R3_LLM讀書會：Semantic search(語意搜尋)

小組成員：庭羽、Abby、小嫥、Ken

Embedding

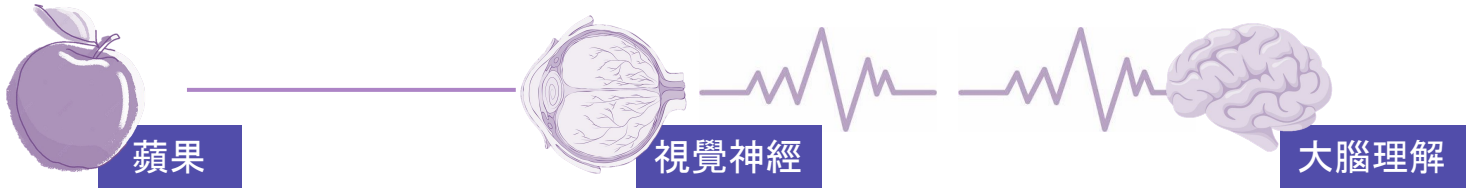
Manifold Hypothesis
流形假設

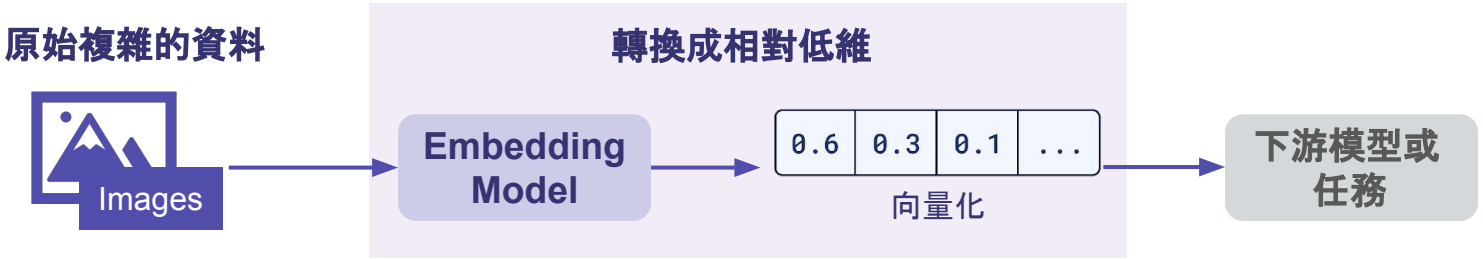
高維空間產生維度冗餘

Embedding
嵌 入

將實體(entity)高維離散的特徵映射到相對低維的連續向量空間

Embedding 是模型理解世界的方式





Embedding 結果作為中間 產物，串接不同的下游模型完成任務

Embedding 主要任務

複雜實體 → 低維向量空間

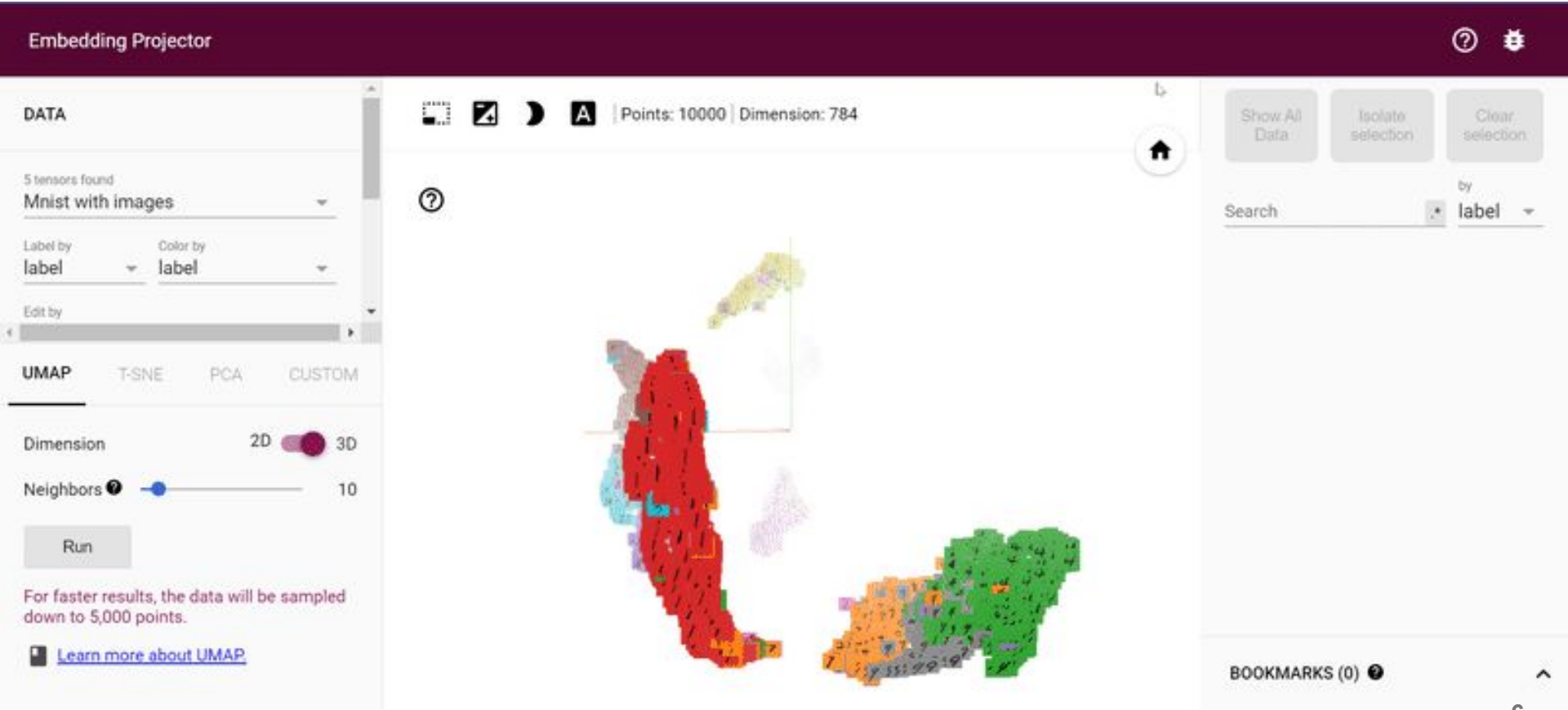
將行為、語言、產品描述等複雜的實體，轉換成能代表它們的一串數字輸入給模型學習

表達實體的特徵

Embedding 可以表達實體間概念的類似或不同，而好的表達能有助於模型學習

相似的實體在空間中接近，相異的實體在空間中遠離

Embedding Projector



Embedding 的崛起

複雜的人類語言
(成千上萬字詞、複雜與義關係)

Input

自然語言處理模型
(NLP, Natural Language Processing)

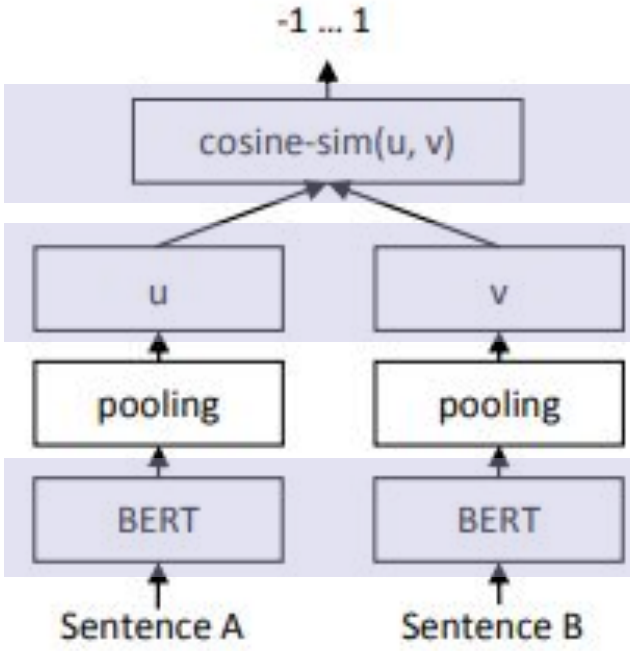
one-hot encoding
稀疏數據不利模型使用

我	[1,0,0,0]
愛	[0,1,0,0]
資料	[0,0,1,0]
科學	[0,0,0,1]

Word2vec 的出現帶動大量
word embedding 模型發展
如：ELMO、GPT、BERT等

語意相似度分析
— Sentence-BERT (SBERT)

利用孿生網路 (siamense networks) 作文本語意相似度的訓練



3. 計算兩者相似度
2. 輸出代表兩個句子的向量
1. 採用BERT模型，並且模型共享參數

文本相似度計算

```python
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-MiniLM-L6-v2')

# Two lists of sentences
sentences1 = ['The cat sits outside',
              'A man is playing guitar',
              'The new movie is awesome']

sentences2 = ['The dog plays in the garden',
              'A woman watches TV',
              'The new movie is so great']

#Compute embedding for both lists
embeddings1 = model.encode(sentences1, convert_to_tensor=True)
embeddings2 = model.encode(sentences2, convert_to_tensor=True)

#Compute cosine-similarities
cosine_scores = util.cos_sim(embeddings1, embeddings2)

#Output the pairs with their score
for i in range(len(sentences1)):
    print("{} \t\t {} \t\t Score: {:.4f}"
```

Model : all-MiniLM-L6-v2

兩個句子

計算兩個句子的向量

相似度計算 (cosine similarity)

Output

```
The cat sits outside        The dog plays in the garden        Score: 0.2838
A man is playing guitar     A woman watches TV                 Score: -0.0327
The new movie is awesome    The new movie is so great          Score: 0.8939
```

文本相似度應用 – 文章主題

文章:
Fed 緊縮擔憂降溫、中國推動刺激政策 美中市場皆獲資金流入

主題分類:
股市

文章:
杭州亞運/中華隊524位選手拚破10金

主題分類:
體育

```
model = SentenceTransformer('distiluse-base-multilingual-cased-v1')
```

```
article = 'Fed 緊縮擔憂降溫、中國推動刺激政策 美中市場皆獲資金流入'
categories = ["汽車","房地產","體育","股市"]
```

```
#Compute embedding for both lists
article_embedding = model.encode(article, convert_to_tensor=True)
for cat in categories:
  category_embedding = model.encode(cat, convert_to_tensor=True)
  cosine_scores = util.cos_sim(article_embedding, category_embedding)
  print("{} Score {:.4f}".format(cat,cosine_scores[0][0]))
```

```
汽車 Score -0.0502
房地產 Score -0.0115
體育 Score -0.0452
股市 Score 0.1220
```

```
article = '杭州亞運／中華隊524位選手 拚破10金'
categories = ["汽車","房地產","體育","股市"]
```

```
#Compute embedding for both lists
article_embedding = model.encode(article, convert_to_tensor=True)
for cat in categories:
  category_embedding = model.encode(cat, convert_to_tensor=True)
  cosine_scores = util.cos_sim(article_embedding, category_embedding)
  print("{} Score {:.4f}".format(cat,cosine_scores[0][0]))
```

```
汽車 Score -0.0214
房地產 Score 0.0108
體育 Score 0.2799
股市 Score 0.0130
```

文本相似度應用 – 意圖理解

測試問句:
您好，拿到新申請的憑證後，接下來要怎麼處理?

模型計算相似度最高結果:
申請新卡後問題

```
import pandas as pd

text = "您好，拿到新申請的憑證後，接下來要怎麼處理?"
intentions = ["申請新卡後問題","已完成開卡","憑證相關問題","無法開啟網頁","出現錯誤訊息"]

df = pd.DataFrame(columns=["intention","score"])
#Compute embedding for both lists
text_embedding = model.encode(text, convert_to_tensor=True)

cosine_scores = []
for intent in intentions:
  category_embedding = model.encode(intent, convert_to_tensor=True)
  score = util.cos_sim(text_embedding, category_embedding)[0][0].item()
  cosine_scores.append(score)

df['intention'] = intentions
df['score'] = cosine_scores

df.sort_values(by="score",ascending=False)
```

	intention	score
0	申請新卡後問題	0.632594
2	憑證相關問題	0.579011
3	無法開啟網頁	0.233029
4	出現錯誤訊息	0.174600
1	已完成開卡	0.159261

文本相似度應用 – 文本檢索

問句:
洋基隊王牌投手是誰？

檢索結果:
MLB/柯爾優質先發收拾守護者 洋基追平戰局保住生機
洋基隊在美聯分區系列賽沒有退路, **王牌投手柯爾** 今天..

cname	pname	
新聞	不滿波爾被棘比布魯克斯 柯瑞批此類討論是鬼扯	灰熊今天（8號）不但以112比142慘敗更賠了夫人又折兵，而膝傷退場的莫蘭
新聞	小朋友睏安手勢不夠還躺地 柯瑞轉發也引詹皇留言	先前WNBA球星模仿自己上季帶起流行的睏安入睡手勢，柯瑞（Stephen C
運動	NBA／柯瑞總冠軍賽G1戰袍拍賣 出價101次後以600萬元售出	勇士球星柯瑞（Stephen Curry）相關收藏品持續以高價售出，他上月出
運動	NBA／柯瑞是否老了？勇士年輕小將穆迪卻這樣說	已經34歲去年帶領勇士奪下冠軍，並拿下MVP的柯瑞（Stephen Curry）
新聞	以柯瑞生涯為故事主題 Apple推原創運動紀錄片	影藝娛樂消息網站《Deadline》報導，Apple將推出以柯瑞（St
運動	MLB／柯爾優質先發收拾守護者 洋基追平戰局保住生機	洋基隊在美聯分區系列賽沒有退路，王牌投手柯爾（Gerrit Cole）今天對決守
要聞	影／民進黨舉行國務青旗艦營 行政院長蘇貞昌出席講課	民進黨上午舉行「國務青旗艦營」，行政院長蘇貞昌上午應邀出席講課，進入會場時面對記
要聞	影／林佳龍落選接行政院長？國民黨團批違反選罷法	國民黨立法院黨團上午召開「林佳龍落選接行政院長？搓圓仔湯違反選罷法？」記者會，
要聞	蘇貞昌明就任三年 民選後最長行政院長「感謝支持」	行政院長蘇貞昌明天就任將滿三年，將成為民選後在任最長行政院長，蘇貞昌今天上午在行

```
input_text = '洋基隊王牌投手是誰?'
input_embed = model.encode([input_text])

cosine_scores = []
for p in content:
  category_embedding = model.encode(p, convert_to_tensor=True)
  score = util.cos_sim(input_embed, category_embedding)[0][0].item()
  cosine_scores.append(score)

df['intention'] = pname
df['score'] = cosine_scores

paragraph = df.sort_values(by="score",ascending=False).head(1).content.values[0]
df.sort_values(by="score",ascending=False).head(1)
```

	cname	pname	
5	運動	MLB／柯爾優質先發收拾守護者 洋基追平戰局保住生機	洋基隊在美聯分區系列賽沒有退路，王牌投手柯爾（Gerrit Cole）今天對決守護

Spaces | mteb / leaderboard | like 3.1k | Running on CPU UPGRADE | App | Files | Community 127

Massive Text Embedding Benchmark (MTEB) Leaderboard. To submit, refer to the MTEB GitHub repository Refer to the MTEB paper for details on metrics, tasks and models.

Search Bar (separate multiple queries with `;`)

Search for a model and press enter...

Model types

- [x] Open
- [x] Proprietary
- [x] Sentence Transformers
- [x] Cross-Encoders
- [x] Bi-Encoders

Model sizes (in number of parameters)

- [x] <100M
- [x] 100M to 250M
- [x] 250M to 500M
- [x] 500M to 1B
- [x] >1B

Overall | Bitext Mining | Classification | Clustering | Pair Classification | Reranking | Retrieval | STS | Summarization | Retrieval w/Instructions

English | Chinese | French | Polish

Overall MTEB Chinese leaderboard (C-MTEB)

- **Metric:** Various, refer to task tabs
- **Languages:** Chinese
- **Credits:** FlagEmbedding

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (35 datasets)	Classification Average (9 datasets)	Clustering Average (4 datasets)
1	gte-Qwen2-7B-instruct	7613	28.36	3584	131072	72.05	75.09	66.06
2	zpoint_large_embedding_zh	326	1.21	1792	512	71.88	74.43	62.23
3	IYun-large-zh					71.04	74.18	66.35

chunking

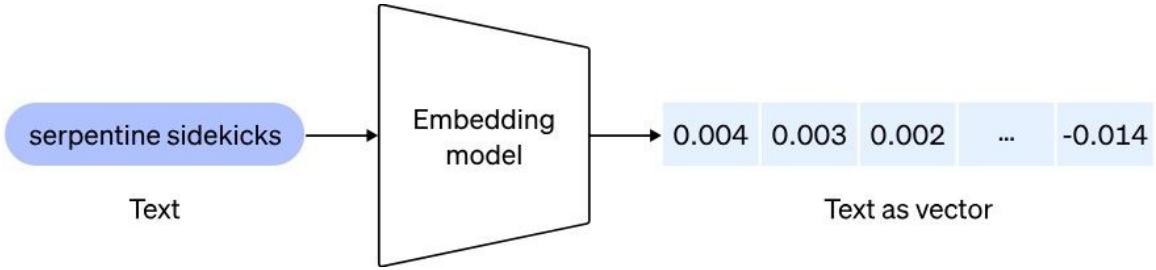
LLM 大語言模型應用的分段（chunking）策略

在構建LLM應用時，我們往往會遇到這樣的問題

....

在使用大型語言模型時，必須將文字或詞彙轉為 **數字向量** 進行處理，

但若輸入的 token 過多，將超出模型的向量長度限制，**影響**數據處理**效率**



如果想要快速且有效地使用大型語言模型，我們需要在 **有限的token數量** 內完成處理。

常用的方法是先將大量資訊 **分成小塊** (chunking)，再放進大模型中，這樣可以更好地從數據庫中提取相關資訊

在討論chunking前,

我們需要先對 chunking的大小有一個認知 …

- 對於 LLM 應用, 往往會有一個 **誤區**, 那就是認為越大的chunking越好, 最好充分利用全部的 token 數量
- 而往 vector store 中索引的時候, **更長的文本** 會產生**更多的噪音** (通常指的是增加無關或雜訊數據)
- 在向量空間中, 進行的是相似性搜索, 過多的噪音可能導致原本相似的向量之間的 **差異看起來更大**

```
sentence1 = embedding.embed_query("吃了個晚飯")
sentence2 = embedding.embed_query("晚上吃了個飯")
np.dot(sentence1, sentence2)
```

```
0.9774936166969695
```

相似度分析：0.97

“吃了個晚飯”

“晚上吃了個飯”

```
sentence1 = embedding.embed_query("忙了一整天餓得精疲力盡，回家快速洗了個澡，然後吃了個晚飯，才恢復了些力氣")
sentence2 = embedding.embed_query("終於到了周末，和家人們晚上吃了個飯，然後在餐桌旁聊了一整晚最近發生的事")
np.dot(sentence1, sentence2)
```

```
0.8573252434443198
```

chunking 的 size 比較大，把前後文一些不相干的資訊也列入比較，相似度會被降低！

相似度分析：0.85

忙了一整天餓得精疲力盡，回家快速洗了個澡，然後 吃了個晚飯，才恢復了些力氣…

終於到了周末，和家人們 晚上吃了個飯，然後在餐桌旁聊了一整晚最近發生的事 ……

其實，分段大小並沒有具體的優劣之分，我們只是需要認識到：

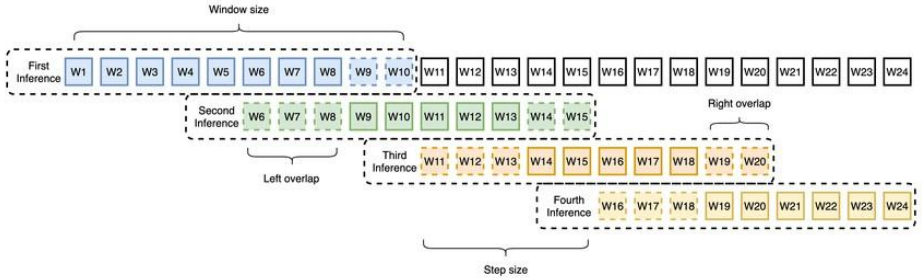
- **過小的chunking**會**忽略**段落或文檔中更廣泛的上下文，但是會**集中**於**具體內容**
- **較大的輸入** 文本大小可能會引入**噪音**或**削弱**單個句子或短語的重要性，使得在查詢索引時更難找到精確的匹配項

我們在選擇 chunking前，我們需要做以下幾點考量 …

- 索引內容的性質：
  - **長篇的文檔資料**
  - **短篇的聊天記錄**
- 所使用到的**embedding模型**在不同chunking大小上
  - sentence-transformers 更適合**短句**
  - text-embedding-ada-002 更適合 256 或 512 token 的**長句**
- 用戶的查詢長度和複雜度：
  - 是**一句話**一句話問
  - 還是一口氣貼**很長的一段文字**

針對文檔資料的分段策略

針對文檔資料的分段策略



我們在選擇chunking前, 我們需要做以下幾點考量 …

1. 固定大小分段

這是**最常見**最直接的一種分段策略, 基本上網上看到的教程都是這個策略

```python
text = "..." # your text
from langchain.text_splitter import CharacterTextSplitter
text_splitter = CharacterTextSplitter(
    separator = "\n\n",
    chunk_size = 256,
    chunk_overlap  = 20
)
docs = text_splitter.create_documents([text])
```

這個分段策略維護上下文關係的思路比較 **簡單粗暴**, 就是在 chunk 和 chunk 之間**保留一定的重疊**

- 優點: 不需要使用任何 NLP 庫或者 LLM, 因此**計算成本低**, 使用簡單
- 缺點: 比較長的段落有可能被掐頭捏尾, 結果 **斷章取義**

針對文檔資料的分段策略 

我們在選擇chunking前，我們需要做以下幾點考量 …

2. 段落分段

這個分段策略是**為了解決**固定大小分段中**斷章取義**的問題。比較粗暴的做法就是，以**句號**和**換行**為分隔符，把整個句子或者段落作為一個 chunk

```
text = "..." # your text
docs = text.split(".")
```

這樣做既**照顧**了**計算成本**，又**避免**了**斷章取義**。

但問題也是顯而易見的，即「.」有時候並不真的是用作句號，可能也是小數點或者省略符。

針對文檔資料的分段策略 

我們在選擇chunking前，我們需要做以下幾點考量 …

3. 語義分段

比段落分段高級一點的做法，是用 **NLTK**(自然語言工具包)這種 Python 庫來識別語義，並根據語義進行分段，**避免**對「.」產生誤解

```
text = "..." # your text
from langchain.text_splitter import NLTKTextSplitter
text_splitter = NLTKTextSplitter()
docs = text_splitter.split_text(text)
```

但是，不管用 NLTK 還是 spaCy，對中文的支持都不是很理想。

如果需要對中文文本進行段落分段，可以考慮 https://github.com/isnowfy/**snownlp** 或者 https://github.com/dongrixinyu/**JioNLP** 這樣的 Python 庫

針對文檔資料的分段策略 

我們在選擇chunking前，我們需要做以下幾點考量 …

4. 遞迴分段

固定大小的分段雖然簡單，但會忽略文本的結構。

遞迴分段方法則會使用一系列 **分隔符** 來逐步將文本分割成小塊。

如果第一次分割不符合要求，它會再次分割，直到達到理想的大小和結構。這樣分割後的塊不會完全相同，但會盡量相近。

Langchain 框架提供了 RecursiveCharacterTextSplitter 類，使用預設的分隔符（“\n\n”, “\n”, “ ”, “”）來分割文本。

```
text = "..." # your text
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    # Set a really small chunk size, just to show.
    chunk_size = 256,
    chunk_overlap  = 20
)

docs = text_splitter.create_documents([text])
```

針對文檔資料的分段策略 

我們在選擇chunking前，我們需要做以下幾點考量 …

5. 格式化分段

格式化分段策略會利用 **文檔本身的格式** 對內容進行分段

例如 Word 格式、Markdown 格式、LaTeX 格式

```python
from langchain.text_splitter import MarkdownTextSplitter
markdown_text = "..."

markdown_splitter = MarkdownTextSplitter(chunk_size=100, chunk_overlap=0)
docs = markdown_splitter.create_documents([markdown_text])
```

針對聊天歷史的分段策略

聊天歷史的分段策略

- 聊天歷史記錄特性:
  - 基於**時間軸**的文本
  - 同時具有 **雙角色**（人-AI）、**多角色**（聊天群）的特徵
- 大型語言模型（LLM）對話流程:
  - 每次互動將所有**先前對話整合** 至**提示詞**（prompt）
  - 獲取回應後續入下一輪
- 分段策略:
  - 聊天記錄的分段與文檔分段 **底層邏輯 相似**
  - 要在LangChain中實現特定功能（例如聊天記錄的分段策略），你需要手動撰寫並部署相關代碼，因為這 **不是LangChain的內建功能**

針對聊天歷史的分段策略

1. 人為主題分段

這是**ChatGPT的做法**，也是最簡單的分段策略，即允許用 戶針對**每個話題開 啟不同的會話**（Session），以方便將**一類話題** 合併到**同一個提示詞**（prompt）

-計算與邏輯：

- 此策略是 基於 產品設計，利用 用戶來區分話題
- 不僅是功能，也是借助 用戶思路的設計策略

-存在的問題：

- 某些用 戶可能不嚴格區分話題。可能導致 多個話題混合在一個會話中

針對聊天歷史的分段策略

2. 時間戳分段

-時間戳分段策略：

- 針對無法人為區分話題的聊天使用
- 以時間為單位，類似固定大小分段策略

-上下文關聯：

- 通過時間戳的重疊來關聯聊天的上下文

-實施需求：

- 聊天不會均勻分布於每個時間點
- 為了保持聊天的連續性，我們需要將 連續的聊天信息分段處理

針對聊天歷史的分段策略

3. 遠近記憶分段

-遠近記憶分段策略 :

- 用於管理 長時間 的聊天歷史, 以符合 LLM的token限制

-實施步驟 :

- 將較早的消息通過 LLM總結, 合併成 簡短的消息
- 保留較新的消息 , 不進行修改

-遞迴分段 :

- 透過 遞迴分段 策略反覆過程, 直到符合 token數量限制

-優點與挑戰 :

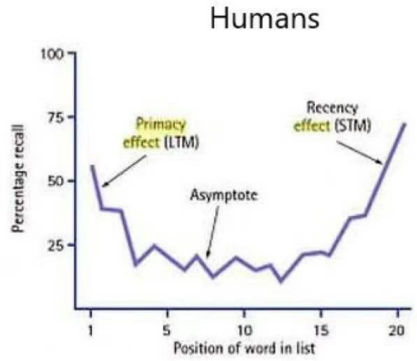
- 策略有效照顧 聊天歷史上下文
- 長聊天 歷史 可能導致早期 細節 內容丟失, 初始信息 的重要性 可能受損

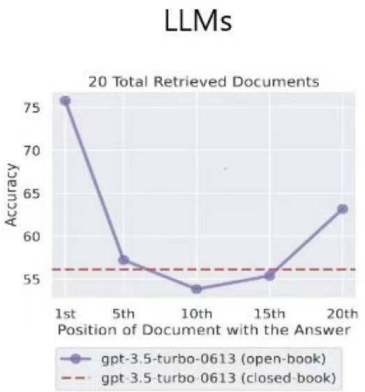
針對聊天歷史的分段策略

4. 微笑曲線分段

為了更加優化遠近記憶分段策略，我們可以考慮模擬人類的記憶：

- **更容易 記得 一件事 最早 和 最近 的內容**
- **更容易 忽略 一件事情 過程中的細節**





同樣地，微笑曲線分段策略要求設計一個梯度，模擬人類的微笑曲線記憶：

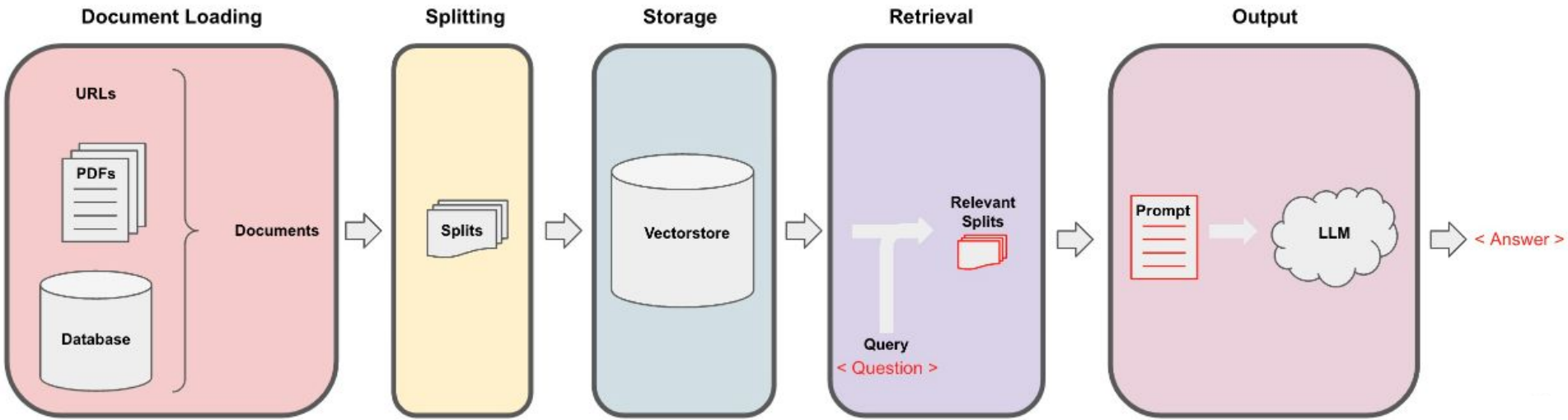
- 把聊天記錄**接近中間的信息** 通過 LLM **做總結**，合併成比較簡短的信息
- 儘量**保留** 聊天記錄**開頭和最新** 的信息。

這個策略的**缺點**是需要 LLM 的大量介入。如果**硬體能力不夠高**的話，整個對話體驗上會有一個**延遲**。隨著算力的不斷提升，這個策略會越來越完美~

Retrieval

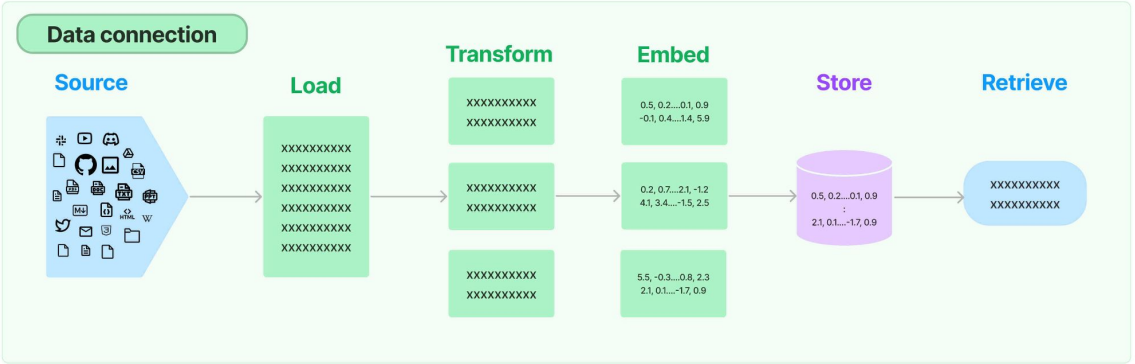
RAG (Retrieval-Augmented Generation)

具有 檢索額外提供的資料，並且產生內容的語言模型應用



Retrieval (檢索)

Data Connection or Retrieval



1.讓它能夠讀取pre-trained LLM model以外的資料
(不在原本模型訓練使用的語料內)

2.將語言模型以外的資料轉換為 embedding

3.將第 2 步所產生的 embedding 存起來

4.告訴語言模型要用什麼方式檢索這些資料，最簡單方式是相似度比較。

向量搜尋 VS 關鍵字搜尋

"what is the capital of Canada?"

dense_retrieval

	_additional	lang	text	title
0	{'distance': -150.8129}	en	The governor general of the province had desig...	Ottawa
1	{'distance': -150.29314}	en	For brief periods, Toronto was twice the capit...	Toronto
2	{'distance': -150.03601}	en	Selection of Ottawa as the capital of Canada p...	Ottawa
3	{'distance': -149.92947}	en	Until the late 18th century Québec was the mos...	Quebec City
4	{'distance': -149.7189}	en	The Quebec Conference on Canadian Confederatio...	Quebec City

keyword_search

	text	title
0	In his 1990 book, "Continental Divide: the Val...	Monarchy of Canada
1	North America outside the zone of Spanish sett...	Early modern period
2	By the Second World War, the Red Ensign was vi...	Flag of Canada

MMR(最大邊際相關性搜尋)

最大邊際相關性優化了所選文件之間查詢的相似性和多樣性。

MMR同時考慮查詢與文件的相關度，以及文件之間的相似度。

相關度確保傳回結果對查詢高度相關，相似度則鼓勵不同語意的文件被包含進結果集。

MMR計算每個候選文件與查詢的相關度，並減去與已選入結果集的文件的最大相似度。這樣更不相似的文檔會有更高的得分。

(提高所選文件的多樣性，查詢到相關但彼此之間內容差異大的文件，創建更豐富、更均衡的搜尋結果)

Vector search libraries

Approximate Nearest-Neighbor Vector search libraries

- Annoy
- FAISS
- ScaNN

- Easy to set up
- Store vectors only

Vector databases

- Store vectors and text
- Easier to update (add new records)
- Allow filtering and more advanced queries

特點	Approximate Nearest-Neighbor	Vector Databases (Vector search libraries)
建立索引速度	相對較快	通常較慢
搜索速度	通常較快	通常較慢
內存使用	通常較低	通常較高
支援多維度向量	是	是
實時更新	有限支援	通常支援
精確性	可配置	高
特徵選擇和數據分割	有限控制	高度可配置
多語言支援	可能需要額外處理	通常支援
擴展性	有限擴展性	通常有較好的擴展性
維護複雜性	較低	較高

Code