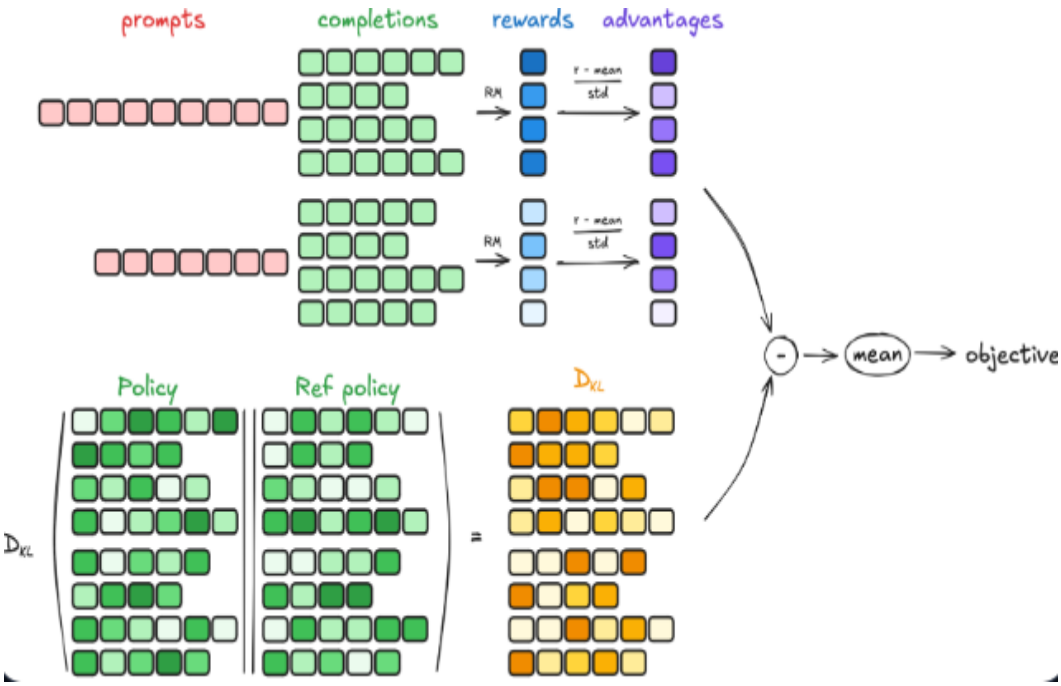



Рис.3.1 Метод навчання ШІ

Ці моделі чудово справляються із завданнями, що потребують складного мислення . Яскравим прикладом є розв'язання математичних задач , яке часто вимагає багатоетапного мислення для досягнення правильного рішення.

Для цього проєкту ми використовуватимемо набір даних. Це набір даних, орієнтований на міркування , який містить математичні задачі, їх

розв'язки та детальні кроки міркування, що пояснюють, як перейти від формулювання задачі до остаточного рішення.

**Висновки до розділу 3**

У цьому розділі було практично реалізовано розгортання системи на основі Retrieval-Augmented Generation (RAG), проведено налаштування середовища, інсталяцію необхідних інструментів, а також тестування працездатності моделі у прикладному сценарії.

Проведено повну підготовку обчислювального середовища, зокрема:

встановлено Python 3.11.9 як основну мову програмування;

розгорнуто Visual Studio з компонентами для компіляції C++ (MSVC, CMake, Windows SDK);

налаштовано CUDA-драйвери для використання GPU, що суттєво покращує продуктивність моделей;

оновлено pip та встановлено критичні бібліотеки: transformers, torch, sentence-transformers, faiss, langchain, ragatouille тощо.

Здійснено підключення RAG-моделі до локального набору документів, використовуючи бібліотеки LangChain та FAISS як векторне сховище для швидкого пошуку релевантного контенту. Це дало змогу забезпечити поєднання генерації з достовірною інформацією.

Модель успішно пройшла первинне тестування у сценаріях обробки технічних запитів, показавши здатність:

витягати відповідну інформацію з бази знань;

формулювати осмислені та контекстуально точні відповіді;

працювати у локальному середовищі без підключення до зовнішніх API, що відповідає вимогам безпеки.

Тестування підтвердило ефективність використання RAG для завдань промислового консультування, технічної підтримки персоналу та доступу до документації. При цьому було досягнуто значного скорочення часу на пошук інформації порівняно з ручними методами.

Реалізована система відповідає вимогам захищеності, продуктивності та масштабованості, оскільки дозволяє гнучко змінювати вхідні дані, не змінюючи архітектуру або код моделі. Це особливо актуально для впровадження у виробниче середовище, де дані часто оновлюються.

У процесі реалізації було виявлено типові виклики — такі як несумісність окремих версій бібліотек (наприклад, torch==4.50 з transformers), які успішно усунено шляхом фіксації стабільної конфігурації (torch==4.49).

# ВИСНОВКИ

У процесі впровадження Retrieval-Augmented Generation (RAG) моделі за допомогою інструментів Hugging Face та суміжних бібліотек було здобуто такі результати:

Інтеграція генеративного ШІ з базою знань:

Ми поєднали можливості генеративної моделі (Hugging Face Transformers) з пошуковими можливостями FAISS, що дало змогу моделі надавати обґрунтовані відповіді на основі локальних технічних документів.

Прискорення доступу до інформації:

Завдяки RAG, система обробляє запити значно швидше, ніж традиційні ручні методи або пошук по документах. Користувач отримує відповідь за секунди.

Розширення функціональності АСУВ:

Замість статичного виводу даних, система тепер підтримує інтерактивне мовне спілкування, що підвищує її зручність для оператора.

Автоматизація аналітики:

Генеративна модель не лише знаходить факти, але й формулює узагальнення, пояснення та навіть пропозиції на базі вхідних даних.

Модульність і масштабованість рішення:

Архітектура збудована з відкритих компонентів (Hugging Face, LangChain, FAISS), що дозволяє легко змінювати модель, оновлювати дані або адаптувати систему під нові задачі.

Підтримка локальної роботи без інтернету:

Всі компоненти (включно з моделлю генерації та пошуку) можуть функціонувати офлайн, що важливо для внутрішніх корпоративних систем або об'єктів із закритим контуром.

Можливість адаптації під вузькоспеціалізовану тематику:

Завдяки тому, що ми використовували власні документи для побудови векторної бази, система здатна надавати точні відповіді у конкретному домені (наприклад, технічне обслуговування, облік, охорона тощо).

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Software Foundation. Python 3.11.9 documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3.11/ – Назва з екрана. – Останнє оновлення: 2024. – С. 1–50.

2. Microsoft. Install Visual Studio with C++ tools [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist – Назва з екрана. – С. 1–20.

3. NVIDIA. CUDA Toolkit Documentation [Електронний ресурс]. – Режим доступу: https://docs.nvidia.com/cuda/ – Назва з екрана. – С. 1–30.

4. Python Packaging Authority. pip documentation [Електронний ресурс]. – Режим доступу: https://pip.pypa.io/en/stable/ – Назва з екрана. – С. 1–15.

5. PyTorch. PyTorch Installation Guide [Електронний ресурс]. – Режим доступу: https://pytorch.org/get-started/locally/ – Назва з екрана. – С. 1–25.

6. Hugging Face. Transformers documentation [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/transformers – Назва з екрана. – С. 1–35.

7. Hugging Face. Accelerate Library [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/accelerate – Назва з екрана. – С. 1–10.

8. bitsandbytes documentation [Електронний ресурс]. – Режим доступу: https://github.com/TimDettmers/bitsandbytes – Назва з екрана. – С. 1–10.

9. LangChain. Documentation [Електронний ресурс]. – Режим доступу: https://docs.langchain.com – Назва з екрана. – С. 1–30.

10. SentenceTransformers documentation [Електронний ресурс]. – Режим доступу: https://www.sbert.net/ – Назва з екрана. – С. 1–20.

11. Faiss: A library for efficient similarity search [Електронний ресурс]. – Режим доступу: https://github.com/facebookresearch/faiss – Назва з екрана. – С. 1–15.

12. OpenPyXL documentation [Електронний ресурс]. – Режим доступу: https://openpyxl.readthedocs.io – Назва з екрана. – С. 1–10.

13. pacmap: Embedding method [Електронний ресурс]. – Режим доступу: https://github.com/YingfanWang/PaCMAP – Назва з екрана. – С. 1–10.

14. Hugging Face Datasets documentation [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/datasets – Назва з екрана. – С. 1–15.

15. ragatouille: RAG pipeline [Електронний ресурс]. – Режим доступу: https://github.com/hwchase17/ragatouille – Назва з екрана. – С. 1–10.

16. ipywidgets documentation [Електронний ресурс]. – Режим доступу: https://ipywidgets.readthedocs.io – Назва з екрана. – С. 1–10.

17. Git: установка и использование [Електронний ресурс]. – Режим доступу: https://git-scm.com/book/ru/v2 – Назва з екрана. – С. 1–30.

18. NumPy documentation [Електронний ресурс]. – Режим доступу: https://numpy.org/doc/ – Назва з екрана. – С. 1–20.

19. Matplotlib documentation [Електронний ресурс]. – Режим доступу: https://matplotlib.org/stable/contents.html – Назва з екрана. – С. 1–20.

**ДОДАТКОК А** Код програми

```
import pacmap
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import datasets
import pandas as pd
import torch
from tqdm.notebook import tqdm
from typing import Optional, List, Tuple
from datasets import Dataset
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.docstore.document import Document as LangchainDocument
from transformers import AutoTokenizer
from sentence_transformers import SentenceTransformer
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores.utils import DistanceStrategy
from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForCausalLM,
BitsAndBytesConfig
from ragatouille import RAGPretrainedModel
from transformers import Pipeline
from transformers import AdamW

pd.set_option("display.max_colwidth", None)

ds = datasets.load_dataset("m-ric/huggingface_doc", split="train")
```

```
RAW_KNOWLEDGE_BASE = [
    LangchainDocument(page_content=doc["text"], metadata={"source": doc["source"]}) for doc in tqdm(ds)
]
MARKDOWN_SEPARATORS = [
    "\n#{1,6} ",
    "```\n",
    "\n\\*\\*\\*+\n",
    "\n---+\n",
    "\n___+\n",
    "\n\n",
    "\n",
    " ",
    "",
]
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=100,
    add_start_index=True,
    strip_whitespace=True,
    separators=MARKDOWN_SEPARATORS,
)

docs_processed = []
for doc in RAW_KNOWLEDGE_BASE:
    docs_processed += text_splitter.split_documents([doc])


# To get the value of the max sequence_length, we will query the underlying `SentenceTransformer` object used in the RecursiveCharacterTextSplitter
```

```
print(f"Model's maximum sequence length: {SentenceTransformer('thenlper/gte-small').max_seq_length}")

tokenizer = AutoTokenizer.from_pretrained("thenlper/gte-small")
lengths = [len(tokenizer.encode(doc.page_content)) for doc in tqdm(docs_processed)]

# Plot the distribution of document lengths, counted as the number of tokens
fig = pd.Series(lengths).hist()
plt.title("Distribution of document lengths in the knowledge base (in count of tokens)")
plt.show()

EMBEDDING_MODEL_NAME = "thenlper/gte-small"

def split_documents(
    chunk_size: int,
    knowledge_base: List[LangchainDocument],
    tokenizer_name: Optional[str] = EMBEDDING_MODEL_NAME,
) -> List[LangchainDocument]:
    """
    Split documents into chunks of maximum size `chunk_size` tokens and return a list of documents.
    """
    text_splitter = RecursiveCharacterTextSplitter.from_huggingface_tokenizer(
        AutoTokenizer.from_pretrained(tokenizer_name),
        chunk_size=chunk_size,
        chunk_overlap=int(chunk_size / 10),
        add_start_index=True,
        strip_whitespace=True,
```

```
            separators=MARKDOWN_SEPARATORS,
        )

        docs_processed = []
        for doc in knowledge_base:
            docs_processed += text_splitter.split_documents([doc])

        # Remove duplicates
        unique_texts = {}
        docs_processed_unique = []
        for doc in docs_processed:
            if doc.page_content not in unique_texts:
                unique_texts[doc.page_content] = True
                docs_processed_unique.append(doc)

        return docs_processed_unique

    docs_processed = split_documents(
        512,
        RAW_KNOWLEDGE_BASE,
        tokenizer_name=EMBEDDING_MODEL_NAME,
    )

    # Let's visualize the chunk sizes we would have in tokens from a common model

    tokenizer = AutoTokenizer.from_pretrained(EMBEDDING_MODEL_NAME)
    lengths = [len(tokenizer.encode(doc.page_content)) for doc in
tqdm(docs_processed)]
    fig = pd.Series(lengths).hist()
```

```
    plt.title("Distribution of document lengths in the knowledge base (in count of tokens)")
    plt.show()

    embedding_model = HuggingFaceEmbeddings(
        model_name=EMBEDDING_MODEL_NAME,
        multi_process=True,
        model_kwargs={"device": "cuda"},
        encode_kwargs={"normalize_embeddings": True},  # Set `True` for cosine similarity
    )

    KNOWLEDGE_VECTOR_DATABASE = FAISS.from_documents(
        docs_processed, embedding_model, distance_strategy=DistanceStrategy.COSINE
    )
    user_query = "How to create a pipeline object?"
    query_vector = embedding_model.embed_query(user_query)

    embedding_projector = pacmap.PaCMAP(n_components=2, n_neighbors=None, MN_ratio=0.5, FP_ratio=2.0, random_state=1)

    embeddings_2d = [
        list(KNOWLEDGE_VECTOR_DATABASE.index.reconstruct_n(idx, 1)[0]) for idx in range(len(docs_processed))
    ] + [query_vector]

    documents_projected = embedding_projector.fit_transform(np.array(embeddings_2d), init="pca")
```

```
df = pd.DataFrame.from_dict(
    [
        {
            "x": documents_projected[i, 0],
            "y": documents_projected[i, 1],
            "source": docs_processed[i].metadata["source"].split("/")[1],
            "extract": docs_processed[i].page_content[:100] + "...",
            "symbol": "circle",
            "size_col": 4,
        }
        for i in range(len(docs_processed))
    ]
    + [
        {
            "x": documents_projected[-1, 0],
            "y": documents_projected[-1, 1],
            "source": "User query",
            "extract": user_query,
            "size_col": 100,
            "symbol": "star",
        }
    ]
)

# Visualize the embedding
fig = px.scatter(
    df,
    x="x",
    y="y",
    color="source",
```

```
            hover_data="extract",
            size="size_col",
            symbol="symbol",
            color_discrete_map={"User query": "black"},
            width=1000,
            height=700,
        )
        fig.update_traces(
            marker=dict(opacity=1, line=dict(width=0, color="DarkSlateGrey")),
            selector=dict(mode="markers"),
        )
        fig.update_layout(
            legend_title_text="<b>Chunk source</b>",
            title="<b>2D Projection of Chunk Embeddings via PaCMAP</b>",
        )
        fig.show()

        print(f"\nStarting retrieval for {user_query=}...")
        retrieved_docs =
KNOWLEDGE_VECTOR_DATABASE.similarity_search(query=user_query, k=5)
        print("\n==================================Top
document==================================")
        print(retrieved_docs[0].page_content)
        print("==================================Metadata==========
=====================")
        print(retrieved_docs[0].metadata)

        READER_MODEL_NAME = "HuggingFaceH4/zephyr-7b-beta"

        bnb_config = BitsAndBytesConfig(
```

```
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
)
model = AutoModelForCausalLM.from_pretrained(READER_MODEL_NAME, quantization_config=bnb_config)
tokenizer = AutoTokenizer.from_pretrained(READER_MODEL_NAME)

READER_LLM = pipeline(
    model=model,
    tokenizer=tokenizer,
    task="text-generation",
    do_sample=True,
    temperature=0.2,
    repetition_penalty=1.1,
    return_full_text=False,
    max_new_tokens=500,
)
READER_LLM("What is 4+4? Answer:")
prompt_in_chat_format = [
    {
        "role": "system",
        "content": """Using the information contained in the context,
give a comprehensive answer to the question.
Respond only to the question asked, response should be concise and relevant to the question.
Provide the number of the source document when relevant.
If the answer cannot be deduced from the context, do not give an answer.""",
    },
```

```
    {
        "role": "user",
        "content": """Context:
{context}
---
Now here is the question you need to answer.

Question: {question}""",
    },
]
RAG_PROMPT_TEMPLATE = tokenizer.apply_chat_template(
    prompt_in_chat_format, tokenize=False, add_generation_prompt=True
)
print(RAG_PROMPT_TEMPLATE)
retrieved_docs_text = [doc.page_content for doc in retrieved_docs]  # We only need the text of the documents
context = "\nExtracted documents:\n"
context += "".join([f"Document {str(i)}:::\n" + doc for i, doc in enumerate(retrieved_docs_text)])

final_prompt = RAG_PROMPT_TEMPLATE.format(question="How to create a pipeline object?", context=context)

# Redact an answer
answer = READER_LLM(final_prompt)[0]["generated_text"]
print(answer)

def answer_with_rag(
    question: str,
    llm: Pipeline,
```

```
        knowledge_index: FAISS,
        reranker: Optional[RAGPretrainedModel] = None,
        num_retrieved_docs: int = 30,
        num_docs_final: int = 5,
    ) -> Tuple[str, List[LangchainDocument]]:
        # Gather documents with retriever
        print("=> Retrieving documents...")
        relevant_docs = knowledge_index.similarity_search(query=question, k=num_retrieved_docs)
        relevant_docs = [doc.page_content for doc in relevant_docs]  # Keep only the text

        # Optionally rerank results
        if reranker:
            print("=> Reranking documents...")
            relevant_docs = reranker.rerank(question, relevant_docs, k=num_docs_final)
            relevant_docs = [doc["content"] for doc in relevant_docs]

        relevant_docs = relevant_docs[:num_docs_final]

        # Build the final prompt
        context = "\nExtracted documents:\n"
        context += "".join([f"Document {str(i)}:::\n" + doc for i, doc in enumerate(relevant_docs)])

        final_prompt = RAG_PROMPT_TEMPLATE.format(question=question, context=context)

        # Redact an answer
        print("=> Generating answer...")
```

```
        answer = llm(final_prompt)[0]["generated_text"]

        return answer, relevant_docs

    RERANKER = RAGPretrainedModel.from_pretrained("colbert-ir/colbertv2.0")

    print("=====================================Answer=============================")
    print(f"{answer}")
    print("=====================================Source docs=============================")
    for i, doc in enumerate:
        print(f"Document {i}----------------------------------------------------------")
        print(doc)
```

**ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ**

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

# АВТОМАТИЗОВАНА СИСТЕМА УПРАВЛІННЯ ВИРОБНИЧИМИ ПРОЦЕСАМИ НА ОСНОВІ МОДЕЛІ RETRIEVAL-AUGMENTED GENERATION (RAG)

Виконав: здобувач вищої освіти групи: ШІД-41 Арсеній КОРОВКІН

Керівник: *доц каф к.т.н* Антон ШАНТИР

Київ 2025

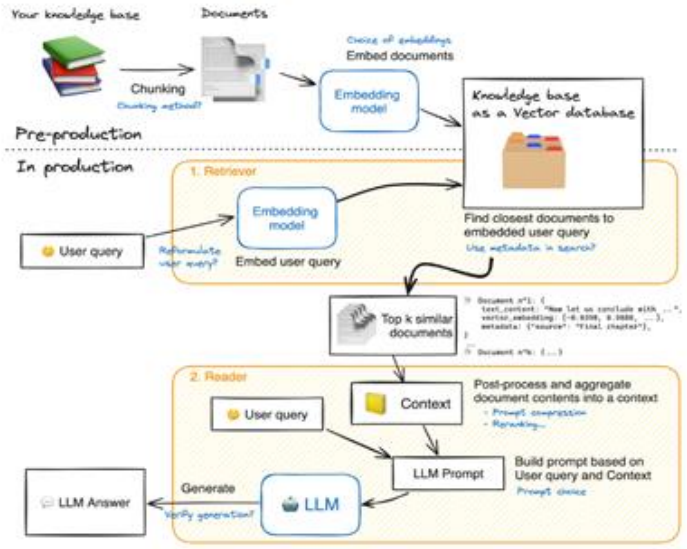
## Актуальність теми

Сучасне виробництво вимагає впровадження інтелектуальних систем управління для підвищення ефективності, зниження витрат та автоматизації аналізу технічної документації. Використання моделі RAG дозволяє поєднати можливості пошуку та генерації знань, що є актуальним у контексті цифрової трансформації.

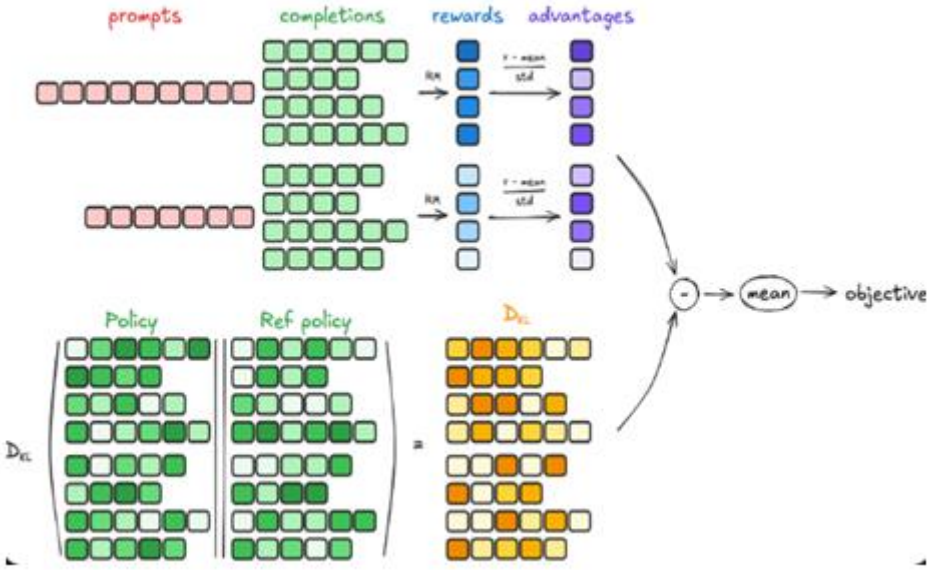
## Мета та завдання

Розробити автоматизовану систему керування виробничими процесами включаючи технологію штучного інтелекту, для розпізнавання конструкторських розрахунків та бухгалтерських наряд-рахунків с записом у файл для бази даних.

## Як працює RAG




## Система навчання




## Висновки

У процесі впровадження Retrieval-Augmented Generation (RAG) моделі та суміжних бібліотек було здобуто такі результати:

**Розширення функціональності АСУВ**

**Автоматизація аналітики**

**Інтеграція генеративного ШІ з базою знань**

**Прискорення доступу до інформації**