Лабораторна робота №2

Оптимізація стратегій чанкінгу

Мета роботи: провести порівняльний аналіз методів розбиття тексту на фрагменти (чанкінг), дослідити вплив параметрів chunk_size та chunk_overlap на якість релевантної видачі та обчислити метрику Context Recall для різних стратегій.

Науково-теоретичне обґрунтування

У системах RAG якість відповіді LLM критично залежить від контексту.

Проблема. Занадто малий блок (chunk) може втратити контекст (втрата смислових зв'язків), а занадто великий - містити надлишковий шум, що знижує точність семантичного пошуку.

Методи розбиття:

- *Fixed-size chunking - механічне розбиття за кількістю символів/токенів.*
- *Fixed-size chunking with overlap - стратегія фіксованого розміру з перекриттям.*
- *Sentence-based chunking - розбиття з урахуванням граматичних меж речень.*
- *Semantic chunking - адаптивне розбиття, що базується на зміні косинусної відстані між послідовними векторами речень (виявлення зміни теми).*

Стек технологій

LangChain / LlamaIndex — інструменти для реалізації стратегій розбиття.

ChromaDB — векторне сховище для індексації фрагментів.

Sentence-Transformers (all-MiniLM-L6-v2) — для генерації ембедингів.

Pandas / NumPy — для розрахунку метрик.

Зміст роботи

Завдання 1. Оберіть довгий опис фільму з датасету netflix_titles.csv. Сформулюйте 5 запитань, відповіді на які точно присутні у тексті.

Методичні рекомендації

Наприклад, довгий текст для аналізу: *У 2045 році світ занурився у цифрову антиутопію. Головний герой, колишній інженер Марк, живе у підземному місті Новий Едем. Його основна мета – знайти втрачений архів даних свого батька, який містить коди доступу до системи життєзабезпечення поверхні. Під час подорожі Марк зустрічає Сару, лідера повстанців "Тіні майбутнього". Вони виявляють, що корпорація 'Aethelgard' використовує ШІ для маніпуляції спогадами громадян. Вирішальна битва відбувається в центрі управління корпорації, де Марк повинен вибрати між особистою помстою та порятунком залишків екосистеми Землі. Ключовий код доступу виявився зашифрованим у ДНК Марка, що стало несподіваним відкриттям для всіх учасників.*

5 контрольних запитань з відповідями:

Питання 1: *Де живе головний герой Марк?* Відповідь: *Новий Едем.*
Питання 2: *Як називається повстанська група Сари?* Відповідь: *Тіні майбутнього.*
Питання 3: *Яка корпорація маніпулює спогадами?* Відповідь: *Aethelgard.*
Питання 4: *Де відбулася вирішальна битва?* Відповідь: *Центр управління корпорації.*
Питання 5: *Де виявився зашифрованим ключ доступу?* Відповідь*: У ДНК Марка.*

Завдання 2. Реалізуйте різні стратегії чанкінгу. Створить 3 окремі векторні бази для подальшого дослідження

1. Стратегія фіксованого розміру без перекриття (fixed_no_ov)

Методичні рекомендації

Найпростіший метод. Він просто відраховує наприклад, 200 символів і ріже, не дивлячись на слова чи розділові знаки. З плюсів – працює швидко, з мінусів – може порізати слово навпіл, що псує ембединг.

Реалізуйте розбиття з параметрами: *chunk_size=150, overlap=0*:

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

fixed_no_ov_docs = RecursiveCharacterTextSplitter(
      chunk_size=150,
      chunk_overlap=0).create_documents([long_text])
```

2. Стратегія фіксованого розміру з перекриттям (fixed_with_ov)

Методичні рекомендації

Як працює код *chunk_size=150, chunk_overlap=50*? Текст ділиться на шматки по 150 символів, але кожен наступний чанк починається на 50 символів раніше, ніж закінчився попередній.

Переваги заключаються у збереженні контексту. Перекриття (overlap) гарантує, що семантичний зв’язок між реченнями не буде втрачено. Якщо запит користувача стосується фрази на межі розриву, вона все одно потрапить у векторний простір цілісно в одному з двох чанків.

Реалізуйте розбиття з параметрами: *chunk_size=150, overlap=50*:

```
fixed_with_ov_docs = RecursiveCharacterTextSplitter(
      chunk_size=150,
      chunk_overlap=50).create_documents([long_text])
```

3. Семантичне розбиття (Semantic Chunking).

Методичні рекомендації

Це найбільш сучасний підхід, який не дивиться на кількість літер. Алгоритм бере перше речення, друге речення, порівнює їхні вектори. Якщо семантична відстань між ними велика (тема змінилася), він робить розріз.

Параметр *breakpoint_threshold_type="percentile"* визначає чутливість до зміни теми.

Використовуйте зміну семантичної близькості для визначення меж.

```
from langchain_experimental.text_splitter import SemanticChunker
from langchain_huggingface import HuggingFaceEmbeddings

semantic_splitter = SemanticChunker(embeddings,
breakpoint_threshold_type="percentile")
semantic_docs = semantic_splitter.create_documents([long_text])
```

або

```
semantic_splitter = SemanticChunker(HuggingFaceEmbeddings())
chunks_semantic = semantic_splitter.split_text(long_text)
```

4. Вимірювання метрики Context Recall. Для кожного запитання знайдіть top-k релевантних фрагментів у *ChromaDB*. Зберіть дані і розрахуйте *Context Recall* наступним чином:

$$CR = \frac{|Знайдені\ фрагменти,\ що\ містять\ відповідь|}{|Загальна\ кількість\ питань\ з\ відповідями\ у\ тексті|}$$

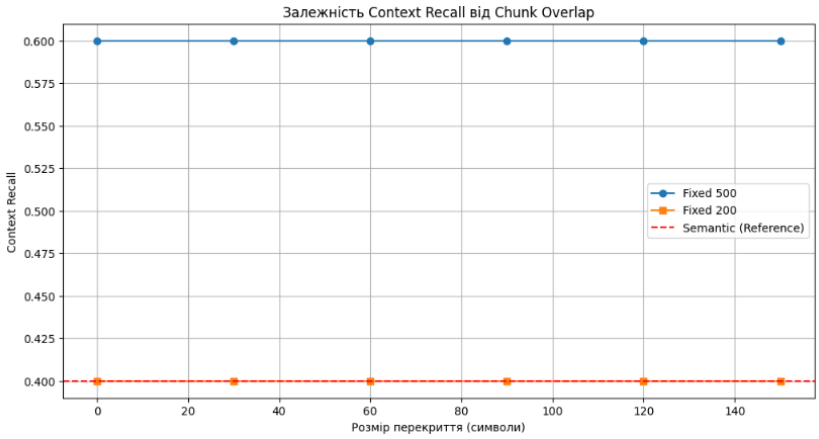
5. Експериментальна частина. Для наведеного прикладу тексту таблиця порівняння стратегій буде мати наступний вигляд:

Strategy	Params	Chunks	Avg Distance	Context Recall
Fixed (No Overlap)	150/0	8	0.5841	0.4
Fixed (With Overlap)	150/50	8	0.5841	0.4
Semantic Chunking 95 percentile		2	1.1000	0.8

Завдання 3.

1. Побудуйте графік залежності Context Recall від розміру chunk_overlap (від 0 до 150 символів).

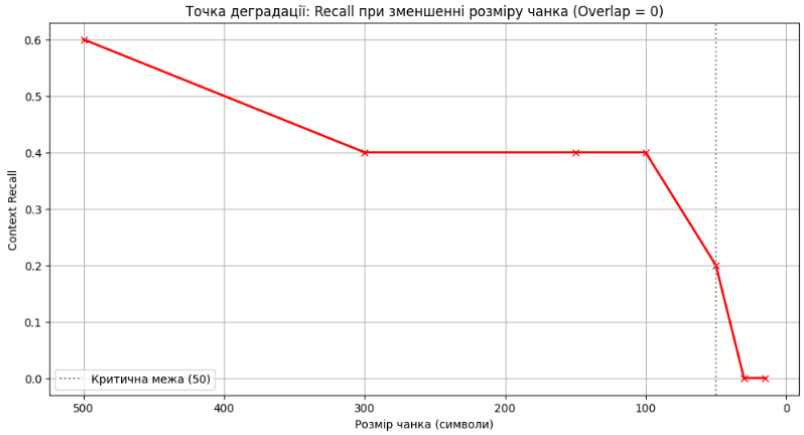
Для наведеного прикладу залежність *Context Recall* від *Chunk Overlap* бути мати наступний вигляд:



2. Визначте «точку деградації» для кожної стратегії (розмір блоку, при якому пошук перестає знаходити правильну відповідь).

Для наведеного прикладу пошук точки деградації буде наступним:

```
Пошук точки деградації (зменшення розміру блоку).
Size: 500 | Recall: 0.60
Size: 300 | Recall: 0.40
Size: 150 | Recall: 0.40
Size: 100 | Recall: 0.40
Size:  50 | Recall: 0.20
Size:  30 | Recall: 0.00
Size:  15 | Recall: 0.00
```



Висновки. При розмірі чанка < 100 символів контекст стає занадто вузьким для розпізнавання сутностей.

Завдання 4. Провести аналіз результатів

Дати відповіді на наступні питання:

– *Інформаційна щільність*. Як семантичне розбиття впливає на «чистоту» контексту порівняно з фіксованим?

– *Проблема втраченої середини (Lost in the Middle)*. Чи допомагає збільшення `overlap` при пошуку сутностей, що знаходяться на межі блоків?

– *Економічна ефективність*. Як вибір стратегії впливає на кількість токенів, що передаються в LLM, і, відповідно, на вартість запиту?

Контрольні запитання

1. Що таке chunking і яку роль він відіграє в архітектурі RAG-систем?
2. Як параметри *chunk_size* та *chunk_overlap* впливають на кількість інформаційного шуму в контексті?
3. Поясніть різницю між *Fixed-size* та *Semantic* розбиттям тексту.
4. Що таке метрика *Context Recall* і чому вона є важливою для оцінки якості RAG?
5. Опишіть алгоритм визначення меж у `SemanticChunker`
6. Яку функцію виконують *embeddings* у процесі семантичного пошуку?
7. Як стратегія *overlap* допомагає вирішити проблему розриву сутностей (*entities*) на межі чанків?
8. Що таке «точка деградації» пошуку і якими факторами вона зумовлена?
9. Чому використання занадто великих чанків може бути економічно неефективним при роботі з *LLM API*?
10. Як модель ембедингів *all-MiniLM-L6-v2* впливає на результати косинусної відстані при семантичному чанкінгу?
11. Як `chunk_overlap` допомагає зберегти цілісність анафоричних посилань (займенників) у тексті?
12. В яких випадках стратегія `Fixed-size` може бути ефективнішою за `Semantic`?