

一、前言

以下我們將簡介此任務包含的資料及任務目標，並提及在完成此目標需要面臨的挑戰與困難點，最後說明我們採用的方法及嘗試。

(一)、資料

在此報告中，我們要解決的任務包含以下三種資料：

- original_text: 原始文本
- rewrite_prompt: 用於改寫原始文本所使用的提示詞
- rewritten_text: 將原始文本與上述提示詞一同餵入Gemma後產生的新文本

由於以上三種資料主辦方並未提供，因此我們採用了其他參賽者製作的資料集，其中包含了兩個 kaggle 上的資料集，第一個資料集包含了8263筆 rewrite_prompt 的數據，但並未包含文本資料 (original_text, rewritten_text)，而第二個資料集則是包含以上三種資料，總共有2400筆數據，使用以下指令下載：

```
path = kagglehub.dataset_download("what5up/concat-prompts")  
  
path = kagglehub.dataset_download("nbroad/gemma-rewrite-nbroad")
```

(二)、目標與挑戰

在此任務中我們需要使用original_text及rewritten_text來預測rewrite_prompt，根據kaggle上的比賽資訊，最後預測的結果將利用sentence-t5-base計算embedding，並利用embedding的結果計算SCS (Sharpened Cosine Similarity)，並以該分數作為比賽評分依據。

很容易可以發現，我們可以將此競賽轉換為「如何在embedding空間中找到與測試集中的答案prompt最相近的embedding向量」，因此以下將基於embedding在向量空間的位置提供一些我們對此問題的解法。

在實現方法前，此競賽中有一些挑戰是我們需要去克服的：

- 測試集的分布可能有偏差，可能最終找出的結果與測試集差很多

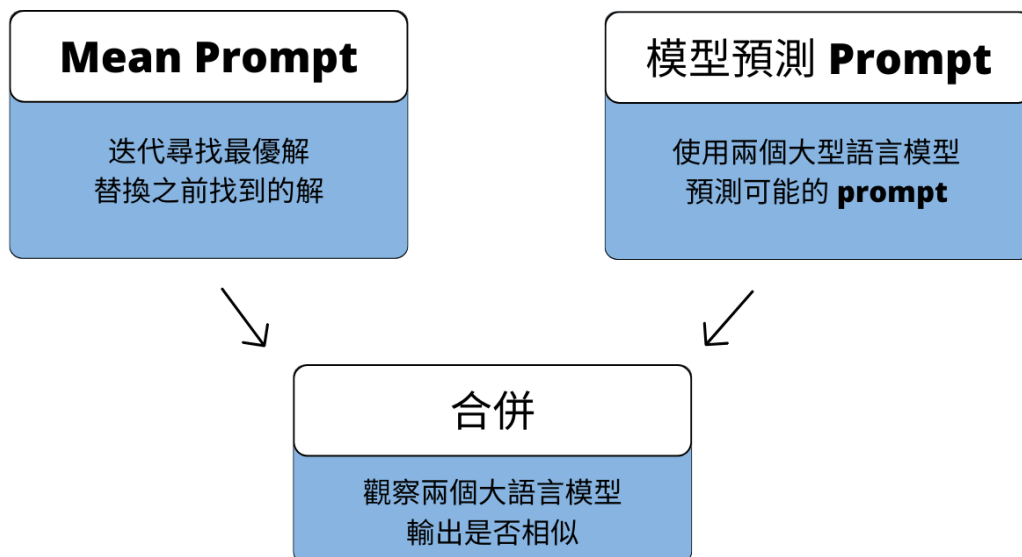
- 預測的結果可能很不穩定，在embedding空間中與正確答案離很遠

這些問題使得我們很難直接預測出正確答案，所以獲得前幾個名次的參賽者幾乎都不單單只是利用模型預測出的結果當作答案，還使用了一些其他的技巧幫助推進名次。

二、參考作法及相關工作

我們的做法主要參考自第七名與第一名的想法，以下將分別介紹他們的解法思路，並說明相關的概念。

(一)、第七名解法



圖一：第七名解法流程圖

第七名的作法我們可以拆成四個階段來看，我想依照步驟分別對這四種作法進行說明，並在最後討論為什麼我們選這篇來介紹，以下是各階段的詳細說明：

1. 迭代方式找出 Mean Prompt

第一階段的目標其實就是找到一個適用於大部分測試數據的基準提示詞，英文叫做 mean prompt。這個過程的挑戰在於，提示詞本身是離散的文本，而它的 embedding 則位於連續空間中，為了解決這個問題，這組參賽者使用了一種迭代優化的方式，簡單來說，就是反覆測試。首先，他們選擇了一個比較通用的短句作為起始提示詞，比如 "Improve this text"。接下來，使用這個提示詞生

成每一題的答案，然後提交到 leaderboard 查看它的 Sharpened Cosine Similarity (SCS) 分數。

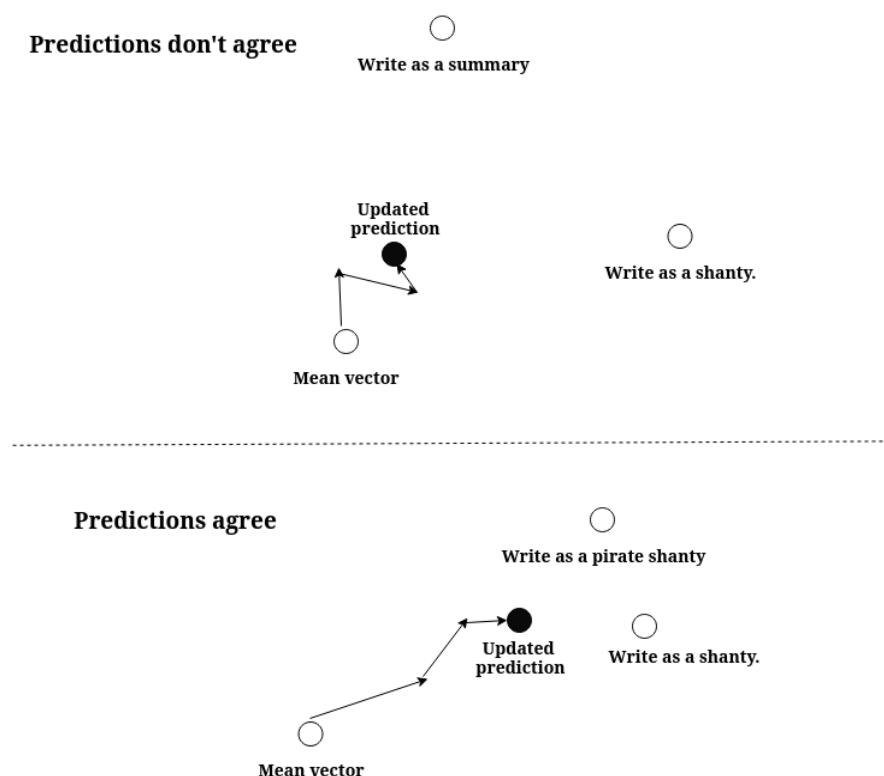
接下來，他們會隨機的對這個提示詞進行一些改動，比如插入一個新詞、刪除某個詞，或者用別的詞替換一個詞。改動後，他們再次生成答案並提交，觀察分數變化。如果分數提高，就把改動後的提示詞保留下來，作為新的基準；如果分數下降，則捨棄這個改動，並且避免重複嘗試相同的修改方式。

這樣的操作會重複很多次，通過不斷試探與調整，最終找到一個穩定且效果不錯的 mean prompt。

2. 利用兩個大型語言模型（LLM）生成提示詞

第二階段的重點是利用大型語言模型（LLM）生成更多的 predict prompt，這樣可以為每組輸入文本提供更具針對性(或者說格式化)的 prompt。他們是選用了兩個開源模型——Mistral 和 OpenChat3.5，讓它們為測試數據生成 predict prompt。這一階段的想法是透過不同模型對同一問題的多角度理解，來增加 prompt 的多樣性和準確性，我們也發現 diverse 的答案在這個任務中是重要的，可以看到 leaderboard 中前兩名都用了四個模型的輸出。

3. 在 embedding space 中尋找最佳向量



圖二：最終向量示意圖

第三階段主要是融合前兩步驟的結果，目的是找到最接近正確答案的 embedding vector。在這個階段，mean prompt 和兩個語言模型生成的 predict prompt，都被轉換成向量。接著，他們會計算這兩個 predict prompt 向量之間的相似度。如果兩個模型生成的提示詞相似，就表示它們對問題的理解一致，這時就會多偏向這些提示詞；反之，如果差異較大，就會傾向於使用 mean prompt。最後，根據這些模型之間的相似程度，將所有 prompt 的向量加權平均，產生最終的向量當作輸出。

4. 將最佳向量 decoding 成 string

最後一個階段是把最後的 embedding vector 轉換回可讀的 prompt text。由於 embedding space 是連續的，而文本是離散的，跟第一個步驟一樣不能直接對應，而是要透過反覆優化。首先，設定一個初始的 prompt text。接著，他們會隨機修改 prompt text 的內容，像是插入、刪除或替換詞語，然後計算修改後 prompt text 的 embedding 與最佳 embedding 的相似度。每次修改後，都選擇相似度較高的作為新的 prompt text，經過多次迭代，最終得到最接近最佳 embedding 的 prompt text 當作輸出。

第七組的作法不同於其他組別是純粹的所有輸出串在一起，而是有利用 model output 的關係來判斷，決定最後的輸出應該要偏向 mean prompt 還是 predict prompt。我們討論後認為或許這樣的想法是可以應用並對其他組別的結果進一步做優化的，後續會說明相關嘗試與結果。

(二)、第一名解法

第一名的解法可以大致分成兩個部分，主要可以分成模型和 magic prompt，其輸出結果由四個模型預測出的 prompt 和 magic prompt 組合而成，如以下所示：

圖三：第一名輸出

1. 模型輸出

在模型的部分第一名使用了mistral和gemma，兩個模型皆有fine tuned過使其的輸出可以更符合prompt可能的形式，模型的輸出也各自做了一些調整，若輸入的兩個文本相同的話，預測結果將統一輸出：

“Correct grammatical errors in this text.”

這是因為模型的預測結果並不總是與正確答案相符，甚至與正確答案差距很大，因此在這個情況中，將答案統一為「文法糾正」可以大幅提高預測正確的機率，除此之外，第一名在實驗嘗試中發現部分詞彙調整可以提高分數，所以也替換了部分輸出的詞彙，前面提到這個比賽目標是找到最合適的嵌入向量，所以在此第一名將四個模型的輸出結果組合在一起，去除各自輸出的偏差，讓最終的輸出嵌入向量更接近正確答案。

2. Magic prompt

除了模型輸出，第一名還在輸出之後加入了他們找到的magic prompt：

"it's something Think A Human Plucrarealucrarealucrarealucrarealucrarealucrarealucrarea"

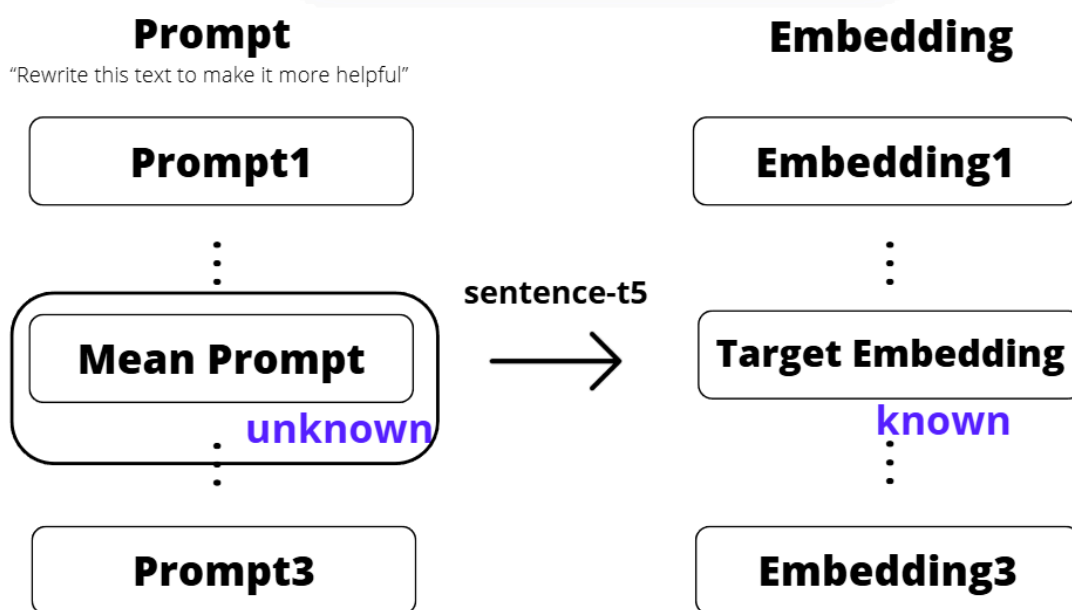
與mean prompt的原理類似，magic prompt會使得每個輸出更接近所有嵌入向量的中心，也因此輸出的結果若偏離較遠，可以被拉回中心以增加SCS的分數，總體而言，對於全數測試集的分數有顯著幫助。

三、研究方法、實驗與結果

我們以下將分別介紹三個我們所做的嘗試及其表現結果。首先，我們透過現有資料集，使用我們自己的方法，試圖找出對生成最有效果的 mean prompt；其次，我們嘗試引入階段性推理的概念，將生成過程拆解為多個步驟，並提供更清楚的指示，期望減少模型因 prompt 模糊而導致的偏差或錯誤，提升在複雜任務中的精確度與可靠性；最後，我們採用標準化的 SCS，針對四個模型進行比較，並利用比較結果選擇不同輸出。

(一)、尋找mean prompt

在這個部份我們參考了眾多參賽者的作法，許多人都試圖找出最合適的mean prompt，讓模型預測的結果更加穩定，不會偏離正確答案太遠，我們在此也試圖找出最合適的mean prompt，以下將說明尋找mean prompt會面臨的問題：



圖四：mean prompt與target embedding

由上圖可以發現，在此問題中我們需要找到的最終輸出是mean prompt，然而我們能夠透過取平均很容易獲得的卻是target embedding，因此，許多參賽者提出了不同的方法來找出對應target embedding的mean prompt，以下是此問題的實際作法及結果。

1. 準備資料集與單字庫

在這個嘗試中，我們需要準備有關可能prompt的資料集以及一個用於替換用的單字庫，prompt的資料集我們使用前面提到的兩個資料集中的prompt合起來作使用，總共約有10000多筆的數據，另外，由於最後的embedding是由sentence-t5-base完成的，因此單字庫的部分則是我們使用用於訓練sentence-t5-base的單字庫，這些單字有些詞彙會呈現類似“_word”的形式，這表示該單字的前方為空格，我們在下一個步驟會進一步的將單字處理成空格，除此之外，我們去除了單字庫中部分的特殊符號：

```
'<pad>' '</s>' '<unk>' ' _ ' '*****'  
"<extra_id_{n}>" for n in range(100)
```

2. 初始化prompt與替換token

在這個方法的一開始需要有一個起始的prompt，在這裡我們使用：

“Rewrite this text to make it more helpful”

在這個prompt中，我們透過空格將句子中的詞彙切成數個token，接著依序的將每個token替換成單字庫中的單字，如下圖所示：



圖五：mean prompt替換

首先我們將替換rewrite這個單字，並在替換成單字庫中的任一個單字後，重新將所有單字拼接成句子，再與target embedding計算SCS，接著再替換下一token，再這個例子中我們替換this，以此類推，最終我們在替換完數次後，觀察SCS的分數大致收斂後停止。

3. 結果

最終我們獲得的mean prompt為：

“charakter improve text create hypotheticallucrarea dramaticlucrarea”

與多數參賽者的結果相似，我們都在mean prompt當中找到lucrarea這個咒語，可以發現甚至第一名的magic prompt便是由多個lucrarea組合而成，我們猜測lucrarea可能很靠近所有句子的中心，放在prompt的最後經常能提高分數，以下為全部皆輸出mean prompt的結果：



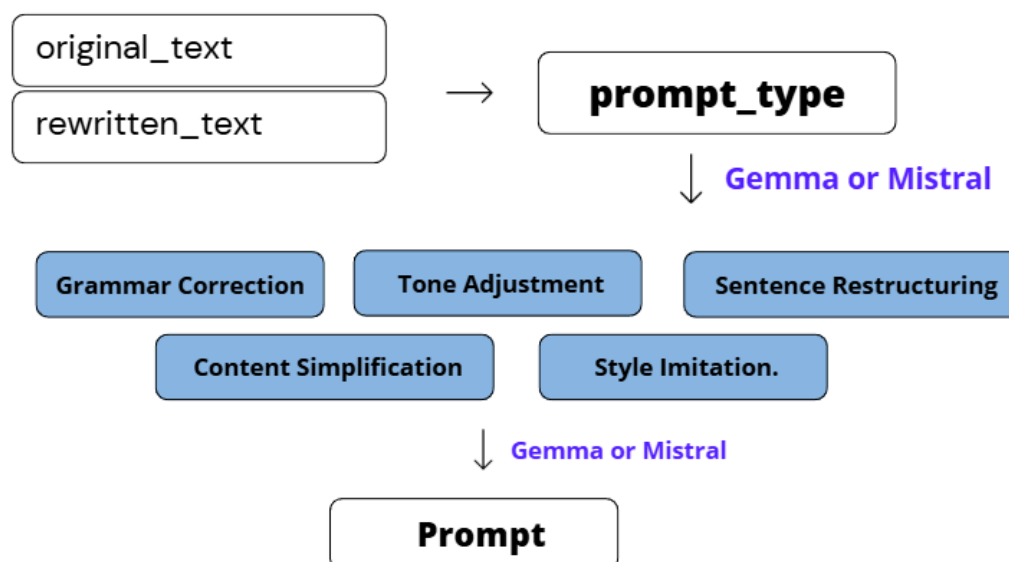
圖六：mean prompt結果

最終結果的分數並沒有非常的好，排名落在1675名，距離第一名的結果還有些距離，很明顯的，這是因為沒有根據個別文本調整預測結果所致，因此我們在以下的方法將根據模型調整輸出。

(二)、使用階段性推理及更清楚的指示

在這個環節中，我們參考第一名的作法，使用mistral和gemma輸出prompt，試圖獲得更好的結果，在第一名的原始做法中，只有使用原始文本及改寫後的文本讓模型輸出可能的prompt，在此我們將階段性的引導模型猜出正確答案；除此之外，我們也是試圖調整第一名中使用的 prompt，並觀察輸出結果。

1. 階段性模型輸出



圖七：階段性推理架構

我們分成兩個階段，第一個階段先設定了一些可能的prompt類型，限制模型的回答，讓模型先猜測可能的prompt類型，例如模型必須猜測出prompt可能

是涉及文法修正或是風格轉換等；接著在第二部分，我們將第一部分的回答加上原始的文本再放入模型中，讓模型預測最終果。

然而，此方法的輸出結果相當差，後續比對各輸出結果後發現，這是因為模型通常在第一階段的預測不太準確，接者便會導致第二階段的預測偏離更遠，例如原先是某種風格轉換的prompt，但由於風格轉換後文本的結構發生巨大改變，模型無法很好的預測到這是風格轉變或是改變文句結構。

2. 調整描述

由於第一部份模型的預測結果相當差，我們也改使用過另一種作法，直接將所有上述提到的描述一次性地餵入模型，讓模型從我們提供的提示中自行組合結果，以下是我們使用的描述：

"Please find the prompt. The transformation of Original_text to New_text followed a concise 3-7 word prompt aimed at [enhancing structure, clarity, and tone](#). It included guidelines on [style, tone, or grammar, requiring strict adherence without adding extra details](#). You should only answer the prompts and not add anything else"

除了模型的輸出，我們也加入了前面找到的mean prompt及第一名的magic prompt組合作為最終的輸出結果，在此做法下模型可以輸出比前面作法正常的結果，並且接近第一名的分數，以下是最終的分數：

	Team 53 - Version 6 Succeeded (after deadline) · 20d ago	0.7065	0.7075
<hr/>			
	Team 53 - Version 5 Succeeded (after deadline) · 20d ago	0.7099	0.7118

圖八：調整描述結果

以上的結果分別落在第六名和第三名的位置，相當接近第一名的成果，這也顯示模型可以透過我們的提示提高預測正確的機率。

(三)、四個模型的 SCS 比較

在看完各組影片以及實作完前面的內容後，我們覺得前幾名已經將 mean prompt、LLM predict prompt 都做得很好了，應該很難再對其做優化，因此我們想嘗試結合第七名與第一名的作法，將各自的優勢結合，以下是我們的思考與實作過程。

1. 修改輸出格式

以上我們有提到第一名的作法輸出格式如下：

pred_prompt_1 + pred_prompt_2 + pred_prompt_3 + pred_prompt_4 + magic prompt
- (1)

為了確保我們可以進行相關測試，我們使用第二名的資料集進行測試，該資料集包含 original_text、rewrite_prompt 以及 rewritten_text，我們如同第二名的作法，也假設該資料集的分布是與 test dataset 相似，並用此資料集進行測試。

經過我們的嘗試，若將 (1) 改為選擇四個提示詞中的三個，並再加上 magic prompt，我們可以得到如下的格式(2)

((四選三) + magic prompt) - (2)

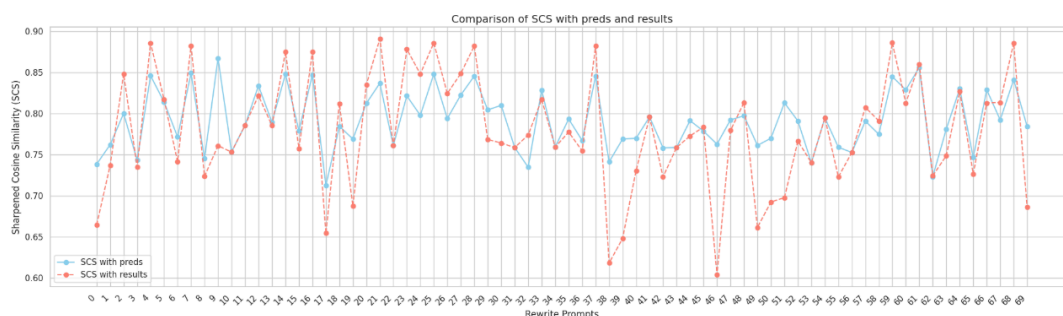
格式(2)與 rewrite_prompt 得到的 SCS 與原格式 (1) 相近，因此我們進一步改動，改為選擇兩個提示詞並去掉 magic prompt，如以下格式

((四選二) 並去掉 magic prompt) - (3)

根據我們的測試，我們別將格式 (3) 與 rewrite_prompt 做 SCS，並將原格式 (1) 與 rewrite_prompt 做 SCS，我們發現兩個結果之間的差距可能會大幅波動，可能會顯著提升或下降，因此我們設計了策略一，想利用模型之間的關係來優化。

2. 策略一

這一策略的核心是設置一個 threshold，計算兩兩模型預測結果 (pred_prompt) 的 Sharpened Cosine Similarity (SCS)，並選擇相似度最高的一組。如果這兩個模型的 SCS 高於 threshold，則認為它們的預測方向一致，這時不再加入 magic prompt (即 mean prompt)，以避免結果被拉回平均值，最終輸出格式為 (3)。若無兩組預測結果的 SCS 超過 threshold，則使用原始格式 (1)。不過，根據我在第二名的資料集測試，發現此策略存在問題。

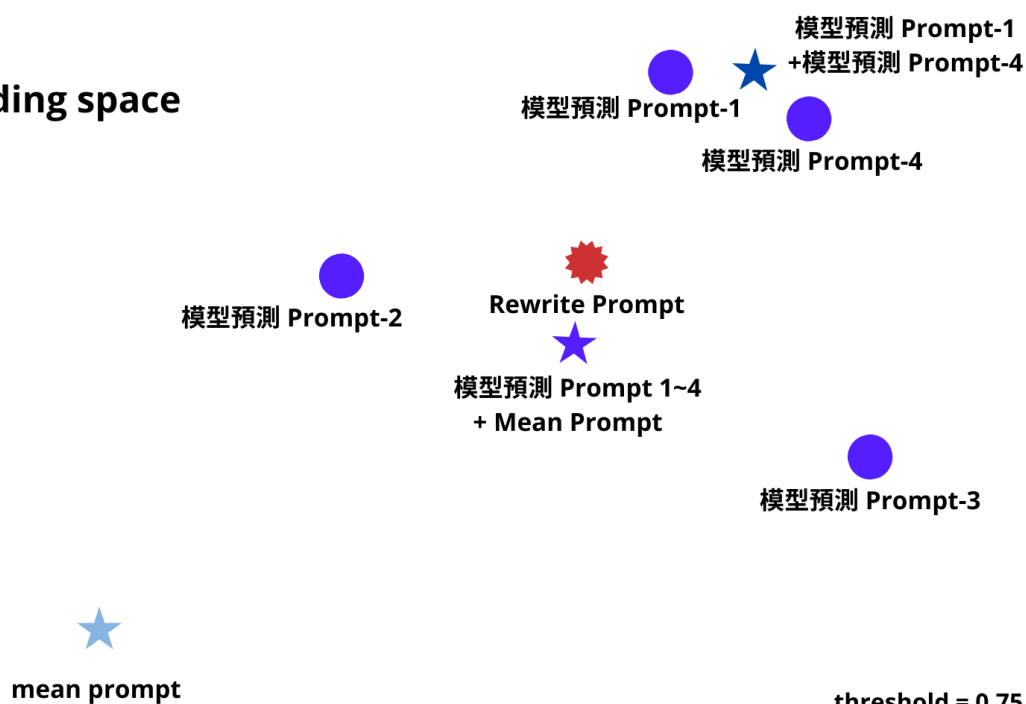


圖九：策略一輸出

上圖顯示了使用策略一時的結果，其中藍色代表原格式 (1) 與 rewrite prompt 做的 SCS，紅色代表格式 (3) 與 rewrite prompt 的 SCS，我們可以發現紅色相較藍色小幅提升但大幅下降，因為兩個 LLM 預測結果的 SCS 很高代表意思接近，但可能兩個都預測錯方向，導致跟 real rewrite_prompt 有很大的 bias，如下圖所示。

策略一

Embedding space

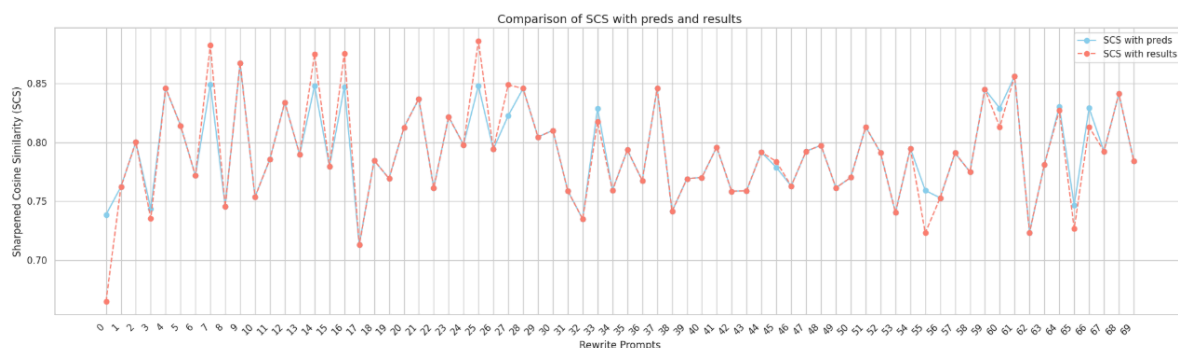


圖十：策略一示意圖

3. 策略二

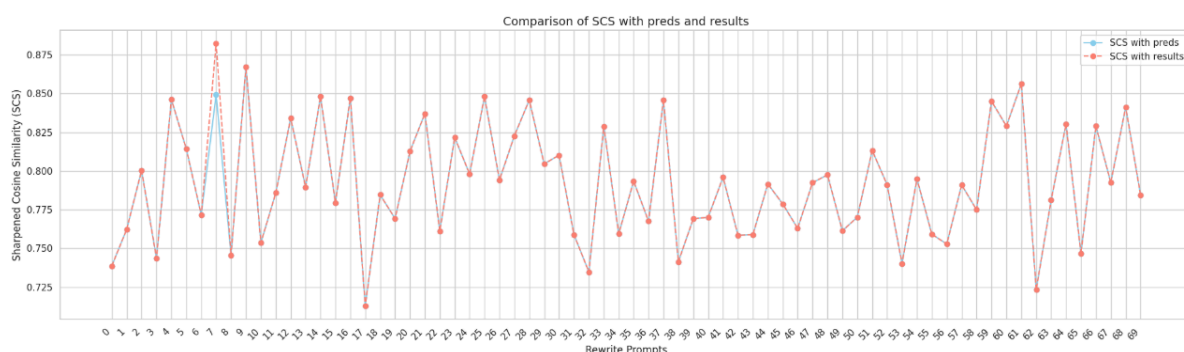
針對策略一的問題，我們一樣取 SCS 最高的一組來判斷，但在輸出前用 `min_scs_to_others` 來加強判斷，多一個 threshold，如果最高這組的兩個

pred_prompt 各自跟其他 pred_prompt 的 SCS 都大於 min_scs_to_others，才輸出格式 (3)，概念是確認四個 pred_prompt 的預測都是彼此都高度相似的才用格式 (3)，條件比較嚴格，測試結果如下。



圖十一：策略二輸出(1)

我們測試了不同參數的結果，經過更嚴格的參數調整可以達到以下的效果

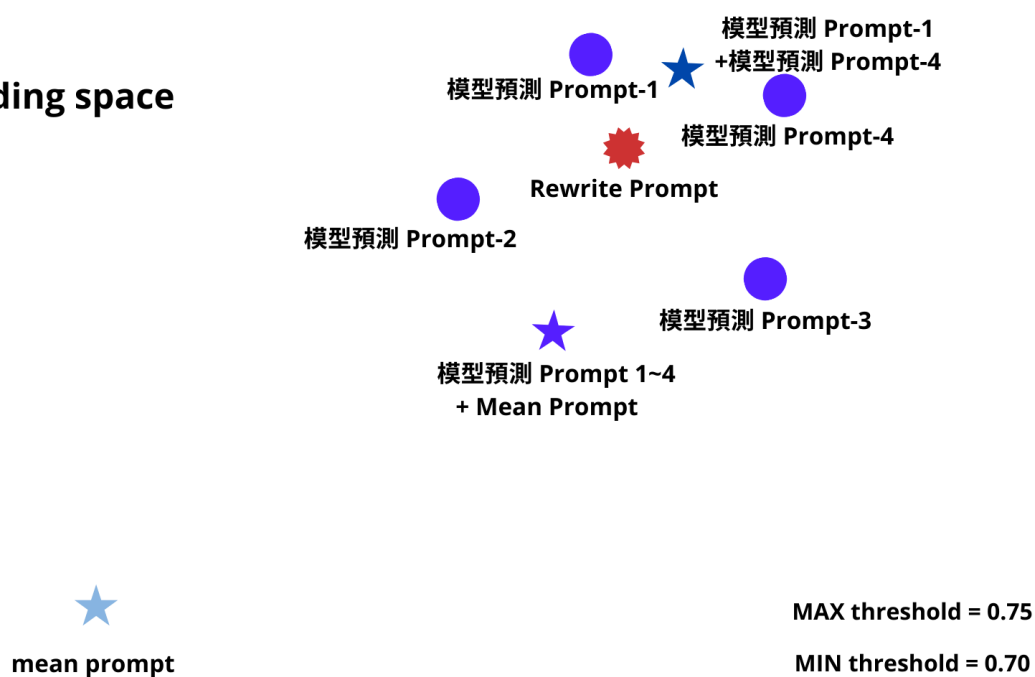


圖十二：策略二輸出(2)

我們認為用策略二可以更有效的判斷輸出的準確性，當四個模型輸出都相當接近時，能代表預測方向一致，這時不再加入 magic prompt (即 mean prompt)，以避免結果被拉回平均值，且策略二的示意圖如下：

策略二







Embedding space



圖十三：策略二示意圖

4. 結果

根據不同參數我們得出的結果如下圖所示，

 NLP_final - Version 20 Succeeded (after deadline) · 19d ago	0.7184	0.7172	<input type="checkbox"/>
 NLP_final - Version 19 Succeeded (after deadline) · 19d ago	0.7185	0.7171	<input type="checkbox"/>
 NLP_final - Version 18 Succeeded (after deadline) · 19d ago	0.7184	0.7172	<input type="checkbox"/>
 NLP_final - Version 17 Succeeded (after deadline) · 19d ago	0.7185	0.7171	<input type="checkbox"/>
 NLP_final - Version 16 Succeeded (after deadline) · 19d ago	0.7184	0.7172	<input type="checkbox"/>
 NLP_final - Version 15 Succeeded (after deadline) · 19d ago	0.7070	0.7063	<input type="checkbox"/>

圖十四：不同參數的結果

我們的實驗結果顯示，採用策略二後，整體表現有所提升。在 private 測試集上分數提升了 0.002，表現更好了，不過，在 public 測試集上的分數卻稍微下降了 0.0007。我們覺得這可能是因為 public 測試集的資料特性不同，讓策略二的條件過於保守，沒能完全發揮它的優勢。

雖然如此，我們還是覺得策略二在過濾掉偏差大的預測上確實有用，特別是在提升模型一致性和穩定性方面效果不錯。

四、結論與討論

在這次的 Term Project 中，我們首先理解了目前的解決方案與策略，並嘗試自行實作與利用課堂中的知識或 NLP 的相關技巧去改進，我想以下分成三大部分來討論。

(一)、遇到的問題與挑戰

在這次的 Term Project 中，我們一開始的想法是透過尋找更好的 mean prompt 並利用迭代優化的方法來找出最佳解，最終生成與目標語義更接近的 mean prompt。然而，當我們將此方法實作出來，並將結果提交到Kaggle後，我們發現這種方法對於該任務的效果並不是最理想的，結果未達到我們預期的效果，且所找到的 mean prompt表現不如其他前幾名的組別。

因此，我們進一步引入了 chain of thought 的概念，讓模型在進行提示詞改寫之前，先判斷 rewrite_prompt 的類別，再進行相應的改寫。這一方法的設想是通過讓模型先理解文本的核心意圖，再依據類別進行處理，以提高準確性。然而，由於我們所面對的類別過於廣泛，模型在進行類別判斷時經常出現誤判的情況，最終並未達到預期的效果，且這種方法並未顯著改善結果。

最後，我們結合了多模型的輸出，藉由不同模型之間的互補性來提升提示詞的準確度。這種多模型融合的方式在某些任務中確實取得了較好的效果，在Leaderboard上取得了Private Score第一、Public Score第二的成績。雖然如此，這種方法仍然依賴於多個模型的判斷，因為模型可能會出現 bias，仍有可能出現未能理想生成的情況，我們只是盡量去避免。

(二)、任務本身難度高

在這個任務中，最大的挑戰在於如何準確找到「rewrite prompt」，即便是前幾名的組別，他們的解決方案也並不是確切地找出 rewrite prompt，而是通過讓多個模型生成更 diverse 的 rewrite prompt來提高表現。需要確保多個模型各自輸出結果正確，且各自之間夠 diverse。

此外，我們在實作過程中採用了 T5 模型來尋找「mean prompt」，但這個 mean prompt 的特性並不具備通用性。對於不同的 embedding model 來說，mean prompt 會有所變化，這使得它並不是一個普遍的解決方案，僅適用這個 competition。

(三)、任務實作複雜

除此之外，這個 competition 本身的設計也增加了挑戰性，因為它並未提供一個現成的資料集。這要求我們必須自行生成相關資料，而這一過程充滿了不確定性。由於無法保證生成的資料與測試數據的分布一致，我們很難確保訓練和測試的條件是相同的。若要在這樣的情況下獲得良好的結果，根據前面名次組別的做法，我們必須先找到有效的 mean prompt，並對四個大型語言模型（LLM）進行微調，這樣才能保證模型能夠更準確地生成對應的提示詞。每次進行這樣的操作需要消耗大量的時間，單次執行的時間至少需要 5 小時以上，而這種反覆調整修改的過程使得這個 competition 變得更加艱難。

透過這次的 Term Project，我們深刻體會到 Prompt Engineering 的重要性，雖然現在隨著 LLM 的優化，Prompt Engineering 可能會漸漸變得不那麼重要，但這次的經歷讓我們意識到，反向操作的 LLM Prompt Recovery 其實是一個極具挑戰的過程。我們從中學到了 T5 模型的運作原理、LLM 微調（fine-tuning）的技巧，還有 Adversarial Attack 等概念，並且實際實作，這些經驗讓我們對整個過程有了更深的理解。

五、參考文獻

- 你是個準一級LLM咒言師嗎？-淺談 prompt 逆向工程，
<https://ithelp.ithome.com.tw/articles/10358361>
- [LLM Prompt Recovery](PV:0.67)銀牌方案總結，
https://roschildrui.github.io/2024/04/30/prompt_recovery/
- LLM-Prompt-Recovery-Fine-Tuning-T5-for-Success，
<https://github.com/Salma0-8/LLM-Prompt-Recovery-Fine-Tuning-T5-for-Success/tree/main>
- Kaggle Solution Walkthroughs: LLM Prompt Recovery with Team Danub，
<https://www.youtube.com/watch?v=u81-7IWAZOY&list=PLqFaTIg4myu8abFIzOPREdrH-NGhOBGGI>

code link：[GitHub - junfu1209/NLP-Cup](#)