

LLM 輔助論文綜整與檢索

組別：第 17 組

組員：李佑軒(112550011)、謝侑哲(112550069)、蔡承志(112550183)

Introduction

動機：為了方便找到適合閱讀的論文，做出可以搜尋論文的使用者介面

以往在找適合的論文閱讀時，我們常常需要一一檢視一份論文的 **abstract**, **introduction**, **conclusion** 等等內容，所需時間成本極高。所幸現在，我們可以透過 **GPT** 來最有效率的幫我們整理資訊，然而就算讓 **GPT-4** 幫忙略讀，因為 **GPT-4** 的資料庫只更新到 2023 年，因此要查詢最新的論文，就需要一份份地下載 **PDF**，再丟給它解析。更何況 **GPT-4** 的免費用量有限，其他常用的 **GPT-3.5**, **Copilot** 都無法解析 **PDF**，**GPT-3.5** 也只擁有到 2021 年的資料庫。

因此，我們打算製作論文整理綜整與檢索的系統，收集最新的(2024)論文並解析其內容，再將解析結果，利用 **RAG LLM** 的方式，以本地資料庫的形式供 **LLM** 搜索，讓 **LLM** 可以檢索出符合使用者需求敘述且足夠新的論文，最後以 **discord Bot** 的方式製作使用者介面，與使用者互動及呈現資料。

Literature Review / Related work

與這個 project 相關的系統有: **Elicit**, **R – discovery** , **Consensus** 等等。

Elicit 透過關鍵字搜尋，會提供前四篇推薦論文的摘要統整，並再將四篇的摘要整合出一份摘要的摘要給使用者參考，以網頁的方式呈現，但目前只支援英文介面。

R-discovery 的部分則是 **mobile App**，一樣透過關鍵字搜尋並推薦相關論文，甚至也會提供與使用者搜尋的關鍵字的有關的關鍵字，與其它系統的不同之處就在於其可在手機上使用的便利性，同樣也只支援英文介面。

至於 **Consensus** 則是透過提問式的搜尋方法，讓使用者能更直覺式的去操作，除了提供推薦相關論文、摘要中的摘要，還會額外提供品質 **tag** 如瀏覽次數最高等等，讓使用者能除了只憑自己的主觀意見外，能有更客觀的標準可以參考，然而其同樣只有英文介面。

我們的系統比較接近於 **Consensus**，在搜尋部分一樣使用提問式搜尋論文，主要的差別在於介面的呈現，以及資料庫的差異。介面呈現中我們讓同時輸出英文與中文的版本，供我們這些中文母語的使用者能更方便搜尋，並且在資料庫上，我們更新到最新的 arXiv 中 AI 類型 2024 年 6 月 8 號的資料，能夠找到最新的論文

Dataset

論文資料網站：<https://arxiv.org/>

論文分類為 computer science 裡面的 Artificial intelligent，資料範圍為從 2024 年一月到 2024 年六月 8 號的所有論文，總共有 13107 篇。

Authors and titles for January 2024

Total of 1922 entries : 1-25 26-50 51-75 76-100 ... 1901-1922
Showing up to 25 entries per page: [fewer](#) | [more](#) | [all](#)

[1] [arXiv:2401.00004](#) [pdf, ps, other]

Informational non-reductionist theory of consciousness that providing maximum accuracy of reality prediction

[E.E. Vityaev](#)

Comments: 14 pages, 7 figures

Subjects: **Artificial Intelligence (cs.AI)**; Neurons and Cognition (q-bio.NC)

[2] [arXiv:2401.00005](#) [pdf, ps, other]

Consciousness as a logically consistent and prognostic model of reality

[Evgenii Vityaev](#)

Comments: 22 pages

Subjects: **Artificial Intelligence (cs.AI)**

原本方案(oldCrawler.py)：將所有的論文 pdf 檔下載下來轉成文字檔使用

1. 先觀察要分析 pdf 之網頁的 HTML 格式，找到所有論文的 pdf 檔案
2. 使用 python 的 wget 將這些 pdf 檔全部下載下來
3. 接下來使用 python 的 PYPDF2 套件，將 PDF 檔裡的文字轉為字串存入.txt 檔裡。

但做完原本方案後，本來預計要使用 hugging face 的 pretrained gpt2 來做全文的總結，因為這樣就不用接 OpenAI 的 api 花錢做總結，不然 1 萬 3000 筆資料我們會不堪負荷。但是遇到的問題是，GPT2 的 input 只能吃 1024 個 token，不足以做全文總結。

因此我們嘗試了 2 個方式: 1. 分割各段並總結 2. 只總結 Abstract。但碰到了不能解決的問題，首先是要分割各段對我們來說並不是一個簡單的工作，我們並沒有想到一個方法可以精準的分割 1 萬 3000 篇論文，並且 GPT2 的效果非常差，總結出來的東西幾乎不能看，不如論文自帶的 Abstract。所以我們就決定直接收集 Abstract 的資料來當作我們的資料庫

最後方案(Crawler.py)：每篇論文只留標題、連結，還有每篇論文的摘要(abstract)

1. 使用 **Chrome driver** 進入論文網站，**implicitly_wait** 是 **driver** 在尋找 **html** 的 **element** 要等待的最長時間，因為進入一個網頁需要時間，如果沒有等待直接尋找會找不到而出錯。
2. 觀察網頁 **html** 發現每篇論文 **a** 標籤的文字都不一樣，是"arXiv:2406.+(自己的編號)"，我們利用 **selenium** 的 **By.PARTIAL_LINK_TEXT** 可以一次把他們全部找出來。
3. 分別進入每個連結，將找到的 **title** 與 **abstract** 連同網址一起寫入.txt 檔裡。
4. 一頁的資料全部抓完後，利用連結的 **XPATH** 切換到論文網站的下一頁，並重複上述操作。

網站上的資料：

Informational non-reductionist theory of consciousness that providing maximum accuracy of reality prediction

[E.E. Vityaev](#)

The paper considers a non-reductionist theory of consciousness, which is not reducible to theories of reality and to physiological or psychological theories. Following D.I.Dubrovsky's "informational approach" to the "Mind-Brain Problem", we consider the reality through the prism of information about observed phenomena, which, in turn, is perceived by subjective reality through sensations, perceptions, feelings, etc., which, in turn, are information about the corresponding brain processes. Within this framework the following principle of the Information Theory of Consciousness (ITS) development is put forward: the brain discovers all possible causal relations in the external world and makes all possible inferences by them. The paper shows that ITS built on this principle: (1) also base on the information laws of the structure of external world; (2) explains the structure and functioning of the brain functional systems and cellular ensembles; (3) ensures maximum accuracy of predictions and the anticipation of reality; (4) resolves emerging contradictions and (5) is an information theory of the brain's reflection of reality.

最後論文.txt 檔的格式如下：

```
Name
Informational non-reductionist theory of consciousness that providing maximum accuracy of reality prediction

Abstract
The paper considers a non-reductionist theory of consciousness, which is not reducible to theories of reality and to physiological or psychological theories. Following D.I.Dubrovsky's "informational approach" to the "Mind-Brain Problem", we consider the reality through the prism of information about observed phenomena, which, in turn, is perceived by subjective reality through sensations, perceptions, feelings, etc., which, in turn, are information about the corresponding brain processes. Within this framework the following principle of the Information Theory of Consciousness (ITS) development is put forward: the brain discovers all possible causal relations in the external world and makes all possible inferences by them. The paper shows that ITS built on this principle: (1) also base on the information laws of the structure of external world; (2) explains the structure and functioning of the brain functional systems and cellular ensembles; (3) ensures maximum accuracy of predictions and the anticipation of reality; (4) resolves emerging contradictions and (5) is an information theory of the brain's reflection of reality.

Link:
https://arxiv.org/abs/2401.00004
```

Baseline

這裡我們採用的 **baseline** 是 **google scholar**，因為這是個在搜尋論文時，最常用到的搜尋引擎之一，並且我們實際試用過後，他對於中文的輸入表現並不是太好，因此我們的目標就是希望可以超過 **google scholar** 的表現，以此來判斷我們的 **model** 有沒有足夠好的表現。

Main approach

1) LLM

在 model 的實作上，我們主要使用 langchain 套件來建造 LLM 與其他功能的串接，形成一個"chain"。首先，建立一個 Agent class，把整個 chain 包成一個物件，裡面主要有三個部分: init, ask_without_question, ask_question。

在我們的論文檢索 RAG LLM 中，主要的流程會是: 1. 建立 RAG 的資料庫供 LLM 參考、2. 建立 LLM、3. 將使用者的輸入做 prompt engineering、4. 使用 LLM 做運算、例外處理並輸出

1. 建立資料庫

第一部分是建立資料庫供 LLM 在運算時參考，這塊寫在 init 的部分。我們利用了 langchain 的 documentloader，把我們在 dataset section 提到的，利用爬蟲蒐集得來的 1 萬 3 千筆資料，以 document object 的形式下載下來，作為資料庫的資料。接著因為 Transformer based 的 LLM 是有輸入的限制的，因此我們使用了 langchain 的 TextSplitter 把文件拆分成每 100 個 token 為一組，才能正確地應用在 model 上。拆分完文字之後，model 並不能直接讀取文字，因此要使用 model 對應的 embedding model 來把 token 轉成向量，而這裡我們主要採用 GPT3.5-turbo 的模型，所以使用的是 OpenAI 的 Embedding model。得到向量之後，使用 langchain 的 vector database 來儲存向量，我們使用的是 Chroma 資料庫，因為搜索的效率很高，很適合用來做大規模資料的搜索。至此就建立好了 RAG 的資料庫。

2. 建立 LLM

第二部分是建立 LLM，在本專案中，我們建立了兩種 LLM，一個是有連結到資料庫的，一個是沒有連結到資料庫的。有連結到資料庫的是使用 langchain 提供的 RetrievalQA chain，是一個包含了用來運算的 LLM model，用來檢索的 retriever，LLM model 採用的是 GPT3.5-turbo，retriever 則是使用上文提到的 Chroma 資料庫本身的搜索功能，並搭配 map_reduce 的搜索方法，一個一個文件去找出相關的資料，然後再彙整在一起。沒有連結到資料庫的 model 則是直接使用 openai api 的 chatcompletion 的 GPT3.5-turbo 的 chatmodel 來做，就只有 GPT3.5 被訓練時的資料，沒有我們新加的資料。

在第二部分中，會做兩個 model 的原因是，RAG model 會傾向於找資料庫中的資料，如果資料庫中沒有，可能也不會找他一開始被訓練時的資料，而是會直接輸出"不知道"之類的，所以在這種情況發生時，就必須用沒

有連接到資料庫的 LLM 來進行輸出，以避免輸出"不知道"。至於選用 GPT3.5-turbo 的原因是，GPT4 的價格太貴，一下就會把我們儲值的用量用光，所以使用比較便宜但效果也還有一定水準的 GPT3.5-turbo。

3. Prompt Engineering

第三部分是將使用者的輸入做 prompt engineering，我們這裡使用的方法主要有 5 個：

1. 角色扮演
2. question optimization
3. few-shot prompting
4. 中英混合
5. 符號強調

第一個角色扮演的方法，我們在 prompt 中，下了指令，讓 LLM 扮演"professional computer science scientist"，讓 model 的回答能夠更貼近預期(搜尋 computer science 的 paper)。

第二個 question optimization 的方法中，因為我們無法預期使用者的輸入，所以我們直接先預寫好該問的問題，請 LLM 搜尋相關的論文，然後讓他另外參照使用者的輸入去搜尋，用來讓 LLM 能明確的理解問題，而不會被使用者的意外輸入混淆。

第三個 few-shot prompting 的方法中，我們給予 LLM 輸出的範例，讓他知道該給出什麼樣的結果，也讓輸出的格式能夠比較好的符合我們的設計，並在多次嘗試後，得出足夠精簡的範例，避免範例過長影響了 LLM 的表現。

第四個中英混合的方法中，因為我們預設使用者會使用中文來問問題，因此 LLM 用英文的 prompt engineering 的表現會相對比較差，在實驗的過程中，我們發現在 prompt engineering 中，把一些重點的字詞、句子同時寫中英文兩種方式，可以讓 model 的表現更加穩定。

第五個符號強調的方法中，因為 RAG LLM 的輸入比較長，所以可能導致部分的輸入被 LLM 忽略，後來我們發現在輸入中利用'---'，'***'等符號來進行強調，可以讓 LLM 更重視某部分的輸入，讓輸出的結果更加穩定而不會忽略掉某些重要的條件。

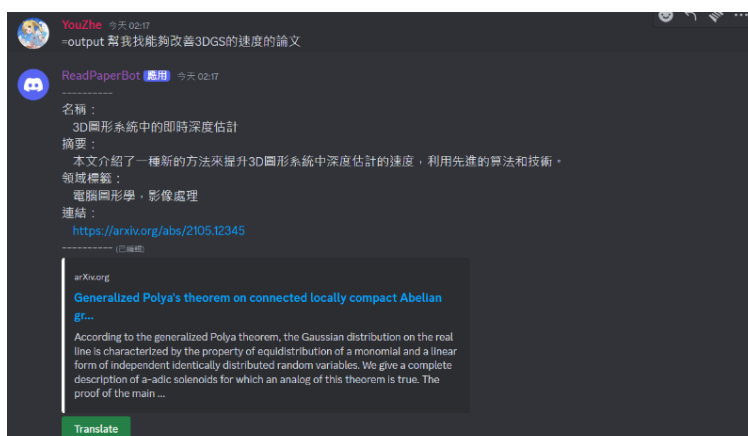
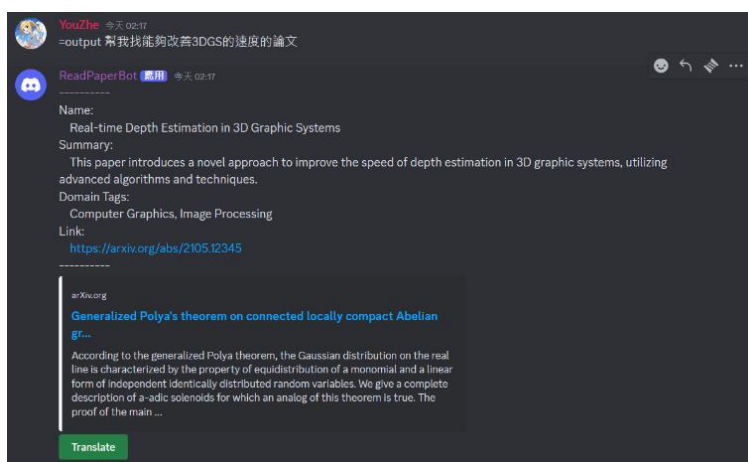
4. 運算、例外處理、輸出

第四部份是使用 LLM 運算、例外處理並輸出，有了上述的 LLM、資料庫、處理過的 prompt，接下來只要把 prompt 餵給 LLM 就可以得到輸出的結果，不過 RAG model 的輸出結果有時候會因為資料庫找不到相符的資料(畢竟只有 6 個月的資料)，而給出不知道的回應，因此我們另外做了例外處理，當 RAG LLM 給出“I don’t”, “I can’t”之類的片段時，通常代表他找不到相應的資料，這時候我們就改用沒有連結到資料庫的 LLM，這時候他就會更傾向於用 LLM 本來被訓練時的資料來回答(2021 年之前的)，雖然不是最新的 2024 年 paper，但 LLM 本身的訓練資料夠大足夠他給出有一定水準的結果。至此就完成了核心 model 的演算法、輸入輸出處理。

2) Discord Bot

接下來講解如何用 Discord Bot 與使用者作互動並取得使用者輸入，輸出相應的結果。輸入指令=search，加一空格後便直接輸入想問的問題(ex: =search 幫我找跟 CNN 相關的最新論文)。略為等待(約 10~20 秒)便可得到輸出結果和一翻譯按鈕，一開始輸出結果為英文，按下翻譯按鈕後可在中英文之間切換。(須注意，翻譯按鈕預設會在 3 分鐘後失效)

(下面圖表是擷取測試時結果，指令名稱尚未從=output 變更為 =search)



Discord Bot 撰寫的部分，使用 `@command` 撰寫指令名稱並獲得使用者輸入 `msg`，引入上述的 `LLMAgent()`，問題丟給 `LLMAgent.ask_question(msg)`後取得搜尋結果並輸出。

```
@commands.command()
async def search(self, ctx, *, msg):
```

Button 則是創一個 `class`，設計完外觀後，監聽 `button` 是否被點擊，當被點擊時，判斷當前 `content` 為中文還是英文，`edit` 至相反 `content`。
(最初的 Bot Setup 跟基本架構參考 [Reference-Discord Bot Setup](#))

Evaluation Metric

我們的 `evaluation` 方式主要有 3 個，格式正確率、內容正確率、內容相關率。格式正確率的定義為：輸出的格式正確無誤，內容正確率的定義為：輸出的內容正確無誤，內容相關率的定義為：輸出的結果與問題相關。評估的方式是，對於 `model` 提問不同多元的問題，包括了中文或英文、不同領域知識、敘事型或關鍵字型。最後由我們人工判讀資料是否正確，並記錄結果分析。

Result & Analysis


對於我們的 `model`，做了 30 次的測試之後，格式正確率為 93.3%(28/30)，內容正確率為 13.3%(4/30)，內容相關率為 96.6%(29/30)，可以看到格式正確率與內容正確率的表現都非常良好，唯獨內容正確率特別的低。


對於我們的 `baseline google scholar`，因為沒有格式問題、正確率問題，因此我們只評估內容的相關率，在做了 30 次測試之後，`google scholar` 的內容相關率只有 6.6%(2/30)，明顯低於我們的模型，因此可以看出我們的模型在我們設計的多元測試中表現更好。

另外，針對我們表現率特別低的內容正確率作分析，在分析之後發現，基本上在名稱、總結、`tag` 的部分都不會出錯，出錯的部分都是 `link` 與給出的論文不符，而是給出隨機的 `link`。所以對於我們未來的改善方式有一大部分要解決這個問題(解決方案寫於下文)。

Error Analysis

在 `model` 建好之後，我們做了幾個實驗，測試對於不同輸入的效果：

 **YouZhe** 今天 01:31
=output I want to find paper about route recommend

 **ReadPaperBot** 應用 今天 01:31

名稱：
學習智能運輸系統的推薦路線

摘要：
我建議這篇名為"Learning to Recommend Routes for Intelligent Transportation Systems"的論文來回答您關於路線推薦的問題。該論文提出了一種機器學習方法，通過利用歷史交通數據和用戶偏好來推薦智能運輸系統的最佳路線。您可以通過此鏈接訪問論文: <https://arxiv.org/abs/2406.00199>

領域標籤：
智能運輸、機器學習


連結：
<https://arxiv.org/abs/2406.00199> (已編輯)


arXiv.org

Exfiltration of personal information from ChatGPT via prompt injection

We report that ChatGPT 4 and 4o are susceptible to a prompt injection attack that allows an attacker to exfiltrate users' personal data. It is applicable without the use of any 3rd party tools and all users are currently affected. This vulnerability is exacerbated by the recent introduction of ChatGPT's memory feature, which allows an attacker t...

[Translate](#)

 **Qoo鸞鸞** 昨天 20:06
=output 幫我找一篇跟路徑相關的論文

 **ReadPaperBot** 應用 昨天 20:06

我推薦以下的論文:

名稱:
路線推薦的調查: 方法、應用和機會

摘要:
這篇論文提供了關於路線推薦的各種方法、應用和機會的調查。

領域標籤:
路線推薦

連結:
<https://arxiv.org/abs/2403.00284>

----- (已編輯)

arXiv.org

A Survey of Route Recommendations: Methods, Applications, and Oppor...

Nowadays, with advanced information technologies deployed citywide, large data volumes and powerful computational resources are intelligentizing modern city development. As an important part of intelligent transportation, route recommendation and its applications are widely used, directly influencing citizens' travel habits. Developing smart and...

[Translate](#)

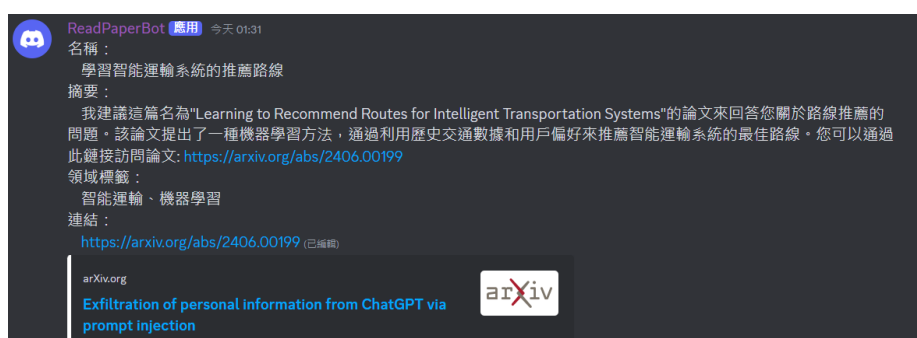
可以看見中英文輸入的都能找到相關的 paper，達成了我們所需求的效果，並且這個表現比起我們的 baseline: google scholar 對比(如下圖)，明顯精準了許多，可以說達成了不錯的效果。



不過在多次的實驗中，我們也發現了不少的問題：

1. 輸出的結果中，給出的資料大多是正確的，但是在 **Link** 的部分，卻常常給出跟資料不符合的 **Link**

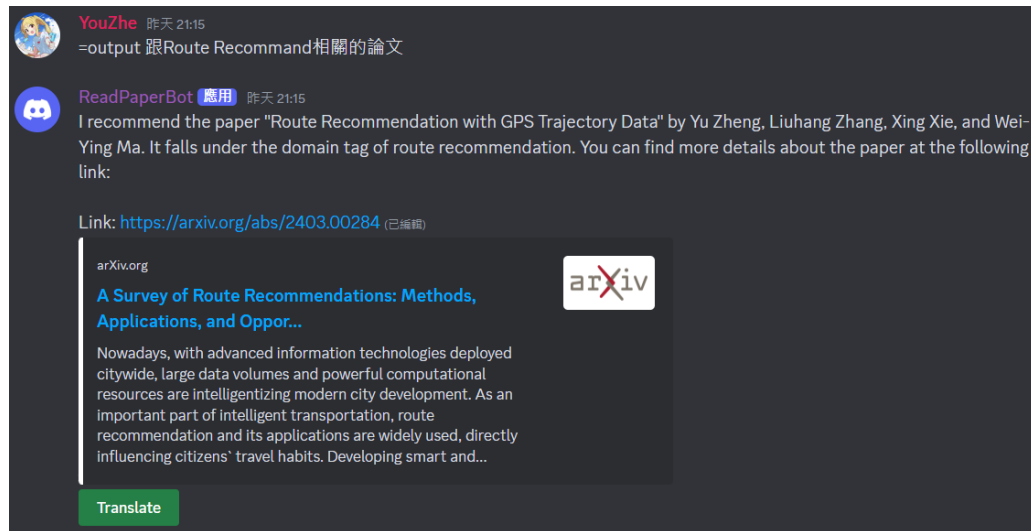
如圖，他推薦了正確的跟路徑推薦相關的論文，在名稱、描述、tag 的部分都是對的，但是 **Link** 的部分卻是錯誤的，給出了一個完全無關的 **GPT attack** 相關的論文。



對於此情況，我們的推測是，在做 **RAG** 的 **retrieve** 時，因為連結本身的意義是無法被解讀的，所以可能會因此搜尋到錯誤的連結。針對此的解決方式，我們認為可以在收集資料時，做另外的處理，讓 **retrieve** 時，可以讓連結的意義能夠被辨識出來，或是想辦法壓縮資料，並增加搜索的 **chunk** 大小，讓 **link** 能跟其他更有意義的資料被一起搜尋到

2. 輸出的結果中，有時候會有不符合格式的狀況

如圖，可以明顯看到他有時候的輸出不會按照我們的格式來進行，而變成完全敘述型的狀況



對於此情況，我們的推測是，因為有比較多的 **prompt engineering**，使得輸入變得比較冗長，而導致格式的要求即使在我們使用符號強調之後，仍可能會被忽略掉。對此的解法，我們認為可以透過引入更多不同的 **prompt engineering** 方法，並進行實驗，找出最好的 **prompt engineering** 方法，來讓他能符合格式的機率最高。

3. 運行時間較久

相較於網頁版的 **chatgpt**，差不多的內容量下，我們的 **bot** 在運行的時長上會比較久，通常會落在 10~20 秒之間，網頁版的 **chatgpt** 卻可以在 10 秒內就完成輸出。

```
time: 9.306124925613403
judge
translate...
ask_without_db start
time: 3.579869270324707
ask_without_db end
time: 3.5808746814727783
```

對於此情況，我們認為是在串接 API 時，會比 GPT 直接輸出的時間慢，並且我們還有比較多額外的翻譯跟其他處理，要解決這問題，我們可以試著採用不同的 GPT 模型，或者找有沒有更快速的方案，並且把一些跟核心內容比較無關的內容精簡掉。

Future Work

對於我們未來的改善方向，首先要改善的是給出不符合的 link 的問題，這算是滿嚴重的影響了結果的成效，因此未來有時間得首先改善這個問題，也可以把 Error Analysis 中提到的不同問題作修復。

做完基本的修復之後，可以做其他的新功能，像是我們可以尋找方式，把原本想做的功能，也就是把論文的 pdf 檔做完整解析與總結的部分做完。也可以自動擴充資料庫，讓她不只有到 2024 年 6 月 8 號的資料，因為 bot 可以長期運行，可以設計讓他每天定時爬取資料，自動擴充資料庫，讓我們的資料能夠保持最新。另外，也可以把介面不只做 Discord Bot 的介面，也可以做成網頁的形式，視覺化更多的資料，讓得到的結果更易於閱讀。

Code

<https://github.com/NYCU-AI-intro/Intro-AI-Final-Project>

Contribution of each member

李佑軒：撰寫網路爬蟲，從 arxiv 網站上透過 selenium 爬取每篇論文標題、連結和摘要，整理到個別的文字檔裡供建立資料庫使用。

謝侑哲：建立 RAG 的資料庫和 LLM，將使用者的輸入做 prompt engineering 後使用 LLM 做運算，檢查並處理輸出結果。把 LLM 跟 Discord Bot 做串接。

蔡承志：撰寫 Discord Bot，製作 UI 提供使用者輸入頁面及搜尋結果呈現。

References

Prompt Engineering:

<https://medium.com/@cch.chichieh/llm-%E5%90%84%E7%A8%AE%E6%8A%80%E5%B7%A7-prompt-engineering-%E6%8C%87%E5%8D%97-6ac4201a4cbe>

Langchain Document:

<https://python.langchain.com/v0.2/docs/introduction/>

<https://python.langchain.com.cn/docs/>

OpenAI API:

<https://platform.openai.com/docs/introduction>

Discord Bot Setup:

https://hackmd.io/@smallshawn95/python_discord_bot_base

<https://www.youtube.com/watch?v=odIQEJW0m1M&list=PLSCgthA1Anif1w6mKM3O6xIBGGypXtrtN>

Discord Bot Button:

https://hackmd.io/@smallshawn95/python_discord_bot_button?fbclid=IwAR063XEP8iiA-EWLpU4jJF-CJ2l66ZWUISJcccXaKy6RTgbuaa4RGfSk8t0

<https://n.sfs.tw/content/index/16017>

Selenium Document :

<https://www.selenium.dev/documentation/webdriver/>