

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Технології штучного інтелекту

*Методичні вказівки до виконання лабораторних робіт
для студентів денної форми навчання за спеціальностями F7 “Комп’ютерна
інженерія”, F5 “Кібербезпека та захист інформації”, F3 “Комп’ютерні
науки”*

ЗАТВЕРДЖЕНО

на засіданні кафедри кібербезпеки та
програмного забезпечення, протокол №1
від 26.08.2025.

Кропивницький

2025

УДК 004.8

Технології штучного інтелекту: методичні вказівки до виконання лабораторних робіт для студентів за спеціальностями F7 «Комп'ютерна інженерія» ,F3 «Комп'ютерні науки» , F5«Кібербезпека та захист інформації» / М-во освіти і науки України, Центральноукр. нац. техн. ун-т; [уклад. : Дреєв О.М, Ткачук Р.О., Козірова Н.Л., Усік П. С.] – Кропивницький: ЦНТУ, 2025. – 60с.

Укладачі: Дреєв О.М. доцент кафедри кібербезпеки та програмного забезпечення, кандидат технічних наук;

Ткачук Р.О. асистент кафедри кібербезпеки та програмного забезпечення, Senior DevOps Engineer at Eram

Козірова Н.Л., викладач кафедри кібербезпеки та програмного забезпечення, програміст бази даних в I&U Group;

Усік П. С., доктор філософії, старший викладач кафедри кібербезпеки та програмного забезпечення;

Рецензенти: Смірнов О. А., докт. техн. наук, професор;

Коваленко О. В., доктор. техн. наук, професор.

© Центральноукраїнський
національний технічний
університет, 2025

ЗМІСТ

Вступ.....	4
Лабораторна робота №1 Основи промптингу та робота з ШІ через веб-інтерфейс або простий API.....	6
Лабораторна робота №2 Програмні інструменти для роботи з моделями ШІ. Поняття API та середовища тестування мовних моделей.....	12
Лабораторна робота №3 Використання бібліотеки-інтерфейсу моделей Transformers від HuggingFace для виконання прикладних задач.....	19
Лабораторна робота №4 Пошук об'єктів на зображенні за допомогою засобів штучного інтелекту.	27
Лабораторна робота №5 Робота з текстовими файлами як джерелом даних для покращення відповідей моделі (RAG).....	33
Лабораторна робота №6 Створення інтерфейсу у вигляді чат-бота для бібліотеки Transformers від HuggingFace.....	39
Лабораторна робота №7 Аналіз метричних даних за допомогою ШІ на основі JSON. Створення моделей.	45
Лабораторна робота №8 Генерація мультимодального контенту на основі готових моделей ШІ.....	53
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	58

Вступ

Навчальна дисципліна «Технології штучного інтелекту» орієнтована на формування у здобувачів вищої освіти цілісного уявлення про сучасні методи, моделі та програмні засоби штучного інтелекту, а також на набуття практичних навичок їх застосування у прикладних програмних системах.

Метою викладання дисципліни «Технології штучного інтелекту» є формування у студентів теоретичних знань та практичних навичок застосування методів і технологій штучного інтелекту для розв'язання прикладних задач, проєктування та створення інтелектуальних програмних систем, а також ефективного використання програмних засобів і моделей штучного інтелекту з урахуванням обмежень обчислювальних ресурсів.

Завдання дисципліни

Основними завданнями вивчення дисципліни є:

- вивчення базових понять, методів і підходів штучного інтелекту;
- ознайомлення з сучасними технологіями та моделями штучного інтелекту;
- формування навичок проєктування та розроблення програмних систем із використанням методів штучного інтелекту;
- набуття практичних навичок застосування програмних засобів і бібліотек штучного інтелекту;
- розвиток уміння обирати адекватні моделі та алгоритми штучного інтелекту для розв'язання прикладних задач з урахуванням ресурсних обмежень.

У результаті вивчення навчальної дисципліни студент повинен знати:

- принципи побудови інтелектуальних програмних систем;
- основні напрями та методи штучного інтелекту;
- базові алгоритми та моделі штучного інтелекту;
- сучасні програмні засоби та інструменти штучного інтелекту.

- застосовувати технології штучного інтелекту для розв’язання прикладних задач;
- проєктувати та реалізовувати програмне забезпечення з використанням різних парадигм програмування;
- створювати та використовувати моделі штучного інтелекту в практичних застосуваннях;
- працювати з багаторівневими та розподіленими програмними системами, включаючи бази даних і бази знань.

Результати навчання

Після завершення вивчення дисципліни студент здатний:

- розробляти програмні моделі предметних середовищ та обирати доцільну парадигму програмування для реалізації методів штучного інтелекту;
- застосовувати програмні засоби штучного інтелекту для вирішення професійних, освітніх та прикладних задач;
- створювати та використовувати моделі штучного інтелекту з урахуванням обмежень обчислювальних ресурсів.

Лабораторна робота №1 Основи промптингу та робота з ШІ через веб-інтерфейс або простий API

Мета

Сформувати базові навички промптингу та ефективної роботи з ChatGPT через веб-інтерфейс або API для отримання якісних відповідей.

Теоретичні відомості

Промпти - це текстові інструкції, що складаються з ключових фраз або речень і використовуються для взаємодії з мовною моделлю ChatGPT. Система штучного інтелекту застосовує промпт як основу для формування відповіді, аналізуючи зміст і структуру поданого запиту. Саме тому коректність, чіткість і логічність формулювання запиту мають вирішальне значення для отримання релевантного результату.

На сьогодні існують готові шаблони промтів, які користувачі можуть адаптувати до власних потреб, змінюючи їх шляхом доповнення або скорочення. Сфера застосування таких шаблонів є досить широкою: вони використовуються у копірайтингу, створенні художніх текстів, розробці вебсайтів, а також у процесі навчання програмуванню та іншим технічним дисциплінам.

Сучасні версії мовних моделей демонструють високий рівень ефективності. Зокрема, ChatGPT-4 за умови правильно сформульованого запиту здатний створювати тематичні есе, аналізувати зображення та інтерпретувати гумористичні висловлювання. Завдяки цим можливостям штучний інтелект може виступати універсальним інструментом для розв'язання широкого кола завдань. Для результативної роботи з ChatGPT користувачеві необхідно опанувати принципи формування команд і зрозуміти загальну логіку функціонування нейромережі.

Під час формування промту доцільно чітко визначити роль чат-бота, обираючи відповідну формулювання та безпосередньо задаючи цю роль у тексті запиту. Наприклад, можна зазначити, що штучний інтелект має виступати в ролі

фахівця певної галузі, зокрема письменника, який спеціалізується на детективному жанрі, або юриста, що надає правові консультації.

Рекомендується використовувати дієслівні формулювання, що безпосередньо вказують на очікувану дію, уникаючи умовних або невизначених конструкцій. Доцільно застосовувати такі слова, як «створи», «зроби», «сформулюй», «опиши», «запропонуй», що сприяє більш точному виконанню поставленого завдання.

Не варто формулювати запити в абстрактній формі, оскільки у такому разі відповідь моделі також матиме узагальнений характер. Натомість слід прагнути до максимальної конкретизації промпту, що дозволяє підвищити точність і релевантність результату.

За потреби користувач може одразу задати обмеження щодо обсягу відповіді, наприклад визначивши максимальну кількість символів або слів. Це є доцільним у випадках, коли необхідно отримати стислий текст і оптимізувати час взаємодії з моделлю.

Крім того, доцільно зазначати бажаний стиль викладу матеріалу, зокрема офіційно-науковий, розмовний, творчий або публіцистичний. Чим більше чітких і зрозумілих уточнень містить промпт, тим вищою буде якість згенерованої відповіді.

ChatGPT підтримує різні формати подання інформації, тому користувач може заздалегідь визначити форму результату, наприклад у вигляді таблиці, нумерованого переліку, структурованих фрагментів або цитат.

Ключовим елементом ефективного промптингу є чітке формулювання завдання та опис способу, у який штучний інтелект, виступаючи в ролі автора або виконавця, має його реалізувати. Чим детальнішим є пояснення, тим легше моделі інтерпретувати запит і обрати оптимальний підхід до його виконання.

Тип завдання	Структура промпту	Приклад
Переклад	«Переклади	«Переклади наступний текст з англійської

тексту	наступний текст з мови 1 на мову 2: текст»	на українську: “Artificial intelligence is transforming the world.”»
Створення контенту	«Напиши тип контенту на тему <i>тема</i> , враховуючи <i>особливості</i> »	«Напиши статтю на тему “Переваги здорового харчування”, враховуючи аудиторію підлітків.»
Резюме та висновки	«Узагальни наступний текст: текст»	«Узагальни наступний текст: “В останнє десятиліття технології швидко розвиваються, впливаючи на всі аспекти життя...”»
Генерація ідей	«Запропонуй <i>кількість</i> ідей для <i>мети</i> у сфері <i>сфера</i> »	«Запропонуй 5 ідей для маркетингової кампанії у сфері екологічно чистих продуктів.»
Навчання та пояснення	«Поясни поняття <i>термін</i> простими словами»	«Поясни поняття “квантова заплутаність” простими словами.»
Редагування тексту	«Відредагуй наступний текст для <i>мети</i> та виправ помилки: текст»	«Відредагуй наступний текст для публікації в науковому журналі та виправ помилки: “Дослідження показало, що дані були неточними через помилки вимірювання...”»
Складання плану	«Склади план для <i>завдання</i> , включаючи <i>особливості</i> »	«Склади план вивчення іспанської мови протягом 3 місяців, включаючи щоденні заняття по 1 годині.»
Роль або персона	«Уяви, що ти <i>роль</i> , і допоможи мені з <i>завданням</i> »	«Уяви, що ти досвідчений HR-менеджер, і допоможи мені підготуватися до співбесіди на посаду аналітика даних.»
Аналіз даних	«Проаналізуй наступні дані та надай висновки:»	«Проаналізуй наступні дані щодо продажів за квартал і надай висновки: [дані].»

	дані»	
Творчі завдання	«Придумай щось у жанрі <i>жанр</i> , включаючи <i>елементи</i> »	«Придумай коротке оповідання у жанрі наукової фантастики, що включає подорожі в часі та парадокси.»

Приховані можливості ChatGPT поступово стають очевидними та стрімко набувають популярності. Однією з таких можливостей є здатність мовної моделі виступати в ролі інтелектуального помічника в різних професійних і прикладних сферах. Для використання цієї функції достатньо чітко визначити роль, яку має виконувати штучний інтелект, і сформулювати відповідний запит. Задані ролі можуть бути як традиційними (наприклад, менеджер з туризму або юрист), так і нетиповими — ілюзіоніст або стоматолог.

Шеф-кухар. Користувач формулює запит із метою отримання вегетаріанського меню, що складається з різноманітних страв. Початковий запит може мати вигляд: «Запропонуй ситну вечерю, розраховану на дві порції».

Візажист. Штучному інтелекту надається роль фахівця з макіяжу, який працює з різними косметичними продуктами та текстурами і спеціалізується на денному та вечірньому макіяжі. Приклад запиту: «Розроби образ для 17-річної дівчини на випускний вечір з урахуванням сухого типу шкіри».

Стиліст. Користувач визначає бюджет, а модель пропонує декілька варіантів образів для конкретного заходу. Наприклад: «Бюджет становить 7000 гривень. Запропонуй 10 ідей образів для вечірки в стилі 1990-х років».

Шахіст. ChatGPT може виконувати роль суперника в грі в шахи, обмінюючись ходами з користувачем у визначеному порядку. Наприклад: «Перша партія — я граю білими. Перший хід — d3. Пояснення ходів не надавай, після кожного ходу оновлюй стан дошки».

Туристичний гід. Модель тимчасово виконує функції гіда, пропонуючи місця для відвідування на основі поточного місцеперебування користувача та

його вподобань. Приклад запиту: «Я перебуваю в Києві й не хочу відвідувати музеї».

Автор есе. ChatGPT може виконувати роль автора аналітичних або публіцистичних есе: досліджувати тему, формулювати тезу та створювати логічно структурований текст. Приклад: «Напиши есе про важливість сортування відходів для збереження довкілля».

UX/UI-розробник. Модель отримує опис застосунку або вебсайту та пропонує креативні способи покращення користувацького досвіду, зокрема шляхом створення прототипів, аналізу варіантів дизайну та оцінювання їх ефективності. Початковий запит: «Розроби дизайн мобільного застосунку для туристичної компанії».

Ріелтор. Штучний інтелект виконує роль спеціаліста з нерухомості, допомагаючи підібрати житло відповідно до бюджету, стилю життя та вимог до місця розташування, з урахуванням особливостей локального ринку. Приклад запити: «Знайди двоповерховий будинок для сім'ї з п'яти осіб поблизу центру Праги».

Завдання лабораторної роботи

1. Ознайомитися з веб-інтерфейсом ChatGPT (або подібним інструментом) та його основними можливостями.
2. Ознайомитися з принципами промптингу:
 1. створення ефективних запитів (prompt) до моделі;
 2. використання уточнень, обмежень та форматів відповідей.
3. Скласти 3–5 різних промптів для вирішення конкретних завдань (наприклад, створення списку, генерація тексту, перефразування).
4. Відправити промпти через веб-інтерфейс або простий API і зафіксувати отримані результати.
5. Проаналізувати відповіді моделі:
 1. оцінити точність та релевантність;

2. визначити, як зміна формулювання промπτу впливає на результат.
6. Скласти короткий звіт із прикладами промπτів та відповідей, висновками про ефективність промπτингу.

Лабораторна робота №2 Програмні інструменти для роботи з моделями ШІ. Поняття API та середовища тестування мовних моделей

Мета

Ознайомитися з інструментами Ollama та LM Studio, отримати досвід роботи з промтингом та шаблонізацією.

Теоретичні відомості

Для запуску LLM необхідно почати з встановлення застосунку Ollama або LM Studio.

Ollama — це інструмент з орієнтацією на командний рядок і API, який спрощує завантаження, запуск і керування LLM. Це самий простий шлях отримати інтерфейс для взаємодії з моделями через командний рядок (TUI-інтерфейс).

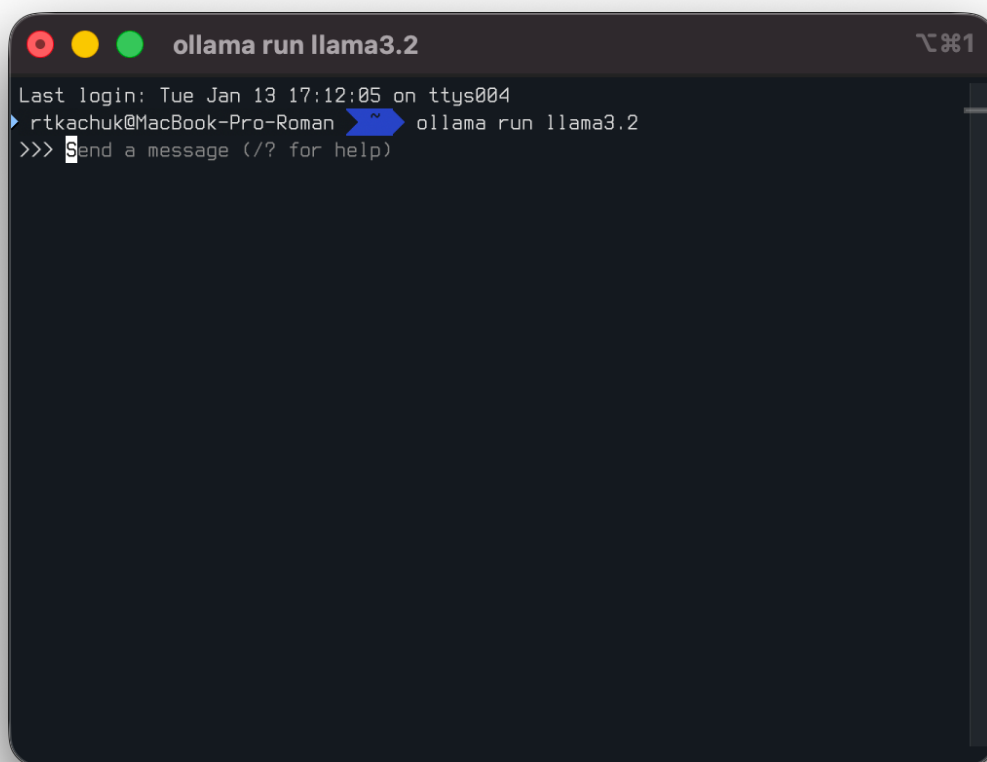
Для встановлення Ollama необхідно переконатися, що операційна система підтримує локальний запуск великих мовних моделей. Підтримуються операційні системи Linux, macOS та Windows, причому на всіх платформах установка максимально спрощена і не потребує ручного налаштування Python-оточення чи драйверів CUDA на базовому рівні.

На macOS інсталяція виконується шляхом завантаження офіційного інсталятора з сайту проекту. Після відкриття завантаженого .dmg-файлу програма переноситься до стандартної директорії Applications. Перший запуск може супроводжуватися попередженням системи безпеки, яке потрібно підтвердити вручну. Після цього Ollama автоматично запускає локальний сервіс, доступний з терміналу.

У середовищі Linux встановлення відбувається через виконання одного shell-скрипта, який завантажує бінарний файл, встановлює його у систему та реєструє Ollama як сервіс. Після завершення інсталяції демон запускається автоматично, а керування здійснюється через команду ollama у терміналі. У більшості дистрибутивів додаткові дії з налаштування прав доступу не потрібні.

На Windows Ollama встановлюється за допомогою стандартного .exe-інсталятора. Під час встановлення створюється локальний сервіс, який працює у фоновому режимі. Після завершення процесу команда ollama стає доступною у PowerShell або Command Prompt, без необхідності додатково змінювати змінні середовища.

Після встановлення перевірка працездатності виконується через запуск будь-якої тестової моделі, наприклад командою запуску базової LLM. У момент першого запуску модель автоматично завантажується з репозиторію Ollama та кешується локально, що може зайняти певний час залежно від швидкості інтернет-з'єднання та обсягу моделі. Подальші запуски відбуваються миттєво, оскільки всі ваги вже збережені на диску.

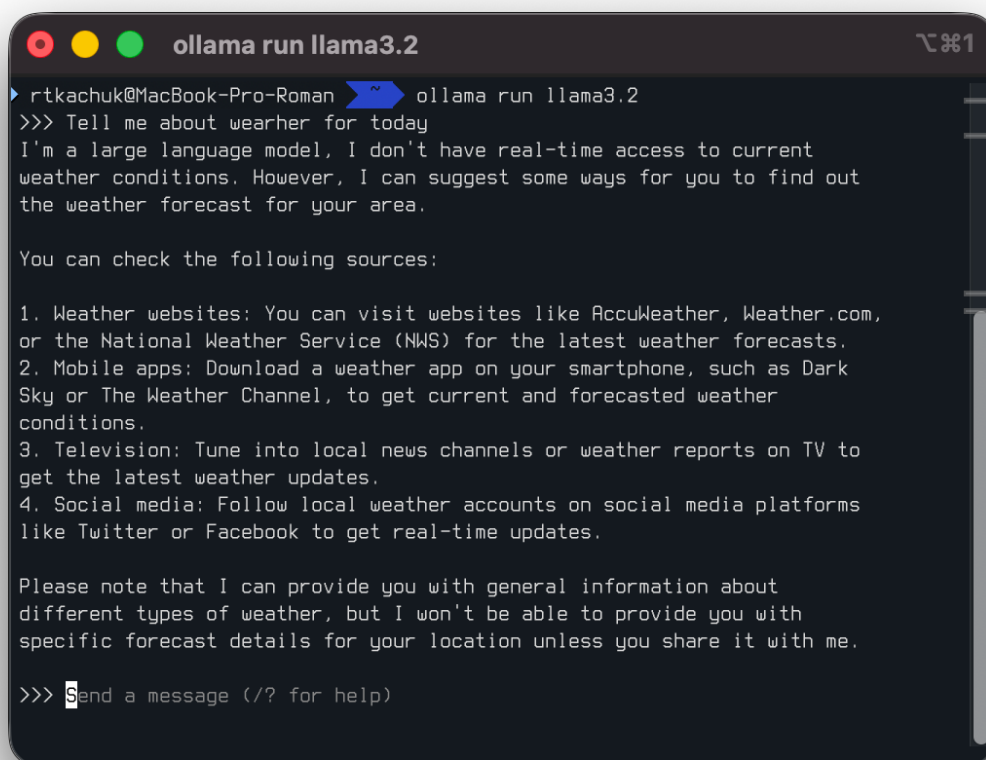
A screenshot of a macOS terminal window titled "ollama run llama3.2". The terminal shows the following text: "Last login: Tue Jan 13 17:12:05 on ttys004", followed by a prompt "rtkachuk@MacBook-Pro-Roman" and the command "ollama run llama3.2" which has been executed. Below the command, the prompt "Send a message (/? for help)" is visible. The terminal window has standard macOS window controls (red, yellow, green buttons) and a keyboard shortcut icon (Command+Q) in the top right corner.

```
ollama run llama3.2
Last login: Tue Jan 13 17:12:05 on ttys004
rtkachuk@MacBook-Pro-Roman ~ % ollama run llama3.2
>>> Send a message (/? for help)
```

Рисунок 1

Таким чином, Ollama встановлюється як повноцінний локальний сервіс для роботи з LLM, не вимагаючи від користувача глибоких знань у сфері машинного навчання або ручного керування залежностями, що робить його зручним інструментом для навчання, експериментів і розробки AI-застосунків.

Після цього користувач має можливість запускати мовні моделі через аргумент `run`. Для цього необхідно запуснути команду `ollama run llama3.2`. Аргумент `run` використовується для вибору моделі, з якою ми будемо працювати та автоматично її завантажить у випадку її відсутності на комп'ютері. Всі моделі зберігаються в теці `.ollama/models` домашнього каталогу вашого користувача. За замовчуванням, завантажується звичайна модель на 8 мільярдів параметрів, яка розуміє лише англійську мову.



```
ollama run llama3.2
> rtkachuk@MacBook-Pro-Roman ~ % ollama run llama3.2
>>> Tell me about wearher for today
I'm a large language model, I don't have real-time access to current
weather conditions. However, I can suggest some ways for you to find out
the weather forecast for your area.

You can check the following sources:

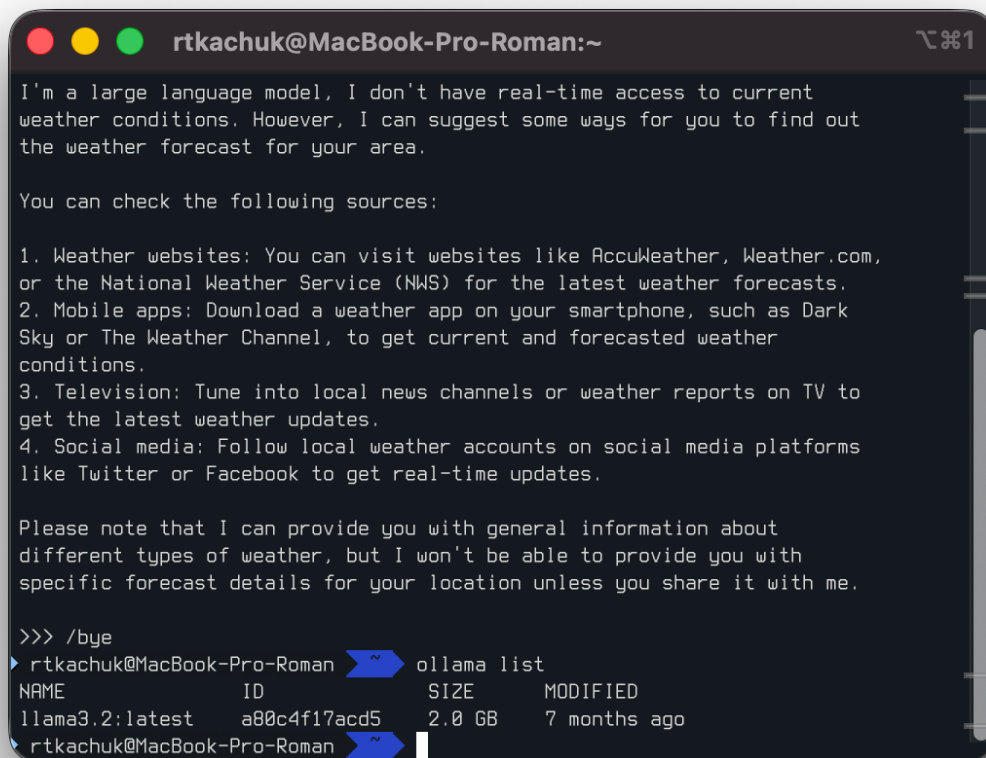
1. Weather websites: You can visit websites like AccuWeather, Weather.com,
or the National Weather Service (NWS) for the latest weather forecasts.
2. Mobile apps: Download a weather app on your smartphone, such as Dark
Sky or The Weather Channel, to get current and forecasted weather
conditions.
3. Television: Tune into local news channels or weather reports on TV to
get the latest weather updates.
4. Social media: Follow local weather accounts on social media platforms
like Twitter or Facebook to get real-time updates.

Please note that I can provide you with general information about
different types of weather, but I won't be able to provide you with
specific forecast details for your location unless you share it with me.

>>> Send a message (/? for help)
```

Рисунок 2

Завершити роботу з моделлю можна за допомогою команди `/bye`, а продивитись список всіх завантажених моделей можна побачити за допомогою аргументу `list` до команди `ollama`:



```
rtkachuk@MacBook-Pro-Roman:~  
I'm a large language model, I don't have real-time access to current weather conditions. However, I can suggest some ways for you to find out the weather forecast for your area.  
  
You can check the following sources:  
  
1. Weather websites: You can visit websites like AccuWeather, Weather.com, or the National Weather Service (NWS) for the latest weather forecasts.  
2. Mobile apps: Download a weather app on your smartphone, such as Dark Sky or The Weather Channel, to get current and forecasted weather conditions.  
3. Television: Tune into local news channels or weather reports on TV to get the latest weather updates.  
4. Social media: Follow local weather accounts on social media platforms like Twitter or Facebook to get real-time updates.  
  
Please note that I can provide you with general information about different types of weather, but I won't be able to provide you with specific forecast details for your location unless you share it with me.  
  
>>> /bye  
▶ rtkachuk@MacBook-Pro-Roman ~ ollama list  
NAME          ID          SIZE      MODIFIED  
llama3.2:latest  a80c4f17acd5  2.0 GB   7 months ago  
▶ rtkachuk@MacBook-Pro-Roman ~
```

Рисунок 3

Таким чином ми отримали повноцінний інтерфейс для початкової взаємодії зі ШІ, а саме моделями `ollama`.

Для більш графічного представлення та роботи у графічному середовищі, необхідно використовувати такий інструмент як **LM Studio**.

LM Studio — це настільний застосунок з графічним інтерфейсом, орієнтований на зручну взаємодію з моделями та їх параметрами.

На macOS встановлення виконується шляхом завантаження `.dmg`-файлу з офіційного сайту `LM Studio`. Після відкриття образу програму необхідно перемістити до каталогу `Applications`. Під час першого запуску система безпеки

може запросити підтвердження довіри до застосунку, оскільки він завантажений з інтернету. Після підтвердження LM Studio запускається як звичайна настільна програма з повноцінним графічним інтерфейсом.

У Windows процес встановлення здійснюється за допомогою стандартного інсталятора .exe. Після завершення інсталяції програма стає доступною через меню «Пуск» або ярлик на робочому столі. Додаткове налаштування змінних середовища або системних сервісів не потрібне, оскільки LM Studio працює як самостійний користувацький застосунок.

На Linux LM Studio зазвичай постачається у вигляді AppImage або архіву з виконуваним файлом. Після завантаження достатньо надати файлу права на виконання та запустити його без інсталяції у систему. Програма працює в межах користувацького простору і не вимагає прав адміністратора.

Після першого запуску LM Studio пропонує завантажити мовну модель з вбудованого каталогу. Обрана модель автоматично зберігається локально та надалі використовується без підключення до інтернету. Усі обчислення виконуються на локальному процесорі або відеокарті, якщо апаратне прискорення підтримується системою. Після завершення завантаження модель одразу готова до використання через чат-інтерфейс або локальний API-сервер.

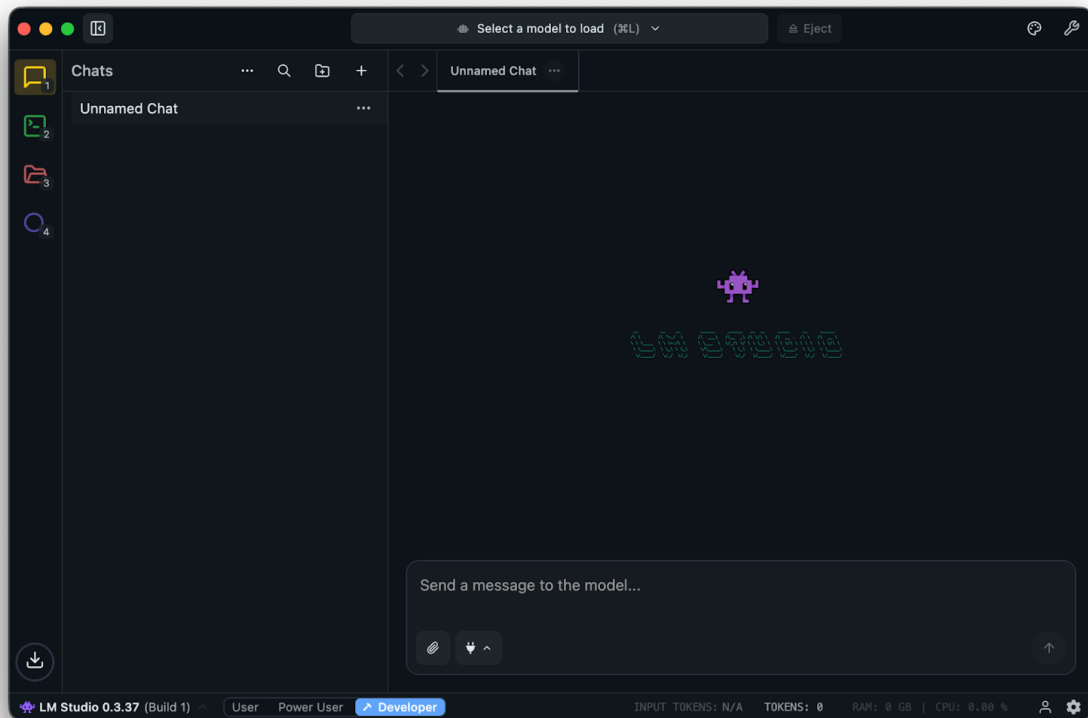


Рисунок 4

Завдання лабораторної роботи

1. Ознайомитися з інтерфейсом та основними можливостями **Ollama** та **LM Studio**.
2. Встановити та налаштувати обране середовище для локальної роботи з великими мовними моделями (LLM).
3. Ознайомитися з поняттям **API** для взаємодії з мовними моделями.
4. Скласти та протестувати кілька промптів для генерації тексту, відповідей на запитання або іншого завдання.
5. Використати шаблони промптів для автоматизації створення повторюваних запитів.
6. Проаналізувати результати роботи моделей: які промпти працюють ефективніше, як змінюється якість відповіді залежно від формулювання запити.

7. Скласти короткий звіт із прикладами промптів, шаблонів та отриманих результатів.

Лабораторна робота №3 Використання бібліотеки-інтерфейсу моделей Transformers від HuggingFace для виконання прикладних задач

Мета

Ознайомитися з бібліотекою Hugging Face Transformers, навчитися використовувати готові мовні моделі для виконання практичних завдань; набути навичок роботи з моделями ШІ.

Теоретичні відомості

Бібліотека **Transformers** виконує функцію середовища визначення моделей для сучасних передових моделей машинного навчання, що застосовуються у сфері обробки тексту, комп'ютерного зору, аудіо- та відеоданих, а також мультимодальних моделей, як у режимі інференсу, так і під час навчання.

Вона централізує визначення архітектур моделей, забезпечуючи їх узгодженість у межах усього екосистемного середовища. **Transformers** виступає ключовою ланкою між різними фреймворками: якщо визначення моделі підтримується бібліотекою, воно, як правило, є сумісним із більшістю систем навчання (зокрема Axolotl, Unsloth, DeepSpeed, FSDP, PyTorch Lightning тощо), рушіїв інференсу (vLLM, SGLang, TGI та інші), а також суміжних бібліотек моделювання (наприклад, llama.cpp, mlx), які використовують визначення моделей із **Transformers**.

Розробники бібліотеки декларують зобов'язання підтримувати нові передові моделі та сприяти демократизації їх використання шляхом забезпечення простоти, гнучкості налаштування та ефективності визначення моделей.

На платформі **Hugging Face Hub** доступно понад один мільйон контрольних точок (checkpoint) моделей **Transformers**, які можуть бути використані для досліджень і практичних застосувань.

Користувачам пропонується ознайомитися з можливостями Hub для вибору відповідної моделі та швидкого початку роботи з використанням бібліотеки **Transformers**. Крім того, розділ **Models Timeline** надає змогу відстежувати новітні архітектури текстових, візуальних, аудіо- та мультимодальних моделей, реалізованих у **Transformers**.

Функціональні можливості

Бібліотека **Transformers** надає повний набір інструментів, необхідних для інференсу та навчання попередньо навчених моделей найвищого рівня. До основних можливостей належать:

Pipeline — спрощений та оптимізований інтерфейс для виконання інференсу у широкому спектрі завдань машинного навчання, зокрема генерації тексту, сегментації зображень, автоматичного розпізнавання мовлення, відповіді на запитання в документах та інших;

Trainer — комплексний модуль для навчання моделей PyTorch, що підтримує такі можливості, як змішана точність обчислень, torch.compile, технологію FlashAttention, а також розподілене навчання;

generate — високопродуктивний механізм генерації тексту за допомогою великих мовних моделей (LLM) і візуально-мовних моделей (VLM), який включає підтримку потокової генерації та різних стратегій декодування.

Такий підхід робить **Transformers** універсальним і стандартизованим інструментом для розробки, навчання та застосування сучасних моделей штучного інтелекту.

Бібліотека **Transformers** розроблена для програмістів, інженерів машинного навчання та наукових дослідників. Її архітектура ґрунтується на низці ключових принципів проектування:

Висока швидкодія та простота використання. Кожна модель реалізується на основі лише трьох основних класів — конфігурації, моделі та препроцесора, що забезпечує швидке застосування як для інференсу, так і для навчання за допомогою інструментів **Pipeline** або **Trainer**.

Використання попередньо навчених моделей. Застосування попередньо навчених моделей дає змогу зменшити обчислювальні витрати, часові затрати та негативний вплив на довкілля порівняно з повним навчанням моделей з нуля. Кожна така модель відтворюється максимально наближено до оригінальної архітектури та параметрів і забезпечує сучасний рівень продуктивності (state of the art).

Зазначені принципи роблять **Transformers** ефективним інструментом для дослідницької та прикладної діяльності у сфері штучного інтелекту.

Для подальшої роботи необхідно створити віртуальне середовище, у межах якого буде встановлено бібліотеку **Transformers**:

```
python3 -m venv .env
source .env/bin/activate
```

Встановлення бібліотеки **Transformers** виконується за допомогою такої команди:

```
pip3 install transformers
```

Для використання апаратного прискорення на графічному процесорі (GPU) необхідно встановити відповідні драйвери CUDA для PyTorch.

Щоб перевірити, чи система коректно визначає графічний процесор NVIDIA, виконайте команду:

```
nvidia-smi
```

У разі потреби встановлення версії **Transformers**, що працює виключно на центральному процесорі (CPU), слід виконати такі команди:

```
pip3 install torch --index-url
https://download.pytorch.org/whl/cpu
pip3 install transformers
```

Для перевірки успішності встановлення можна виконати наступну команду. У разі коректної інсталяції вона поверне мітку та числову оцінку для заданого тексту:

```
python -c "from transformers import pipeline;
print(pipeline('sentiment-analysis')('hugging face is the
best'))"
```

Очікуваний результат виконання команди має вигляд:

```
[{'label': 'POSITIVE', 'score': 0.9998704791069031}]
```

Це підтверджує правильність встановлення бібліотеки та її готовність до використання.

Найбазовішим об'єктом у бібліотеці **Transformers** є функція `pipeline()`. Вона поєднує модель із необхідними етапами попередньої та подальшої обробки даних, що дає змогу безпосередньо подавати текстові дані на вхід і отримувати зрозумілу та інтерпретовану відповідь. До прикладу її використання, наступний код проводить оцінку тексту.

```
from transformers import pipeline
classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace
course my whole life.")
```

Після запуску через інтерпритатор Python отримуємо відповідь:

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

За замовчуванням цей конвеєр обирає певну попередньо навчену модель, яка була додатково налаштована (*fine-tuned*) для виконання аналізу настрою англійських текстів. Під час створення об'єкта класифікатора модель автоматично завантажується та зберігається в кеші. У разі повторного виконання відповідної команди буде використано вже кешовану версію моделі, тому повторне завантаження не є необхідним.

Під час передавання тексту до конвеєра виконуються три основні етапи:

1. Текст проходить попередню обробку та перетворюється у формат, зрозумілий для моделі.

2. Попередньо оброблені вхідні дані передаються безпосередньо до моделі.
3. Результати прогнозування моделі піддаються подальшій обробці з метою їх інтерпретації та представлення у зручному для користувача вигляді.

Доступні конвеєри для різних модальностей

Функція `pipeline()` підтримує роботу з кількома модальностями, що дозволяє опрацьовувати текстові, візуальні, аудіодані, а також мультимодальні завдання. У межах цього курсу основну увагу буде зосереджено на текстових завданнях, однак для розуміння потенціалу трансформерної архітектури доцільно коротко розглянути всі доступні можливості.

Нижче наведено огляд основних типів конвеєрів.

Текстові конвеєри

- **text-generation** — генерація тексту на основі заданого запиту;
- **text-classification** — класифікація тексту за наперед визначеними категоріями;
- **summarization** — створення скороченого викладу тексту зі збереженням ключової інформації;
- **translation** — переклад тексту з однієї мови на іншу;
- **zero-shot-classification** — класифікація тексту без попереднього навчання на конкретних мітках;
- **feature-extraction** — отримання векторних представлень тексту.

Конвеєри для роботи з зображеннями

- **image-to-text** — генерація текстового опису зображення;
- **image-classification** — ідентифікація об'єктів або класів на зображенні;
- **object-detection** — визначення місця розташування та типів об'єктів на зображеннях.

Аудіоконвеєри

- **automatic-speech-recognition** — перетворення мовлення у текст;
- **audio-classification** — класифікація аудіосигналів за заданими категоріями;

- **text-to-speech** — перетворення тексту на усне мовлення.

Мультиmodalні конвеєри

- **image-text-to-text** — формування текстової відповіді на основі зображення та текстового запиту.

Зазначений набір конвеєрів демонструє універсальність бібліотеки **Transformers** та її здатність ефективно працювати з різними типами даних у межах єдиної архітектурної парадигми.

Для виконання завдань перекладу можна використовувати модель за замовчуванням, якщо у назві завдання зазначено мовну пару (наприклад, `translation_en_to_fr`). Однак найбільш простим і гнучким підходом є вибір конкретної моделі безпосередньо з **Model Hub**. У наведеному нижче прикладі демонструється переклад з французької мови на англійську:

```
from transformers import pipeline
translator = pipeline("translation",
model="Helsinki-NLP/opus-mt-fr-en")
translator("Ce cours est produit par Hugging Face.")
```

Результат виконання:

```
[{'translation_text': 'This course is produced by Hugging Face.'}]
```

Аналогічно до завдань генерації тексту та його узагальнення, для результату перекладу можна задати параметри максимальної (`max_length`) або мінімальної (`min_length`) довжини, що дозволяє контролювати обсяг згенерованого тексту.

Одним із потужних напрямів застосування трансформерних моделей є їхня здатність поєднувати та обробляти дані, отримані з кількох джерел. Це є особливо корисним у випадках, коли необхідно:

- здійснювати пошук інформації в кількох базах даних або репозиторіях;
- узагальнювати та інтегрувати дані з різних форматів, зокрема текстових, візуальних і аудіо;
- формувати цілісне уявлення про взаємопов'язану інформацію.

Наприклад, на основі трансформерних моделей можна побудувати систему, яка:

- виконує пошук інформації в базах даних з використанням кількох модальностей, таких як текст і зображення;
- об'єднує результати з різних джерел в єдину логічно узгоджену відповідь, зокрема поєднуючи дані з аудіофайлів і текстових описів;
- відображає найбільш релевантну інформацію з бази документів і супровідних метаданих.

Такий підхід демонструє потенціал трансформерних моделей як універсального інструмента для інтеграції, аналізу та представлення різномірних даних у складних інформаційних системах.

Завдання лабораторної роботи

1. Ознайомитися з бібліотекою **Hugging Face Transformers** та її основними компонентами (моделі, токенизатори, pipeline).
2. Встановити необхідні пакети та налаштувати середовище для роботи з моделями Transformers.
3. Завантажити готову модель через API Hugging Face для виконання практичних задач (генерація тексту, класифікація, переклад тощо).
4. Використати моделі для обробки прикладних завдань, складених у вигляді промптів або даних для аналізу.
5. Провести тестування роботи моделей: оцінити якість результатів та релевантність відповідей.

6. Скласти короткий звіт із прикладами роботи моделей та висновками щодо їх застосування у практичних задачах.

Лабораторна робота №4 Пошук об'єктів на зображенні за допомогою засобів штучного інтелекту.

Мета

Ознайомитися з підходом пошуку об'єктів на зображенні за допомогою штучного інтелекту.; отримати базові практичні навички використання моделей для визначення та пошуку об'єктів.

Теоретичні відомості

Мультимодальні моделі штучного інтелекту відкривають нові можливості для аналізу візуальної інформації засобами природної мови. Модель LLaVA поєднує візуальний енкодер і велику мовну модель, що дозволяє здійснювати семантичний аналіз зображення через текстовий запит. На відміну від класичних алгоритмів детекції об'єктів, вона не формує координатних рамок, а інтерпретує зміст сцени у вигляді опису. Використання платформи Ollama забезпечує локальний запуск моделі без звернення до хмарних сервісів, що є важливим у навчальних і дослідницьких середовищах з підвищеними вимогами до конфіденційності.

LLaVA функціонує на основі інтеграції візуального енкодера, який перетворює зображення у багатовимірне векторне представлення, та мовної моделі, що генерує текстову відповідь. Отримані з візуального модуля ембедінги проєктуються у простір мовної моделі, після чого здійснюється генерація тексту відповідно до сформульованого запиту користувача. Таким чином, пошук об'єктів реалізується як задача семантичного тлумачення сцени, а не як геометрична локалізація.

Після встановлення Ollama у локальній системі необхідно завантажити модель LLaVA за допомогою відповідної команди в терміналі. Платформа розгортає локальний сервер, який приймає HTTP-запити та забезпечує взаємодію з моделлю. Типовий порт за замовчуванням — 11434, що дозволяє інтегрувати систему з власними програмними застосунками.

Запуск моделі в інтерактивному режимі дає змогу передати шлях до зображення разом із текстовим запитом. У цьому випадку модель аналізує файл та формує текстовий опис або відповідає на поставлене запитання.

Процес пошуку об'єктів на зображенні передбачає формування чітко структурованого запиту. Якщо запит є надто узагальненим, відповідь може бути поверховою або неповною. Натомість уточнені формулювання, які задають тип об'єктів, їх функціональну належність або контекст, суттєво підвищують релевантність результату.

Наприклад, запит щодо переліку всіх фізичних предметів у межах сцени дає ширший опис, тоді як звернення до конкретної категорії, зокрема транспортних засобів або електронних пристроїв, дозволяє отримати більш структуровану відповідь. У разі необхідності подальшої машинної обробки доцільно вимагати від моделі структурований формат, наприклад JSON.

Ollama підтримує REST API, що забезпечує інтеграцію LLaVA у власні програмні системи. Зображення передається у вигляді Base64-кодованого рядка, а відповідь повертається у форматі JSON. Такий підхід дозволяє автоматизувати аналіз великої кількості зображень, формувати текстові описи, створювати метадані або здійснювати попередню семантичну індексацію.

Програмна інтеграція є особливо доцільною в освітніх експериментах, коли необхідно дослідити вплив зміни формулювання запиту на якість відповіді моделі. Студенти можуть варіювати рівень деталізації, порівнювати результати та оцінювати точність інтерпретації.

Оцінювання якості роботи моделі передбачає визначення коректності ідентифікації об'єктів, повноти переліку та логічної узгодженості опису. Важливим аспектом є аналіз явища галюцинацій, коли модель генерує інформацію, що фактично відсутня на зображенні. У навчальному процесі це

може слугувати основою для обговорення обмежень великих мовних моделей та специфіки їх мультимодального навчання.

Також доцільно порівнювати результати LLaVA з класичними алгоритмами детекції об'єктів, що використовують згорткові нейронні мережі. На відміну від них, LLaVA не забезпечує метричної точності, проте має перевагу у формуванні пояснювального текстового опису сцени.

Для роботи з пошуком об'єктів за допомогою ШІ, дуже зручно використовувати Python. В особливості, знадобляться наступні бібліотеки:

Requests – для роботи з сервером Ollama

Base64 – для кодування зображення у формат base64

Json – для передачі даних за допомогою Json

```
import requests
import base64
import sys
import json
from pathlib import Path
```

```
OLLAMA_URL = "http://localhost:11434/api/generate"
MODEL_NAME = "llava"
```

Тепер необхідно зробити функцію, яка буде перетворювати зображення у формат base64:

```
def encode_image_to_base64(image_path: str) -> str:
    with open(image_path, "rb") as image_file:
```

```
        return base64.b64encode(image_file.read()).decode("utf-8")
```

Після цього, необхідно підготувати саму функцію для початку роботи з аналізом зображення:

```
def analyze_image(image_path: str, prompt: str) -> str:
    """Надсилає зображення та запит до Ollama."""

    if not Path(image_path).exists():
        raise FileNotFoundError(f"Файл {image_path} не знайдено")

    image_base64 = encode_image_to_base64(image_path)

    payload = {
        "model": MODEL_NAME,
        "prompt": prompt,
        "images": [image_base64],
        "stream": False
    }

    response = requests.post(OLLAMA_URL, json=payload)

    if response.status_code != 200:
        raise RuntimeError(
```

```

        f"Помилка                                запиту:
{response.status_code}, {response.text}"
    )

```

```

    result = response.json()
    return result.get("response", "")

```

Останнім кроком є функція, яка має запускатись першою під час роботи скрипту

```

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Використання: python analyze_image.py
<шлях_до_зображення>")
        sys.exit(1)

    image_path = sys.argv[1]

    user_prompt = (
        "Перелічіть усі об'єкти, що зображені на
фото, "
        "та коротко вкажіть їх приблизне
розташування."
    )

    try:
        output = analyze_image(image_path,
user_prompt)
        print("\n=== Результат аналізу ===\n")
        print(output)

    except Exception as e:
        print(f"Сталася помилка: {e}")

```

В результаті роботи скрипту ми отримуємо список елементів зображення.

Завдання лабораторної роботи

1. Використовуючи наведений в прикладі код, написати скрипт, який буде здатний відповідати на питання: чи існує введений користувачем об'єкт частиною зображення?
2. Написати невеликий скрипт, який буде шукати зображення схожі з описом користувача.
3. Проаналізувати якість роботи моделі..
4. Зробити висновки щодо можливостей та обмежень локального використання мовних моделей для пошуку об'єктів.

Лабораторна робота №5 Робота з текстовими файлами як джерелом даних для покращення відповідей моделі (RAG)

Мета

Опанувати принципи RAG, навчившись використовувати текстові файли як зовнішні джерела даних для підвищення точності відповідей моделі.

Теоретичний матеріал

Генерація з доповненням пошуку (Retrieval-Augmented Generation, RAG) набула поширення як потужний підхід до розширення можливостей мовних моделей. У цьому навчальному матеріалі розглядається побудова простої RAG-системи з використанням LangChain і Hugging Face, що демонструє поєднання механізмів пошуку документів із генерацією природномовних відповідей для підвищення їх точності та контекстної обґрунтованості.

Підхід RAG поєднує два ключові компоненти:

- **Пошук (Retrieval):** здійснення пошуку в корпусі документів з метою виявлення релевантної інформації відповідно до запиту;
- **Генерація (Generation):** використання мовної моделі для формування зв'язної відповіді на основі отриманого контексту.

Такий підхід дає змогу прив'язувати відповіді моделі до конкретних документів, що підвищує їх достовірність та обґрунтованість.

Першим етапом побудови RAG-системи є підготовка документів, які надалі використовуватимуться для збагачення промптів.

Для завантаження документів можна застосовувати готові модулі завантаження документів, доступні в документації LangChain. У наведеному прикладі використовується ручний підхід, однак документи також можуть бути завантажені за допомогою відповідних завантажувачів.

У цьому прикладі використовується набір даних у форматі поширених запитань і відповідей (FAQ) про фантастичних тварин, зокрема:

```
{
  "questions": [
    {
      "title": "What is a Drakelion?",
      "answer": "A Drakelion is a large, lion-like creature with dragon-like scales covering its body. It is known for its fearsome roar and powerful leaps."
    },
    {
      "title": "Where do Drakelions live?",
      "answer": "Drakelions typically inhabit mountainous regions, often nesting in caves near volcanoes or in rocky cliffs."
    }
  ]
}
```

Набір даних завантажується вручну шляхом створення об'єктів документів:

```
from langchain.docstore.document import Document

documents = []
for question in data['questions']:
    documents.append(
        Document(
            page_content=question['answer'],
            metadata={'title': question['title']}
        )
    )
```

Під час побудови RAG-системи надзвичайно важливо розбивати контент на менші фрагменти, щоб обсяг контексту, який передається до великої мовної моделі (LLM), не був надмірним. Це має особливе значення з огляду на те, що багато комерційних сервісів LLM здійснюють оплату залежно від кількості токенів у запиті. Некоректний поділ може призвести як до збільшення вартості обчислень, так і до перевищення максимально допустимого обсягу контексту.

Поділ на фрагменти (chunk-based splitting)

Поділ на фрагменти є простим у реалізації, проте має низку суттєвих недоліків. Такий підхід може довільно розривати речення або навіть окремі слова, що призводить до втрати смислового контексту та зниження зрозумілості. Крім того, у разі нерівномірного використання розділювачів розмір фрагментів може значно варіюватися, ускладнюючи подальшу обробку.

Семантичний поділ тексту

Більш досконалим підходом є семантичний поділ тексту, який охоплює, зокрема, поділ за реченнями та абзацами:

- **Поділ за реченнями** використовує методи обробки природної мови для виокремлення завершених речень, що сприяє збереженню контексту та покращенню читабельності;
- **Поділ за абзацами** є доцільним для великих документів, оскільки дозволяє зберігати загальну структуру та логічну цілісність тексту.

У наведеній реалізації застосовується **CharacterTextSplitter**, який розбиває текст відповідно до заданого розміру фрагмента та розділювача:

```
text_splitter = CharacterTextSplitter(  
    separator='\n\n',  
    chunk_size=200,  
    chunk_overlap=0  
)  
splitted_documents =  
text_splitter.split_documents(documents)
```

Векторне сховище — це тип бази даних, призначений для ефективного зберігання та пошуку векторів, тобто числових представлень даних. Пошук у такому сховищі здійснюється не за точними збігами, а за семантичною подібністю.

Для збереження документів у векторному сховищі текст попередньо перетворюється у векторні представлення за допомогою **SentenceTransformer** з бібліотеки *sentence-transformers*.

У межах реалізації виконуються такі кроки:

- **Генерація ембедингів:** за допомогою SentenceTransformer створюються щільні векторні представлення речень або текстових фрагментів, які відображають їх семантичний зміст;
- **Збереження у векторному сховищі:** згенеровані ембедінги зберігаються у FAISS для ефективного пошуку за подібністю.

Поєднання SentenceTransformer та FAISS забезпечує можливість семантичного пошуку, за якого порівняння текстів відбувається на основі їх змісту, а не лише за ключовими словами.

```
from langchain_huggingface import
HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS

embedding_model =
HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vector_store =
FAISS.from_documents(splitted_documents, embedding_model)
```

Пошук за подібністю та генерація відповіді

Завершальний етап передбачає:

- виконання пошуку за подібністю для виявлення релевантних документів;
- використання знайденого контексту для генерації відповіді мовною моделлю.

```
results = vector_store.similarity_search(query, k=2)
context = results[0].page_content
```

Далі визначається шаблон промпту:

```
System: You are an expert in fantasy animals. To
answer consider the 'Extra Information' provided.
If you don't know the answer respond with "I don't
know the answer."
```

Extra information: {context}

Query: {query}

Response:

Розглянемо практичний приклад:

Запит:

«Which is the food of the Drakelion?»

Отриманий контекст:

«Drakelions are carnivores, feeding primarily on large mammals such as mountain deer, wild boars, and sometimes even smaller dragons.»

Відповідь:

«Drakelions feed primarily on large mammals such as mountain deer, wild boars, and sometimes even smaller dragons.»

Рекомендації та найкращі практики

- **Поділ документів:** стратегія поділу має відповідати типу контенту та конкретному сценарію використання;
- **Контекстне вікно:** необхідно враховувати обмеження на розмір контексту мовної моделі;
- **Вибір моделі ембедингів:** модель має відповідати завданням семантичного пошуку;

Стратегія пошуку: кількість документів, що вилучаються (параметр k), слід обирати з урахуванням вимог до повноти та точності відповіді.

Застосування підходу RAG уможливорює створення більш точних, інформативних і контекстно обґрунтованих систем на основі великих мовних моделей.

Завдання лабораторної роботи

1. Дослідити підхід RAG шляхом підключення текстових файлів як джерел знань.

2. Реалізувати пошук релевантних фрагментів, інтегрувати їх у запит до моделі та оцінити вплив цього підходу на якість, повноту й точність відповідей.

Лабораторна робота №6 Створення інтерфейсу у вигляді чат-бота для бібліотеки Transformers від HuggingFace

Мета

Полягає у набутті практичних навичок розробки чат-бота для платформи Telegram з використанням бібліотеки Transformers від HuggingFace.

Теоретичний матеріал

Telegram-боти є програмними агентами, що функціонують у межах платформи месенджера Telegram і забезпечують автоматизовану взаємодію з користувачами за допомогою текстових повідомлень, команд та інших типів контенту. З погляду архітектури Telegram-бот є клієнтсько-серверним застосунком, у якому серверна частина реалізується розробником, а клієнтська — інтегрована в інфраструктуру платформи Telegram. Користувачі взаємодіють із ботом через стандартний інтерфейс чату, не потребуючи встановлення додаткового програмного забезпечення.

Ключовим елементом функціонування ботів є Telegram Bot API — прикладний програмний інтерфейс, який надає уніфікований спосіб обміну даними між серверами Telegram і сервером бота. Bot API ґрунтується на протоколі HTTPS та використовує формат JSON для передавання запитів і відповідей. Кожен бот ідентифікується унікальним токеном доступу, який використовується для автентифікації під час звернення до API та забезпечує контроль доступу до функцій бота.

Отримання повідомлень від користувачів може здійснюватися двома основними способами: за допомогою механізму **long polling** або через **webhook**. У випадку long polling сервер бота періодично надсилає запити до Bot API з метою перевірки наявності нових оновлень (updates). Такий підхід є простішим у реалізації та не потребує публічного HTTPS-сервера, однак може бути менш ефективним з точки зору затримок і використання ресурсів. Webhook, натомість, передбачає реєстрацію URL-адреси, на яку сервер Telegram автоматично

надсилає HTTP-запити з новими повідомленнями одразу після їх надходження, що забезпечує більш оперативну обробку подій.

Обробка повідомлень у Telegram-ботах базується на аналізі структури об'єкта `update`, який може містити текстові повідомлення, команди, мультимедійні файли або службову інформацію. Серверна логіка бота зазвичай включає етапи валідації вхідних даних, визначення типу повідомлення та маршрутизації його до відповідного обробника. Така модульна структура спрощує масштабування та підтримку бота, а також дозволяє інтегрувати зовнішні сервіси, зокрема моделі обробки природної мови.

Таким чином, архітектура Telegram-ботів поєднує простоту клієнтської взаємодії з гнучкістю серверної реалізації, а Telegram Bot API виступає універсальним інтерфейсом, що забезпечує надійний та стандартизований обмін повідомленнями між користувачами і програмними агентами.

Для створення та подальшого використання чат-бота в месенджері Telegram необхідно отримати унікальний токен доступу, який слугує засобом автентифікації бота при взаємодії з Telegram Bot API. Процес генерації токена здійснюється за допомогою спеціального службового бота платформи.

Першим кроком є відкриття офіційного чату з ботом BotFather, який виконує функції керування Telegram-ботами. Після початку діалогу з BotFather користувач повинен ініціювати створення нового бота, використовуючи відповідну команду. У відповідь система запропонує вказати назву бота, яка відобразатиметься користувачам у чаті, а також унікальне ім'я користувача (`username`), що має закінчуватися суфіксом *bot* і використовуватиметься для ідентифікації бота в межах платформи.

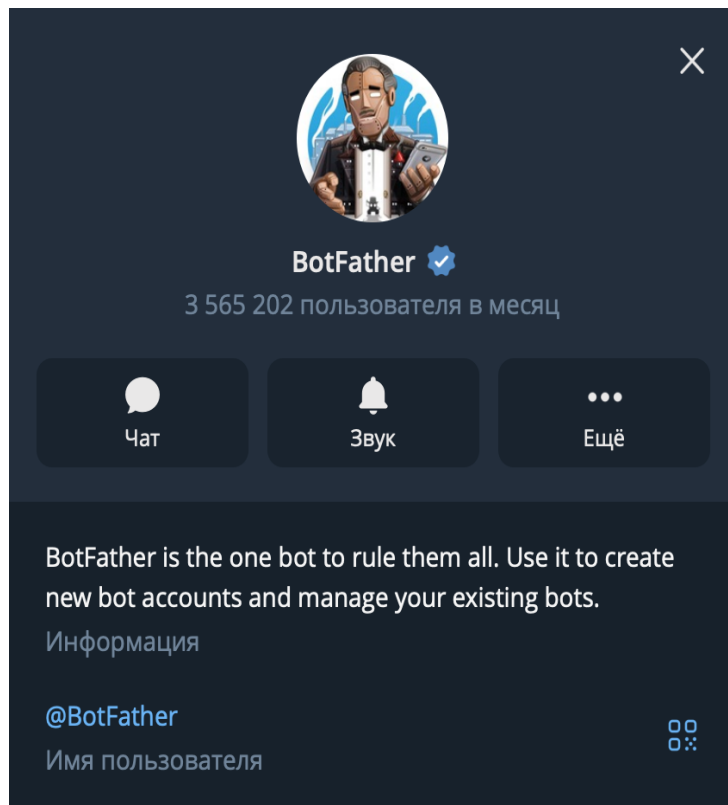


Рисунок 5



Рисунок 6

Після успішного створення бота BotFather автоматично генерує токен доступу. Даний токен являє собою символічний рядок, який надає повний доступ

до функціоналу бота через Telegram Bot API. Токен використовується у серверній частині застосунку для надсилання запитів до API, зокрема для отримання повідомлень від користувачів та відправлення відповідей.

Отриманий токен слід зберігати конфіденційно, оскільки будь-яка особа, що володіє ним, може керувати ботом від імені його власника. У разі компрометації токена BotFather надає можливість його відкликання та генерації нового ключа доступу. Таким чином, створення токена є обов'язковим етапом розгортання Telegram-чат-бота та передумовою його коректної і безпечної роботи.

Для початку роботи з Telegram API необхідно створити пустий проект, який ми будемо використовувати для роботи з бібліотекою Transformers:

```
from telegram import Update
from telegram.ext import ApplicationBuilder,
CommandHandler, ContextTypes

if __name__ == '__main__':
    app = ApplicationBuilder().token("Сюди необхідно
додати ваш токен").build()
    app.run_polling()
```

Таким чином створено простий бот, який на даний момент не вміє абсолютно нічого. Оскільки цей бот використовує Python як основну мову програмування, можна додати код із лабораторної роботи №3 для використання штучного інтелекту.

```
from transformers import pipeline
summarizer = pipeline("summarization")
```

Тепер код вміщує в себе модель штучного інтелекту, яка буде використовуватись для суммаризації тексту. Наступним кроком необхідно створити функцію, яка буде використовуватись у якості триггера на повідомлення користувача.

```

    async def ask(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
        message = update.message.text
        response = summarizer(message)
        await
update.message.reply_text(response[0]["summary_text"])

```

Для її активації необхідно додати відповідний запис у функцію main

```

app.add_handler(CommandHandler("ask", ask))

```

Таким чином повний лістинг коду виглядає наступним чином:

```

from telegram import Update
from telegram.ext import ApplicationBuilder,
CommandHandler, ContextTypes
from transformers import pipeline

summarizer = pipeline("summarization")

async def ask(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    message = update.message.text
    response = summarizer(message)
    await
update.message.reply_text(response[0]["summary_text"])

if __name__ == '__main__':
    app = ApplicationBuilder().token("Сюди необхідно
додати ваш токен").build()
    app.add_handler(CommandHandler("ask", ask))
    app.run_polling()

```

Цей бот стає основою для взаємодії користувача з моделлю штучного інтелекту через зручний та доступний інтерфейс Telegram. Також можливо додати до бота повноцінний RAG-пошук аби вдосконалити його використання у якості асистента з певних тем.

Завдання лабораторної роботи

1. Створити бота, який буде генерувати текст по заданному шаблону (див. text-generation моделі)

1. Додати функціональність генерувати декілька текстів одночасно, щоб користувач міг обрати саме той варіант, який буде кращий.

2. Створити бота, який буде відповідати на запитання (question-answering), та використовувати текстовий файл (RAG) як основну векторну базу даних.

Лабораторна робота №7 Аналіз метричних даних за допомогою ШІ на основі JSON. Створення моделей.

Мета

Формування практичних навичок автоматизованого збору базових апаратних характеристик комп'ютерної системи, зокрема інформації про центральний процесор (CPU) та оперативну пам'ять (RAM), із подальшим аналізом і структурованим поданням отриманих даних у форматі JSON за допомогою засобів штучного інтелекту.

Теоретичний матеріал

Бібліотека **psutil** є кросплатформним інструментом мови програмування Python, призначеним для отримання інформації про стан та ресурси операційної системи. Вона надає програмний інтерфейс для доступу до даних про апаратне забезпечення та системні ресурси без необхідності використання низькорівневих системних викликів або зовнішніх утиліт. Завдяки цьому psutil широко застосовується у задачах системного моніторингу, діагностики продуктивності, автоматизованого адміністрування та збору телеметрії.

Збір інформації про центральний процесор (CPU) за допомогою psutil охоплює отримання як статичних, так і динамічних характеристик. До статичних параметрів належать кількість фізичних та логічних ядер, а також базова архітектурна інформація. Динамічні показники включають поточне завантаження процесора у відсотках, як загальне, так і по окремих ядрах, а також частоту роботи CPU. Отримані дані дозволяють оцінити обчислювальне навантаження системи та можуть використовуватися для аналізу ефективності програм або виявлення перевантажень.

Інформація про оперативну пам'ять (RAM), що надається бібліотекою psutil, включає загальний обсяг пам'яті, кількість використаної та вільної пам'яті, а також відсоток її завантаження. Окремо можуть бути отримані дані про віртуальну пам'ять і механізми підкачки, що є важливими для розуміння

реального стану пам'яттєвих ресурсів системи. Структуроване отримання таких показників у середовищі Python 3 створює передумови для подальшої автоматизації аналізу, формування звітів у форматі JSON та інтеграції з інтелектуальними системами обробки даних.

На великих проектах зазвичай формується величезна кількість інформації, яку розробнику потрібно розуміти та розшифрувати, аби бути готовим до прийняття рішень. В нашому випадку, ми створимо ШІ-помічника, який буде отримувати усю необхідну інформацію за допомогою бібліотеки psutils, та відправляти нам готовий результат у вигляді повідомлення про стан нашої системи. Такий засіб буде дуже зручний у випадку роботи з саморобними пристроями на базі нанокомп'ютерів формату Raspberry Pi чи ESP32. Для збору інформації буде використаний наступний скрипт:

```
import psutil
import time

psutil.cpu_percent(interval=None, percpu=False)
time.sleep(1) # Інформація про використання CPU має
збиратися із затримкою
print ("CPU: " +
str(psutil.cpu_percent(interval=None, percpu=False)) +
"%")
print ("RAM: " + str(psutil.virtual_memory()))
```

Як результат роботи на екрані буде виведено інформацію про використання процесора та оперативної пам'яті:

```
CPU: 3.4%
RAM: svmem(total=25769803776, available=11086364672,
percent=57.0, used=10638065664, free=3226632192,
active=7867809792, inactive=7818412032, wired=2770255872)
```

Тепер необхідно підготувати інформацію у формат json для створення зручного шаблону передачі даних до ШІ:

```

import psutil
import time
import json

psutil.cpu_percent(interval=None, percpu=False)
time.sleep(1) # Інформація про використання CPU має
збиратися із затримкою
cpuUsage = psutil.cpu_percent(interval=None,
percpu=False)
ramUsage = psutil.virtual_memory()
result = {
    "CPU" : cpuUsage,
    "RAM" : {
        "free" : ramUsage.free,
        "total" : ramUsage.total,
        "used" : ramUsage.used,
        "percent" : ramUsage.percent,
    }
}
print(json.dumps(result))

```

Відтепер ми приготували певний код, який може збирати інформацію, та підготовлювати її у форматі JSON. Наступний крок - підготувати модель ШІ. Для цього необхідно використати функціонал *ollama*, а саме *create model*.

Створення моделі використовуючи **modelfile**.

Modelfile є текстовим файлом, у якому послідовно задаються директиви (інструкції). Базовою директивою є FROM, що визначає джерело, на основі якого формується ваша модель: це може бути вже доступна в Ollama модель (наприклад, llama3, gemma3), локальний файл моделі (наприклад, GGUF), або інший підтримуваний формат імпорту. Далі можуть застосовуватися директиви, які керують тим, як модель інтерпретує вхідні повідомлення та як генерує вихід: SYSTEM (системне повідомлення/роль), TEMPLATE (шаблон формування промпу), PARAMETER (параметри генерації на кшталт температури, num_predict, стоп-послідовностей тощо), ADAPTER (підключення адаптерів), LICENSE (ліцензійні умови) та MESSAGE (заздалегідь визначені повідомлення/інструкції у складі рецепта).

Саме за допомогою Modelfile створимо необхідну модель для роботи разом з ollama. Оскільки у минулих роботах більшу частину часу були використані бібліотеки Transformers, цього разу буде використаний окремий веб-сервіс з використанням OpenAI API.

Для підготування моделі використано наступний Modelfile:

```
FROM llama3.2
# Set a custom system message
SYSTEM "You are an monitoring operator. User will
write you json with information about CPU and RAM of the
computer. You need provide answer only in JSON, which
will contain 'message' key with summarization of all data
from user json. Any additional text should be provided in
'additional' key"
# Adjust parameters
```

В даному випадку ми створили окрему модель із завчасно підготовленим промптом, який використовується для пояснення моделі на базі Llama 3.2 основної суті її роботи.

Для того, аби збудувати модель, необхідно звернутися до ollama та за допомогою аргументу *create* створити саму модель.

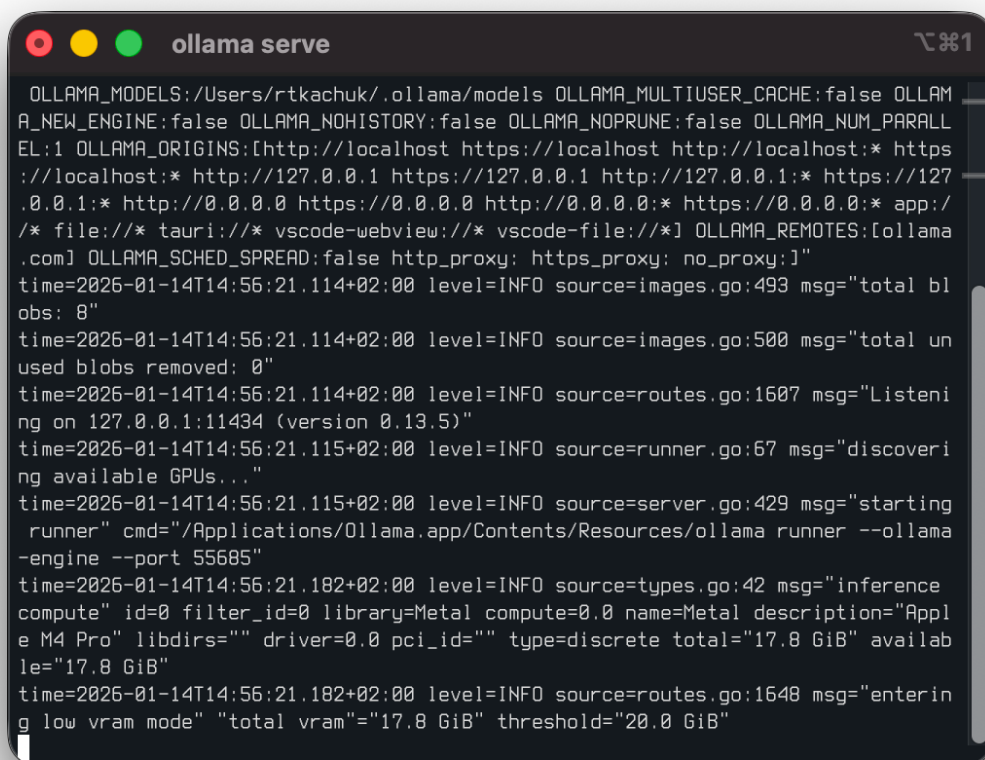
```
ollama create -f MODELFILE monitoring-assistant
```

Таким чином при перевірці існуючих моделей за допомогою *ollama list* отримуємо наступний список:

	NAME	ID	SIZE
	MODIFIED		
	monitoring-assistant:latest	7a42c1b3fd9b	2.0
GB	12 minutes ago		
	llama3.2:latest	a80c4f17acd5	2.0
GB	7 months ago		

Що підтверджує існування нової моделі *monitoring-assistant*. Тепер запустимо сам сервер ollama за допомогою *ollama serve*. Важливий момент - сервер у вас може вже працювати, оскільки по замовчуванню ollama починає працювати одразу після запуску операційної системи. Тому якщо у вас виникне помилка на кшталт "Address already in use" - сервер ШІ вже працює.

В іншому випадку, в терміналі відтвориться наступний лог:



```
ollama serve
OLLAMA_MODELS:/Users/rtkachuk/.ollama/models OLLAMA_MULTIUSER_CACHE:false OLLAMA_NEW_ENGINE:false OLLAMA_NOHISTORY:false OLLAMA_NOPRUNE:false OLLAMA_NUM_PARALLEL:1 OLLAMA_ORIGINS:[http://localhost https://localhost http://localhost:* https://localhost:* http://127.0.0.1 https://127.0.0.1 http://127.0.0.1:* https://127.0.0.1:* http://0.0.0.0 https://0.0.0.0 http://0.0.0.0:* https://0.0.0.0:* app://* file://* tauri://* vscode-webview://* vscode-file://*] OLLAMA_REMOTES:[ollama.com] OLLAMA_SCHED_SPREAD:false http_proxy: https_proxy: no_proxy:]
time=2026-01-14T14:56:21.114+02:00 level=INFO source=images.go:493 msg="total blobs: 0"
time=2026-01-14T14:56:21.114+02:00 level=INFO source=images.go:500 msg="total unused blobs removed: 0"
time=2026-01-14T14:56:21.114+02:00 level=INFO source=routes.go:1607 msg="Listening on 127.0.0.1:11434 (version 0.13.5)"
time=2026-01-14T14:56:21.115+02:00 level=INFO source=runner.go:67 msg="discovering available GPUs..."
time=2026-01-14T14:56:21.115+02:00 level=INFO source=server.go:429 msg="starting runner" cmd="/Applications/Ollama.app/Contents/Resources/ollama runner --ollama-engine --port 55685"
time=2026-01-14T14:56:21.182+02:00 level=INFO source=types.go:42 msg="inference compute" id=0 filter_id=0 library=Metal compute=0.0 name=Metal description="Apple M4 Pro" libdirs="" driver=0.0 pci_id="" type=discrete total="17.8 GiB" available="17.8 GiB"
time=2026-01-14T14:56:21.182+02:00 level=INFO source=routes.go:1648 msg="entering low vram mode" "total vram"="17.8 GiB" threshold="20.0 GiB"
```

Рисунок 7

Тепер ШІ чекає наших запитів, і працює як окремий сервіс виключно для запитів з локальної машини. Для відправлення запиті необхідно використати бібліотеку requests та OpenAI API.

OpenAI-сумісний API в **Ollama** призначений для забезпечення уніфікованого доступу до локально розгорнутих великих мовних моделей через

інтерфейс, сумісний із офіційною специфікацією OpenAI API. Такий підхід дозволяє використовувати вже наявні клієнтські бібліотеки, інструменти та програмні рішення, спочатку розроблені для хмарних сервісів OpenAI, без суттєвих змін у коді. Фактично Ollama реалізує локальний сервер, який відтворює структуру запитів і відповідей OpenAI API, зберігаючи при цьому повний контроль над даними та обчислювальними ресурсами на стороні користувача.

Архітектурно OpenAI-сумісний API Ollama надає REST-інтерфейс, який підтримує ключові кінцеві точки для генерації тексту та діалогової взаємодії, зокрема у форматі chat-completion. Запити надсилаються у вигляді JSON-об'єктів, що містять опис ролей повідомлень (system, user, assistant), параметри генерації та ідентифікатор моделі. У відповідь сервер Ollama повертає структуровані дані, аналогічні відповідям OpenAI API, що дозволяє інтегрувати локальні моделі у клієнтські застосунки, веб-сервіси або агентні системи без прив'язки до зовнішньої інфраструктури.

Для відправлення запитів необхідно використовувати JSON наступного формату:

```
{
  "model": "monitoring-assistant",
  "prompt": "{ CPU : 50 }",
  "stream": false
}
```

Де *model* - це назва нашої завчасно підготовленої моделі, *prompt* - наш JSON згідно з описом ролі нашої моделі, та *stream* - означає, що відповідь буде надана одним об'єктом, замість відправки кожного токена окремо.

Відтепер, за допомогою бібліотеки Requests, доповнимо код аби відправити наш запит. Після цього повний лістинг програми виглядає наступним чином:

```

import psutil
import time
import json
import requests

psutil.cpu_percent(interval=None, percpu=False)
time.sleep(1) # Інформація про використання CPU має
збиратися із затримкою
cpuUsage = psutil.cpu_percent(interval=None,
percpu=False)
ramUsage = psutil.virtual_memory()
data = {
    "CPU" : cpuUsage,
    "RAM" : {
        "free" : ramUsage.free,
        "total" : ramUsage.total,
        "used" : ramUsage.used,
        "percent" : ramUsage.percent,
    }
}
print(json.dumps(data))
payload = {
    "model": "monitoring-assistant",
    "prompt": json.dumps(data),
    "stream": False
}
response =
requests.post("http://127.0.0.1:11434/api/generate",
json=payload)
print (response.json()["response"])

```

Важливо розуміти, що `payload` - це інформація, яка буде відправлена `ollama`, а `data` - інформація, що буде відправлена на пряму моделі. Для генерації відповіді використаємо шлях `/api/generate`, та стандартний порт 11434. Із повного `json` відповіді (`response`) необхідно отримати лише `response`. Таким чином відповідь від моделі буде висновком усіх даних, що були згенеровані від `psutil`

```

{"CPU": 3.1, "RAM": {"free": 1057685504, "total":
25769803776, "used": 11420319744, "percent": 61.9}}
{

```

```
"message": "The computer is running at a moderate
load state with 61.9% of its RAM being used and the CPU
utilization is 3.1%."
}
```

Верхній JSON - це об'єкт, що був надісланий моделі, а наступний JSON - це відповідь самої моделі. Можна зрозуміти, що ШІ цілком проаналізував відправлені дані, та сформував зрозумілий для людини висновок, який можна використовувати у повсякденному житті.

На практиці розробники часто використовують більш детальні сервіси на кшталт *node_exporter*, проте для нас на даний момент цілковито достатньо і самописного нескладного варіанту.

Завдання лабораторної роботи

1. Написати програму, яка буде зчитувати повну інформацію про комп'ютер (а саме включаючи температуру та інформацію про накопичувачі)
2. Створити модель, яка буде розшифровувати інформацію та повертати висновок включаючи додатковий опис. До прикладу, поради що робити якщо температура зависока.
3. Доповнити програму запитам до нової моделі ШІ. Оформити автоматичний режим роботи (відправляти запит з оновленою інформацією кожні 60 секунд). Оформити текстовий інтерфейс програми, аби замість усіх JSON користувач міг побачити тільки потрібний текст.

Лабораторна робота №8 Генерація мультимодального контенту на основі готових моделей ШІ

Мета

Формування теоретичних знань і практичних навичок використання готових моделей штучного інтелекту для генерації мультимодального контенту, зокрема текстового, візуального

Теоретичні матеріали

Мультимодальні моделі класу *image-text-to-text* належать до сучасних систем штучного інтелекту, здатних обробляти та інтегрувати інформацію з різних модальностей, зокрема зображень і тексту, з подальшою генерацією текстової відповіді. Основною ідеєю таких моделей є поєднання візуального та лінгвістичного представлення даних у єдиному семантичному просторі, що дозволяє здійснювати глибший аналіз контексту та формувати змістовні, обґрунтовані текстові висновки на основі зображень і супровідних текстових запитів.

Архітектурно моделі *image-text-to-text* зазвичай складаються з двох ключових компонентів: візуального енкодера та мовної моделі-декодера. Візуальний енкодер, як правило, побудований на основі згорткових нейронних мереж або візуальних трансформерів, і виконує перетворення зображення у набір числових ознак (векторів), що відображають його зміст. Ці вектори репрезентують ключові візуальні характеристики, такі як об'єкти, їхні взаємозв'язки, просторові ознаки та семантичні атрибути.

Текстова складова мультимодальної моделі представлена мовною моделлю, яка може базуватися на трансформерній архітектурі. Вона відповідає за інтерпретацію текстового запиту користувача та генерацію вихідного тексту. Для забезпечення мультимодальної взаємодії візуальні ознаки, отримані з

енкодера зображень, інтегруються у вхідний контекст мовної моделі. Це може реалізовуватися шляхом конкатенації векторів, використання спеціальних токенів або механізмів крос-уваги, які дозволяють моделі зіставляти текстові елементи з відповідними візуальними фрагментами.

Процес роботи моделей *image-text-to-text* передбачає одночасний аналіз двох джерел інформації: зображення та тексту. Наприклад, користувач може подати зображення разом із запитанням або інструкцією, а модель, спираючись на візуальний зміст та мовний контекст, формує текстову відповідь. Такий підхід забезпечує можливість виконання складних завдань, зокрема опису зображень, відповіді на запитання за вмістом зображення, інтерпретації сцен або пояснення візуальних даних у зрозумілій текстовій формі.

Важливою особливістю мультимодальних моделей є їх навчання на великих наборах даних, що містять пари «зображення — текст». Під час навчання модель вчиться встановлювати відповідність між візуальними патернами та мовними описами, що дозволяє їй узагальнювати знання та коректно працювати з новими, раніше невідомими прикладами. Якість роботи таких моделей значною мірою залежить від різноманітності та репрезентативності навчальних даних, а також від способів вирівнювання візуальних і текстових ознак у спільному просторі представлень.

Для роботи з таким типом моделей необхідно використовувати набір бібліотек від HuggingFace, в особливості нас цікавлять моделі *lava* від *llama*.

Моделі **LLaVA (Large Language and Vision Assistant)** належать до класу мультимодальних систем штучного інтелекту, призначених для спільної обробки візуальної та текстової інформації з подальшою генерацією змістовних текстових відповідей. Основною метою створення LLaVA є поєднання потужностей великих мовних моделей із можливостями комп'ютерного зору, що дозволяє системі інтерпретувати зображення у контексті природномовних запитів користувача та виконувати складні завдання в режимі діалогу.

Архітектурно LLaVA ґрунтується на модульному підході, у межах якого використовується попередньо навчений візуальний енкодер та велика мовна модель. Візуальний компонент, як правило, базується на трансформерних архітектурах для аналізу зображень і відповідає за перетворення вхідних візуальних даних у компактне числове представлення. Отримані візуальні ознаки передаються до мовної моделі, яка інтегрує їх у свій контекст і використовує для генерації текстових відповідей.

Ключовою особливістю моделей LLaVA є механізм узгодження (alignment) між візуальними та мовними представленнями. Для цього застосовуються спеціальні проєкційні шари або адаптаційні модулі, які переводять візуальні ознаки у формат, сумісний із внутрішнім простором представлень мовної моделі. Такий підхід забезпечує ефективну взаємодію між модальностями та дозволяє моделі коректно пов'язувати елементи зображення з відповідними мовними концептами.

Навчання моделей LLaVA зазвичай здійснюється у декілька етапів. На початковому етапі використовується велика кількість пар «зображення — текст», що дозволяє моделі засвоїти базові відповідності між візуальними об'єктами та їх мовними описами. Подальше донавчання орієнтоване на діалогові сценарії, у яких модель навчається відповідати на запитання, пояснювати зображення та виконувати інструкції, спираючись одночасно на візуальний і текстовий контекст.

Функціонально моделі LLaVA застосовуються для широкого спектра завдань, зокрема опису зображень, візуально-орієнтованих запитань і відповідей, аналізу сцен, пояснення графічної інформації та підтримки мультимодального діалогу. Завдяки інтеграції з великими мовними моделями такі системи здатні не лише розпізнавати об'єкти, але й формувати логічні, контекстно обґрунтовані текстові висновки.

Для використання моделі, напишемо наступний код

```
from transformers import pipeline
pipe = pipeline("image-text-to-text", model="llava-
hf/llava-interleave-qwen-0.5b-hf")
```

Створюємо конвеєр з необхідною моделлю lava, та почнемо з нею працювати. Моделі необхідно використати json, з досить детальним описом того, що потрібно робити

```
messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "image",
                "image":
"https://huggingface.co/datasets/huggingface/documentatio
n-images/resolve/main/bee.jpg",
            },
            {"type": "text", "text": "Describe this
image."},
        ],
    }
]
```

В даному випадку, створюємо список messages, в якому описано всі картинки, з якими необхідно працювати. Оскільки в прикладі використано лише одну картинку - об'єкт також буде один.

В Json-об'єкті описано основну роль, від імені якої відправлено повідомлення (може бути system або user, та власне сам контент. Перша частина - зображення (type: image), лінку до якого додано у поле image. Друга частина - це текст, в якому необхідно описати завдання, що саме необхідно зробити. Повний лістинг програми виглядає наступним чином

```
from transformers import pipeline

pipe = pipeline("image-text-to-text", model="llava-
hf/llava-interleave-qwen-0.5b-hf")
```

```

messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "image",
                "image":
"https://huggingface.co/datasets/huggingface/documentatio
n-images/resolve/main/bee.jpg",
            },
            {"type": "text", "text": "Describe this
image."},
        ],
    }
]

outputs = pipe(text=messages, max_new_tokens=60,
return_full_text=False)

print (outputs[0]["generated_text"])

```

Після запуску, отримаємо детальний опис зображення

The image captures a moment of tranquility in nature. At the center of the frame, a pink flower with a white center is the main subject. The flower is in full bloom, its petals spread out in a display of beauty. The flower is surrounded by a cluster of red flowers, their vibrant

Оскільки модель досить маленька, опис не буде детальним.

Завдання лабораторної роботи

1. Підготувати програму, яка може отримувати інформацію з декількох зображень.
2. Додати до програми можливість відповідати на питання по вказаним зображенням (question-answering моделі бібліотеки Transformers)

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

Базова

1. Huyen, C. AI Engineering: Building Applications with Foundation Models. — Sebastopol : O'Reilly Media, 2023. — 432 p.
3. Theobald, O. Machine Learning for Absolute Beginners: A Plain English Introduction. — Scatterplot Press, 2017. — 164 p.
4. Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach. — 4th ed. — Hoboken, NJ : Pearson, 2021. — 1136 p.
5. Alamar, J., van Heeswijk, M. Hands-On Large Language Models: Language Understanding and Generation. — Sebastopol : O'Reilly Media, 2024. — 400 p.
6. Raschka, S., Liu, Y., Mirjalili, V., Dzhulgakov, D. Machine Learning with PyTorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python. — Birmingham : Packt Publishing, 2022. — 770 p.
7. Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach, Global Edition. 4th ed. Pearson, 2021. ISBN 9781292401133.
8. Luger G. F. Artificial Intelligence: Principles and Practice. Springer, 2024. ISBN 9783031574368.
9. Hao K. Empire of AI: Dreams and Nightmares in Sam Altman's OpenAI. Penguin Press, 2025. ISBN 9780593657508.
10. Gans J. The Microeconomics of Artificial Intelligence. MIT Press, 2025. ISBN 9780262553544.
11. Portner B. AI Stability Imperative: Human Co-Evolution. Muon Publishing Group, 2025. ASIN B0G3JWYTRD.
12. Wang J. What You Need to Know About AI. Manuscripts LLC, 2025. ISBN 9798889265825.

Допоміжна

1. Jurafsky, Daniel, Martin, James H. *Speech and Language Processing* (3rd Edition, Draft). 2023.
2. LLaMA-Factory GitHub. <https://github.com/hiyouga/LLaMA-Factory>
3. Medium – *Build a Locally Running Voice Assistant*. <https://medium.com/data-science/build-a-locally-running-voice-assistant-2f2ead904fe9>
4. HuggingFace Transformers documentation <https://huggingface.co/learn/llm-course/chapter0/1>
5. Python Documentation. <https://docs.python.org/3/>
6. Bradfield D., Allan S. *Governing the Machine: navigate AI, unlock its potential*. Bloomsbury Business, 2025. ISBN 9781399426298.
7. Fraley S. M. *AI for Beginners: Artificial Intelligence for Beginners With Real World Applications*. 2025. ASIN B0F2ZL16MQ.
8. Shannon K. *The Unfinished Role of the Artist in Artificial Intelligence*. Backbeat Books, 2025. ISBN 9781493073768.
9. Ackerman M. *Creative Machines: AI, Art & Us* [електронне видання]. Wiley, 2025. ASIN B0FSYPB6WM.
10. Papermaster S. *Artificial Intelligence: AI Engineer's Cheatsheet: Silicon Edition (Ultra-large scale reference)*. Independently published, 2025. ISBN 9798267517096.

Інформаційні ресурси

1. Курс «Технології штучного інтелекту» на сервері дистанційної освіти ЦНТУ. – URL: <https://moodle.kntu.kr.ua/course/view.php?id=6413>
2. ChatGPT: вебсервіс штучного інтелекту [Електронний ресурс]. Режим доступу: <https://chatgpt.com>
3. Perplexity AI: інструмент пошуку з підтримкою ШІ [Електронний ресурс]. Режим доступу: <https://www.perplexity.ai>
4. Gemini: сервіс штучного інтелекту від Google [Електронний ресурс]. Режим доступу: <https://gemini.google.com>

5. Copilot: сервіс штучного інтелекту від Microsoft [Електронний ресурс]. Режим доступу: <https://copilot.microsoft.com>
6. Онлайн-курси UDEMY. – URL: <https://www.udemy.com/> –платформа онлайн-курсів різних ІТ тематик.
7. Онлайн-курси Prometheus. – URL: <https://prometheus.org.ua/> – українська платформа безкоштовних онлайн-курсів
8. Онлайн-курси Coursera. – URL: <https://www.coursera.org> –платформа онлайн-курсів різних ІТ тематик.
9. <http://stackoverflow.com/> – система питань і відповідей для професійних програмістів та новачків у програмуванні.
10. <https://dou.ua/> – український веб-сайт з елементами колективного блогу, створений для розповсюдження новин, аналітичних статей та свіжої інформації пов'язаної із інформаційними технологіями.
11. <https://biblprog.org.ua/ua/programming/> – каталог безкоштовних середовищ розроблення ПЗ.
12. Національна бібліотека України імені В. І. Вернадського: Електронні ресурси НБУВ [Електронний ресурс]. – Режим доступу: <http://www.nbuv.gov.ua/>