**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №6

**на тему:** *“Розробка та оцінка систем пошуково-доповненої генерації (RAG).”*

**з дисципліни** *“Штучний інтелект в ігрових застосунках”*

**Лектор:**

асис. каф. ПЗ

Бауск О. Є.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

ст. викл. каф. ПЗ

Бауск О. Є.

«\_\_» \_\_\_\_\_\_\_\_\_ 2025р.

∑=\_\_\_\_\_

Львів – 2025

**Тема роботи:** Розробка та оцінка систем пошуково-доповненої генерації (RAG).

**Мета роботи:** Набути практичних навичок побудови повного конвеєра Пошуково-Доповненої Генерації (Retrieval-Augmented Generation, RAG) з нуля, що працює локально. Дослідити процес обробки документів (PDF), створення текстових ембедингів, реалізації семантичного пошуку та використання великої мовної моделі (LLM) для генерації відповідей на запити на основі знайденого контексту.

# Теоретичні відомості

**Розгортання Jupyter notebook на власній машині за допомогою WSL**

1. Встановлюємо/оновляємо NVIDIA драйвера для GPU до останньої версії через NVIDIA app.
2. Встановлюємо CUDA toolkit для Windows 11 (Основна операційна система)

**PS: Можливо цей крок не потрібний, проте better safe than sorry.**

1. Запускаємо Powershell від імені адміністратора та встановлюємо WSL:

*wsl.exe –install*

1. Пересвідчуємося, що маємо останню версію WSL ядра:  
     
   *wsl.exe --update*
2. Налаштовуємо середовище:

*wsl.exe*

За замовчуванням буде встановлено Ubuntu, не раджу це змінювати

1. Oновляємо Ubuntu:

*sudo apt update && sudo apt upgrade -y*

1. **Не встановлюємо ніяких графічних драйверів** та видаляємо старий GPG key:

*sudo apt-key del 7fa2af80*

1. Встановлюємо [CUDA toolkit](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=WSL-Ubuntu&target_version=2.0&target_type=deb_local) для Ubuntu. Приклад наведено для версії 12.8:

*$ wget* [*https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86\_64/cuda-wsl-ubuntu.pin*](https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin)

*$ sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600*

*$ wget* [*https://developer.download.nvidia.com/compute/cuda/12.8.1/local\_installers/cuda-repo-wsl-ubuntu-12-8-local\_12.8.1-1\_amd64.deb*](https://developer.download.nvidia.com/compute/cuda/12.8.1/local_installers/cuda-repo-wsl-ubuntu-12-8-local_12.8.1-1_amd64.deb)

*$ sudo dpkg -i cuda-repo-wsl-ubuntu-12-8-local\_12.8.1-1\_amd64.deb*

*$ sudo cp /var/cuda-repo-wsl-ubuntu-12-8-local/cuda-\*-keyring.gpg /usr/share/keyrings/*

*$ sudo apt-get update*

*$ sudo apt-get -y install cuda-toolkit-12-8*

1. Встановлюємо python:

*sudo apt install python3-pip python3-dev*

1. Встановлюємо virtualenv:

*sudo apt install python3-virtualenv*

1. Створюємо директорію проєкта з назвою “ai” та переходимо в неї:

$ mkdir ~/ai

$ cd ~/ai

1. Створюємо environment:

*virtualenv ai\_env*

1. Активуємо environment:

source ai\_env/bin/activate

1. Встановлюємо Jupyter notebook:

*pip install jupyter*

1. Встановлюємо tensorflow:

*python3 -m pip install tensorflow[and-cuda]*

1. Запускаємо jupyter notebook

*jupyter notebook*

1. Перевіряємо, чи tensorflow бачить відеокарту:

*from tensorflow.python.client import device\_lib*

*def get\_available\_devices():*

*local\_device\_protos = device\_lib.list\_local\_devices()*

*return [x.name for x in local\_device\_protos]*

*print(get\_available\_devices())*

**

*Рис.1. Очікуваний результат*

**Корисні посилання:**

<https://docs.nvidia.com/cuda/wsl-user-guide/index.html>

<https://learn.microsoft.com/en-us/windows/wsl/install>

<https://www.reddit.com/r/learnmachinelearning/comments/t8sadx/is_anyone_using_wsl2_ubuntu_cuda_jupyter_on/>

# Завдання

1. Відтворити RAG-конвеєр, описаний у розділі "Хід роботи", використовуючи PDF-файл підручника з нутріціології, що є у вільному доступі.
2. Протестувати роботу конвеєра, поставивши різні запитів на тему харчування (можна використати приклади з коду або придумати власні). Проаналізувати якість відповідей та релевантність знайденого контексту.
3. Провести експерименти з параметрами RAG-конвеєра, використовуючи один з документів (оригінальний або власний):
   * Змінити кількість ресурсів, що повертаються для контексту (n\_resources\_to\_return у функції retrieve\_relevant\_resources або ask, наприклад, 3, 7, 10). Проаналізувати вплив на повноту та точність відповіді для 2-3 запитів.
   * Змінити параметр temperature у функції ask (наприклад, 0.2, 1.0). Описати, як змінюється стиль та детермінованість відповіді.

# Індивідуальне завдання

Придумати свої запити до матеріалу власного .pdf документа.

# Хід роботи

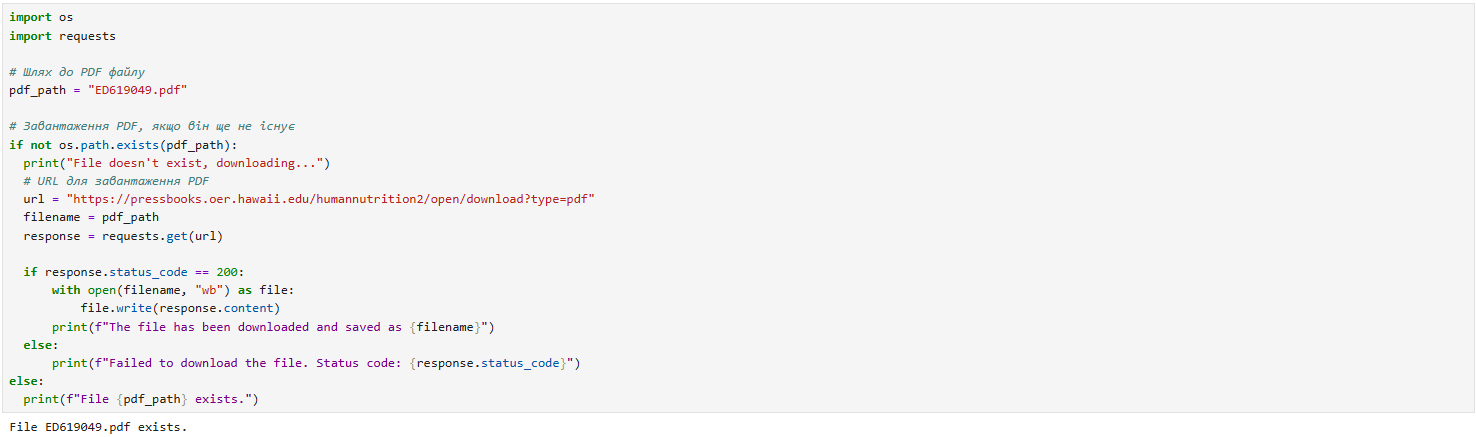
1. **Встановлення бібліотек**

Звичайне встановлення через pip, тільки torch [має бути](https://github.com/huggingface/transformers/issues/34466#issuecomment-2442180500) версії 2.4 (На версії вище Flash attention збирається вічність):

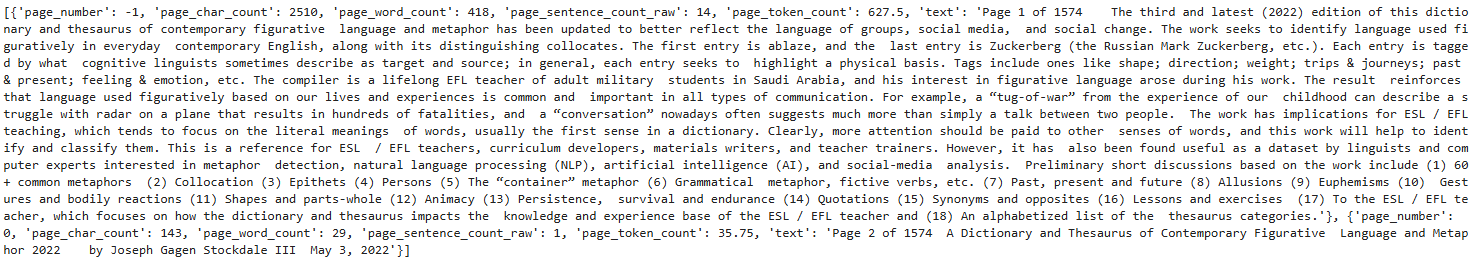
*!pip install torch=='2.4.1+cu121' torchvision=='0.19.1+cu121' torchaudio=='2.4.1+cu121' --index-url*[*https://download.pytorch.org/whl/cu121*](https://download.pytorch.org/whl/cu121)

1. **Обробка документа та створення ембедингів**

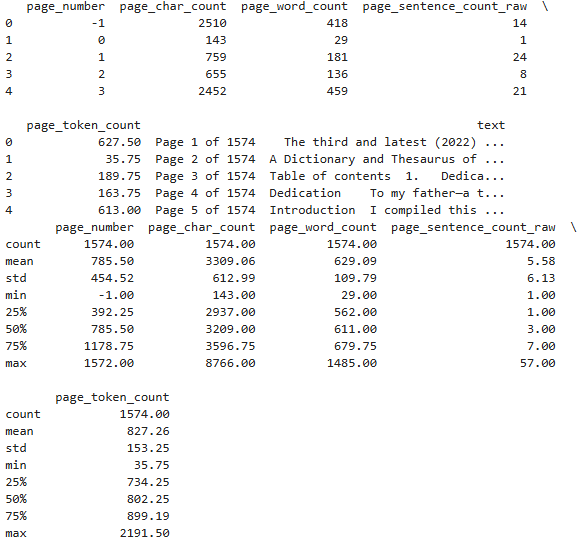
Для лабораторної роботи я взяв документ що є старим посібником-словником з англійської мови: “A Dictionary and Thesaurus of Contemporary Figurative Language and Metaphor 2022” написаний Джозефом Гагено Стокдейлом, 3-го травня 2022р



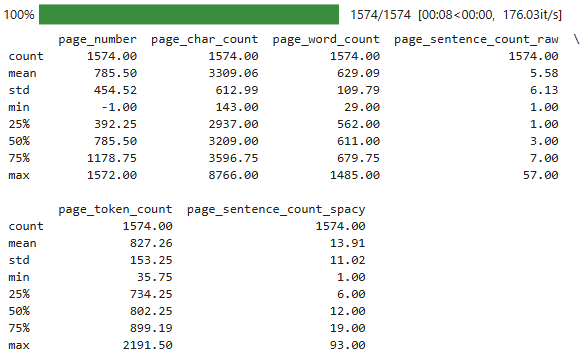
*Рис. 1. Перевірка існування файла*



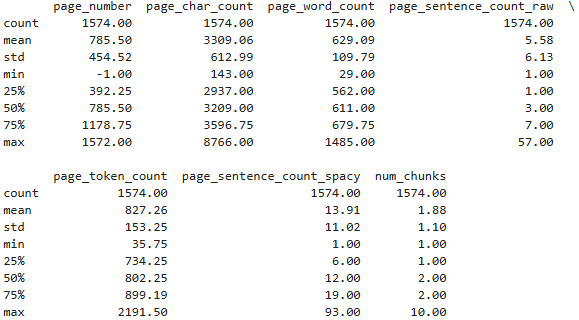
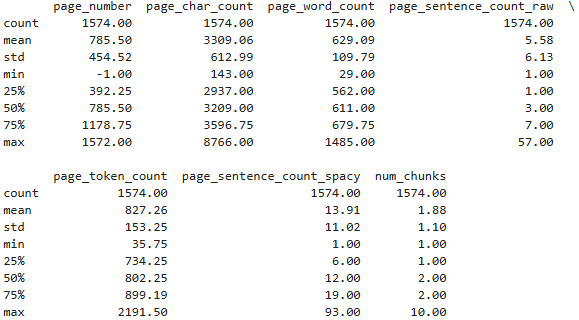
*Рис. 2. Вміст перших двох сторінок*

**

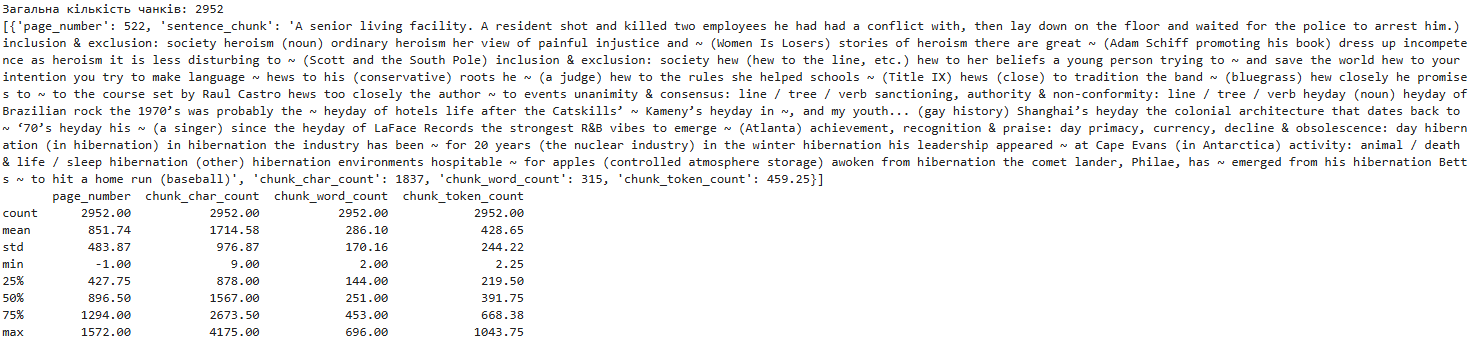
*Рис. 3. Статистика за сторінками*

**

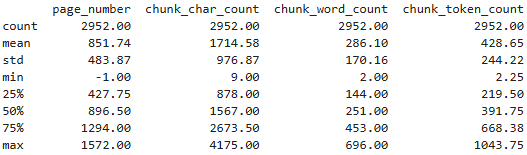
*Рис. 4. Статистика після розділення документа на речення*

**

*Рис. 5. Статистика після чанкінгу*

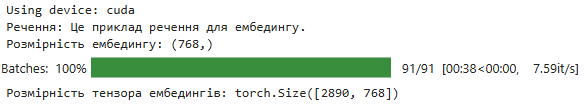
**

*Рис. 6. Окремий чанк*

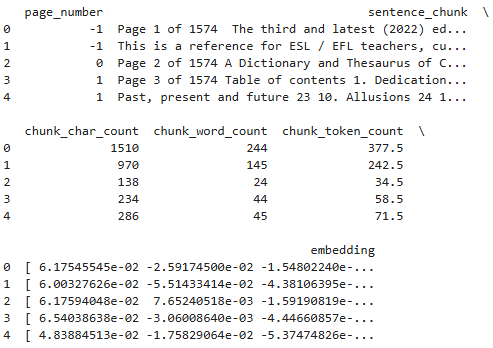
**

*Рис. 7. Статистика чанків*

Після фільтрації чанків (відкидання занадто коротких послідовностей) лишилося 2890 чанків.

**

*Рис. 8. Використання графічних CUDA ядер для перетворення чанків на ембединги.*

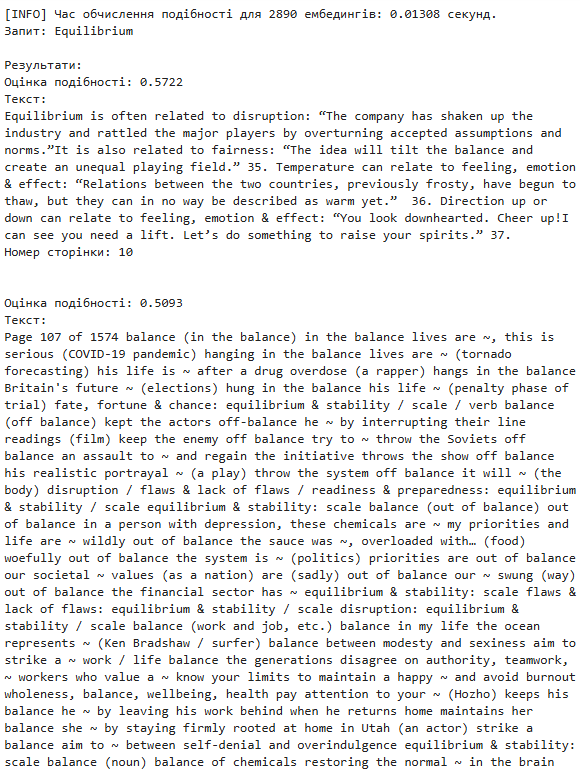
**

*Рис. 9. Перевірка завантаження збережених ембедингів*

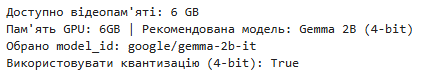
1. **RAG – Пошук та Відповідь**

**

*Рис. 10. Тензор PyTorch отриманий зі збережених чанків*

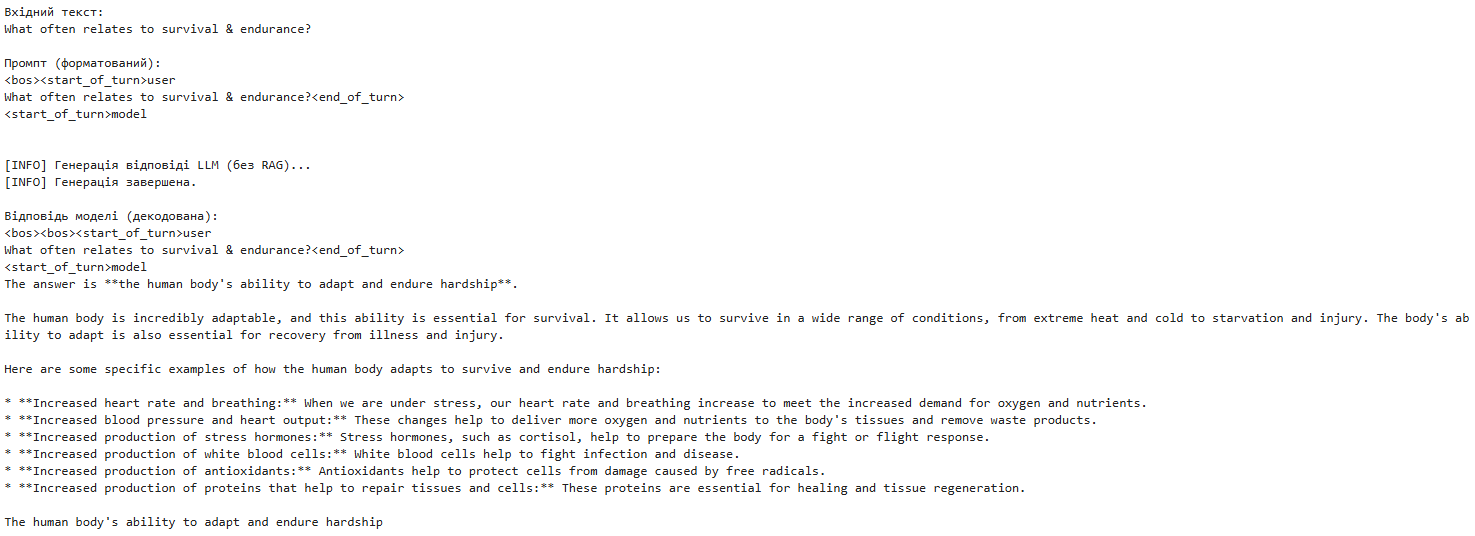
**

*Рис. 11. Перевірка працездатності пошуку по чанках. Оцінка подібності*

**

*Рис. 12. Вибір моделі та конфігурації квантизації на основі пам'яті*

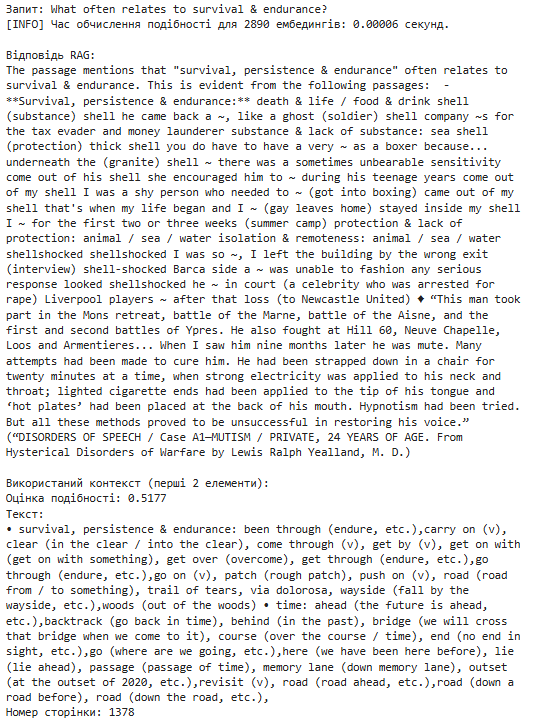
Завантаження моделі google/gemma-2b-it з Hugging Face.

**

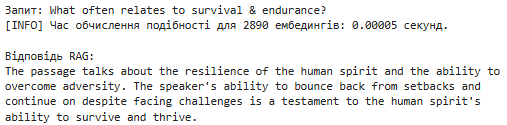
*Рис. 13. Відповідь LLM моделі без RAG*

**

*Рис. 14. Тест форматувальника промпту*



*Рис. 15. Відповідь моделі з використанням RAG (з контекстом)*

**

*Рис. 16. Відповідь моделі з використанням RAG (без контексту)*

**Висновок:** У даній роботі було реалізовано підхід Retrieval-Augmented Generation (RAG), який дозволяє моделі формувати більш точні та контекстно релевантні відповіді шляхом використання зовнішніх даних під час генерації. Для обробки запитів використовувалась модель Gemma 2B, оскільки через обмеження обсягу доступної пам'яті GPU не було можливості використати більші моделі на кшталт Gemma 7B або потужніші. Варто зазначити, що використання графічного процесора значно підвищує ефективність обробки порівняно з CPU, особливо під час роботи із великими мовними моделями. Для забезпечення підтримки GPU локально було проведено трудомістке налаштування середовища: встановлення WSL2 із Ubuntu, інсталяція CUDA для Ubuntu, налаштування відповідної версії TensorFlow та запуск Jupyter Notebook, що вимагало значних витрат часу та зусиль.