國立中正大學應用數學研究所

碩士論文

基於 LLaMA 3.1 和 BART 大語言模型的長文本摘要多層次演算法

Multilevel Algorithm for Long Text Summarization Based on LLaMA 3.1 and BART Large Language Models

蘇珮君

指導教授：賴振耀 博士

中華民國 114 年 01 月 09 日

摘要

本論文提出一種基於 BART 模型的多層次摘要演算法，來處理注意力偏差的問題。此演算法通過限制輸入 token 上限方式，結合分段生成與逐層壓縮摘要，平衡了信息保留與文本壓縮效率。實驗採用 CNN/DailyMail 資料集，並以 ROUGE 指標進行性能評估。結果顯示這種設計在處理長文本摘要時能夠保持語意完整性，因此在長文本摘要任務中具有一定的優勢。

關鍵字：長文本摘要、多層次演算法、LLaMA 3.1、BART 模型

Abstract

This paper introduces a multilevel summarization algorithm based on the LLaMA 3.1 and BART models to deal with attention bias issues. This method balances information retention and text compression efficiency by imposing the upper limit of input tokens, generating segmented summaries and compressing them layer by layer. Experiments were conducted using the CNN/DailyMail dataset, with performance evaluated through ROUGE metrics. The results show that the algorithm offers clear advantages for long-text summarization tasks.

Keywords: Long-text summarization, multi-level algorithm, LLaMA 3.1, BART model

目 錄

圖 目 錄

表 目 錄

第一章　緒論

1.1　研究動機與背景

長文本通常指的是超過數百字甚至數千字的文檔或文章，例如數千字以上的學術論文、技術手冊、政府或法律文件等結構化長文本 (structured long text documents)，以及數千字以上的自由風格的報告或評論、對話記錄和訪談等非結構化長文本 (unstructured long text documents)。在自然語言處理 (NLP) 領域中，長文本通常指其長度超過語言模型或演算法的輸入限制。例如，GPT 或 BERT 等模型的輸入有固定的字元或 token 限制，例如 GPT-3 的 token 限制為 4096 (約等於 3,000 個英文字)，至於常見的各大語言模型的 Tokens 數限制請參閱表1.1。

大語言模型 (LLMs)	Tokens	大語言模型 (LLMs)	Tokens
MPT-30B-Instruct[1]	8192	LongChat-13B (16K)[1]	16384
GPT-3.5-Turbo	4K	GPT-3.5-Turbo (16K)	16K
Claude-1.3	8K	Claude-1.3 (100K)	100K

表 1.1: 各大語言模型輸入的 Tokens 數限制

雖然像是 GPT、BERT 等大型語言模型 (LLMs) 已在語言理解、文本生成和機器翻譯等任務上展現了顯著的能力，透過不斷擴大上下文窗口來容納更多的 token，但對於長文本的處理仍然存在一些限制。特別是在 Transformer 的架構下，注意力機制會因位置偏差 (Position Bias) 而更傾向於關注文本開頭與結尾的資訊，忽略中間內容。

在 Liu 等人 [9] 的研究指出，這種偏差在長文本生成和問答任務中特別明顯。透過調整相關段落在上下文中的位置，發現當問答任務的關鍵資訊位於上下文開頭時，模型性能最佳，而位於中間段位置時性能下降，呈現 U 形曲線。而 Ivgi 等人 [5] 和 Guo 等人 [15] 的研究結果也同樣支持這個結論。

[1]MPT-30B-Instruct 是由 MosaicML 開源的語言模型，適合生成文本和執行指令驅動的多樣化任務；而 LongChat-13B (16K) 則由 LMSYS Org 開發，基於 LLaMA 模型進行微調，並以 ShareGPT.com 收集的對話作為訓練集，專為長文本生成與深度語境理解設計。兩者均採用 Transformer 架構。

此外，Liu 等人 [9] 利用 NaturalQuestions-Open 數據集，分別使用表1.1中的 GPT 和 Claude 模型將關鍵資訊進行位置改變、多文檔問答和鍵值檢索這三項任務，結果發現擴展上下文窗口未必能顯著提升模型在長文本中的性能，並且性能會隨著文本長度增加而逐漸飽和，表明擴展窗口並非解決長文本問題的唯一方法。

在文獻 [11] 中，Pan 和 Wu 等人指出現有語言模型使用資訊熵 (Information entropy) 來刪除提示中的 token 或詞彙單位進行壓縮的做法，往往會導致以下兩個主要問題：(1) 單向的上下文失去關鍵信息; (2) 無法保留完整信息，改善上下文缺失。為了解決這些問題，他們設計了新的方法，從大型語言模型 (LLM) 中提取知識壓縮提示、引入抽取式文本壓縮數據集並且採用 transformer 架構等方法，來改善使用 GPT-4 壓縮過程中的信息保留和模型生成結果的真實度問題。

在該論文的數據蒸餾 (Data Distillation) 小節裡，作者想要通過改進 GPT-4 的提示壓縮效果，達到信息保留、真實度和效率之間的平衡。具體方法是讓 GPT-4 壓縮原始文本，並滿足以下三個標準 (1) 減少 token 數量：降低成本並加快推理速度; (2) 保留關鍵信息：避免重要信息被過度壓縮; (3) 確保壓縮後的提示忠實於原文：避免生成虛構內容。但在 Jiang 等人 [6] 的實驗發現，GPT-4 在壓縮過程中經常無法保留原始文本中的關鍵信息，在 Jiang 等人 [6] 的初步實驗中觀察到，GPT-4 常常修改原始表達方式，有時甚至生成虛構的內容，會影響後續任務的準確性。

在 Jiang 等人 [6] 和 Huang 等人 [4] 過去的研究中，通常會在指令中設定壓縮比率或目標 token 數量。然而，GPT-4 常常無法嚴格遵守這些限制。文本的信息密度會因體裁、風格等因素而有顯著差異。例如，新聞文章通常比會議記錄含有更密集的信息，即使在會議記錄中，不同講話者的信息密度也會有所不同。

基於這些觀察，在 Pan 和 Wu 等人 [11] 研究中的提示詞設計不加入固定壓縮比率的限制，改為提示 GPT-4 在盡可能保留多信息的情況下，盡量將原文壓縮至最短，而這樣的提示可以讓 GPT-4 自動根據不同的句子應用不同的壓縮比率。但針對長文本壓縮問題，此研究進一步的發現 GPT-4 在處理長上下文時，會自動運用較高的壓縮比率，這會導致大量信息流失。為了解決這個問題，**此研究設計了一種分塊壓縮方法，將長文本分割成不超過 512 個 token 的段落，並在每個段落的句號處結束，然後對每個段落單獨進行壓縮。**

基於以上的考量，Pan 和 Wu 等人 [11] 設計了新的蒸餾程序，集中在「提示詞設計」和「分塊壓縮」兩個方面。提示詞設計：

1. You can ONLY remove unimportant words.（只能刪除不重要的詞彙）
2. Do not reorder the original words.（不能重新排序原始詞彙）
3. Do not change the original words.（不能改變原始詞彙）
4. Do not use abbreviations or emojis.（不能使用縮寫或表情符號）
5. Do not add new words or symbols.（不能添加新詞或符號）

本研究針對大型語言模型在長文本中忽略中間內容的問題，設計了一種多層次摘要演算法，多層次摘要的原理是通過 (1) 分塊壓縮 (Pan 和 Wu 等人 [11] 的做法)，每段的長度控制在模型的上下文窗口內;(2) 對每一段生成摘要，提取關鍵信息;(3) 逐步合併這些分塊的摘要進行多層次的壓縮。這種多層次摘要的優點是在每一層摘要中關鍵信息都可以被提取到，也許可以改善大語言模型忽略中間段位置關鍵信息的傾向。

1.2 相關工作

為了提升大型語言模型（LLMs）在處理長文本時的性能，不同研究針對上下文窗口限制和注意力偏差等問題提出了創新的解決方案。

首先，Chen 等人 [1] 提出了「MEMWALKER」方法，專注於克服傳統自注意力機制限制上下文窗口大小的瓶頸。MEMWALKER 的核心概念是模仿人類的閱讀方式，將模型視為「互動式閱讀者」，通過逐步檢索與總結來有效利用長文本資訊。該方法主要分為兩個階段：

1. 記憶樹的構建：將整篇長文本切分為多個片段，對每個片段進行摘要，並逐層壓縮成一個樹狀結構，最終形成涵蓋所有資訊的根節點。這種結構使模型能快速定位相關資訊。

2. 導航與檢索：當模型接收到查詢（Q）後，從根節點開始，逐步搜尋與問題相關的摘要節點，最終生成回應。這一過程類似人類先瀏覽目錄再深入檢索的方式。

該實驗結果顯示，MEMWALKER 在長文本問答（QA）任務中取得了顯著優於現有方法的表現，有效突破了上下文窗口的限制。

Xu 等人 [14] 將擴展上下文窗口與信息檢索增強結合進行比較，並展示了信息檢索增強方法在長文本處理中的顯著優勢。研究表明，即使在上下文窗口較小的模型中，簡單的信息檢索增強也能顯著提高性能，並在問答（QA）和基於查詢的摘要任務中取得與較大上下文窗口模型相當的效果。特別是，結合信息檢索的模型不僅性能更好，且生成速度更快，為選擇適合的長文本處理策略提供了有價值的實踐參考。

He 等人 [2] 聚焦於解決注意力機制中的位置偏差問題，提出了一種名為 PAM QA（Position-Agnostic Multi-step QA）的訓練方法。該方法通過設計特殊的訓練任務，使模型在長文本中能夠忽略資訊的位置，專注於內容本身的相關性，從而有效搜尋並提取所需資訊。實驗結果表明，PAM QA 方法在多文檔問答（Multi-doc QA）和其他基準測試中顯著提升了模型的性能，特別是在隨機排列文本的情境下，準確率有顯著的提升。

最後，在 Cheng 等人 [3] 的研究中提出了一種校準機制來解決位置偏差的問題。此校準方法被稱為「found-in-the-middle」，其通過重新校準注意力權重，確保中間內容資訊能獲得足夠的關注。該方法採用基於位置的調整策略，使注意力分布更緊密地與資訊的相關性對齊，而非僅依賴其在文本中的位置。

這項研究中提出的校準機制在多項任務中展示了顯著的改進，包括：

- 檢索增強生成（Retrieval-Augmented Generation, RAG）任務：模型需要從長文本中檢索相關資訊並生成適當的回應。
- 長上下文理解任務：要求模型在處理包含大量資訊的長文本時，能夠有效地理解和利用其中的內容。

他們的實驗結果顯示，所提出的校準機制在這些任務中達到了顯著的性能提升，特別是在檢索增強生成任務中，性能指標提高了多達 15%。這突顯了解決位置注意力偏差的重要性。

1.3 論文架構

本研究共分為七個章節：第一章「緒論」說明研究動機與背景，概述相關工作，並介紹論文的整體架構與內容安排；第二章「模型介紹」依次介紹 Transformer 架構、BART 模型及 LLaMA 3.1 模型，探討其在文本處理中的核心功能與應用；第三章「多層次摘要演算法」詳述演算法的設計步驟與實現方法。

第四章「評估指標」介紹實驗中使用的 CNN/DailyMail 數據集與 ROUGE 評估指標的計算方法與適用性；第五章「實驗設計」說明實驗細節，包括使用的方法、參數選取及不同壓縮策略；第六章「實驗結果」呈現基於不同設計的實驗成果，並分析演算法的效能與表現；第七章「結論與未來展望」總結研究發現與貢獻，並指出限制與未來改進方向。

第二章　模型介紹

2.1　Transformer 架構簡介

Transformer 是一種基於注意力機制的神經網路架構，由 Vaswani 等人在 2017 年提出 [13]。其論文改變了傳統序列建模的方式，為自然語言處理領域帶來了重大突破。Transformer 的主要特點在於完全拋棄了循環神經網路（RNN）和卷積神經網路（CNN），改以自注意力機制（Self-Attention）處理輸入文本，使模型能夠容易抓住句子中遠距離單詞之間的關係，同時減少訓練所需的時間。

Transformer 採用編碼器-解碼器（Encoder-Decoder）架構，主要由兩個對稱的部分組成：編碼器負責將輸入文本轉化為語義表示，解碼器則根據語義表示生成輸出文本。每一部分都是由多層相同的結構堆疊而成，層與層之間通過殘差連接 (Residual Connection) 相互作用，確保模型訓練能穩定的傳遞。特別之處在於引入了自注意力機制（Self-Attention），該機制可以讓每個單詞根據整個序列中其他單詞的相關性進行動態加權。通過構建查詢（Query）、鍵（Key）和值（Value）三個向量，自注意力機制根據這些向量計算序列中不同單詞之間的相關程度，並根據權重生成最終表示。

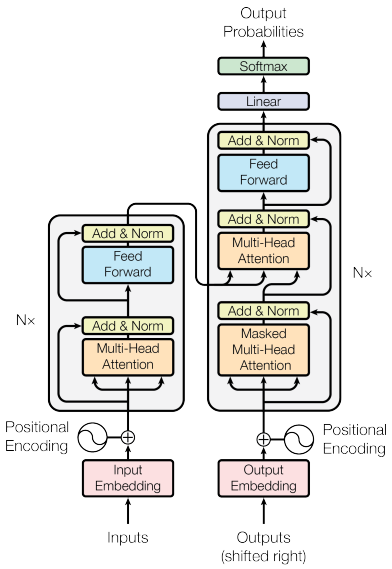

圖 2.1: Transformer 架構圖

基本的 Transformer 由 6 層編碼器 (Encoder) 與 6 層解碼器 (Decoder) 組成，每層均包含多頭注意力機制（Multi-Head Attention）和前饋神經網絡（Feed-Forward Neural Network）。此外，Transformer 中使用了位置編碼（Positional Encoding）來引入序列中的位置信息，克服了其結構中缺乏順序依賴的問題。

自注意力機制的基本形式在處理複雜語義結構時可能存在一定的侷限性，特別是當模型需要同時捕捉句子中細節關係和整體脈絡時。為了解決這一問題，Transformer 引入了多頭自注意力機制（Multi-Head Self-Attention），使模型能夠在多個子空間中並行學習序列的不同層次語義關係。與單一注意力機制相比，多頭注意力機制能捕捉更多樣化的關聯性，從而增強對輸入序列的表達能力。

2.1.1 Scaled Dot-Product Attention

在 Vaswani 等人的文獻 [13] 中，Scaled Dot-Product Attention 讓 Transformer 能夠輕鬆處理句子中需要跨很遠距離才能建立關聯的詞語，並避免了像 RNN 那樣因為逐字處理順序而丟失部分信息。

首先，給定輸入序列 $X = [X_1, X_2, \ldots, X_n]$，每個單詞向量 X_i 通過線性變換映射為三組不同的向量表示：查詢（Query, Q）、鍵（Key, K）和值（Value, V）。具體生成方式為：

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V. \tag{2.1}$$

其中，W^Q、W^K 和 W^V 是學習到的權重矩陣，分別用於生成查詢、鍵和值向量。

接著，通過查詢向量 (Q) 與鍵向量 (K) 的轉置進行矩陣乘法（MatMul），計算出每個查詢與所有鍵之間的相似度分數：

$$\text{Scores} = QK^T. \tag{2.2}$$

這裡的 MatMul 操作將查詢矩陣 Q（大小為 $n \times d_k$）與鍵矩陣 K^T（大小為 $d_k \times n$）相乘，其中 d_k 為 Q 和 K 的向量維度，輸出的分數矩陣形狀為 $n \times n$，表示序列中每個位置的查詢向量與所有位置的鍵向量之間的相關性。分數越大，表示查詢與對應鍵的相似度越高。

為了在序列生成任務中更有效地處理上下文信息，Transformer 在解碼器（Decoder）階段引入了 Mask Attention 機制。目的是確保解碼器在生成第 t 個位置的詞語時，只能關注先前已生成的詞語，避免未來信息的洩漏。這一個特性對自回歸生成（Autoregressive Generation）任務尤其重要，如機器翻譯或文本生成等場景。在解碼器中，Mask Attention 的計算公式如下：

$$\text{Mask Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V. \tag{2.3}$$

其中，M 是遮罩矩陣（Mask Matrix），通常是一個與 QK^T 相同大小的矩陣，用於控制某些位置的注意力分數。假設輸入序列長度為 n，那麼 M 是一個 $n \times n$ 的矩陣，其元素的定義為：

$$\text{Mask}[i, j] = \begin{cases} 0, & \text{if } i \geq j, \\ -\infty, & \text{if } i < j. \end{cases} \tag{2.4}$$

這代表當 $i < j$（也就是未來的位置）時，遮罩矩陣的值為負無窮大（$-\infty$）。這樣在經過 softmax 函數時，這些位置的權重會被壓制為零，不參與注意力計算。而當 $i \geq j$（也就是目前或過去的位置）時，遮罩矩陣的值為零，不會對計算造成影響。透過這種方式，模型可以專注於序列中已生成的部分，達到屏蔽未來資訊的目的。

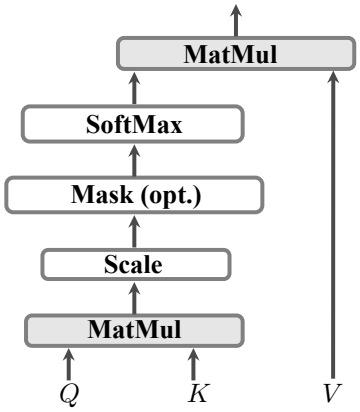

圖 2.2: Scaled Dot-Product Attention.

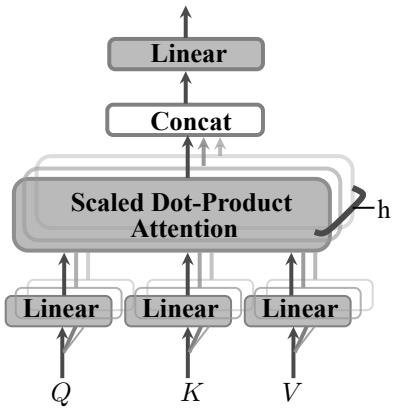

圖 2.3: Multi-Head Attention.

2.1.2 Multi-Head Attention

為了進一步增強模型的表現力，文獻 [13] 中 Transformer 還採用了多頭自注意力機制（Multi-Head Attention）。該機制就像是多組獨立的觀察角度，每個頭代表一

組獨立的 Query-Key-Value 組合，從不同的角度學習句子中的語義信息，最後再把這些結果合併起來，形成更加全面的理解。

為了實現多頭機制，Q、K 和 V 被分解為 h 個子空間，並為每個子空間生成獨立的權重矩陣 (如圖2.3所示)：

$$Q_i = QW_i^Q, \quad K_i = KW_i^K, \quad V_i = VW_i^V. \tag{2.5}$$

這裡，W_i^Q、W_i^K 和 W_i^V 是每個頭的投影矩陣，用於將輸入映射到不同子空間。在每個子空間中，注意力機制的計算方式為：

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i). \tag{2.6}$$

其中，注意力函數的具體形式為：

$$\text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i. \tag{2.7}$$

在這一步，查詢和鍵的內積用於計算序列中每個位置的相關性，其中 d_k 是 Q_i 和 K_i 的向量維度，經過 softmax 正規化後得到的權重用於加權值矩陣 V，最終生成該頭的輸出（如圖 2.2 所示）。

最後，所有頭的輸出結果將被拼接起來，並通過一個線性變換整合為多頭注意力的最終輸出：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O. \tag{2.8}$$

其中，W^O 是整合後輸出的線性投影矩陣，用於進一步生成多頭注意力的輸出。

除了自注意力機制，還包含一些關鍵設計。在圖2.1中可以看到，輸入資料首先經過 Embedding 處理，轉換為密集的向量表示，之後進行位置編碼（Positional Encoding），因為文獻 [13] 中的 Transformer 不具備序列結構，這部分用於為每個單詞添加位置信息，確保模型理解輸入文本的順序。其次是前饋神經網路（Feed-forward Network），它會對每層的輸入進行非線性轉換，進一步提升模型對複雜模式的配適能力。此外，層歸一化（Layer Normalization）和殘差連接則是有助於讓訓練更加穩定，也能加速模型的收斂。這些設計讓 Transformer 在處理大量數據時能快速並行運算，並且在機器翻譯、文本摘要和問答等任務中表現得非常好。

2.2 BART 模型

BART（Bidirectional and Auto-Regressive Transformer）[1]是一種基於 Transformer 架構的序列到序列（seq2seq）模型，由 Facebook AI Research 團隊於 2019 年提出 [7]。它結合了 BERT 的雙向編碼器（Bidirectional Encoder）和 GPT 的自回歸解碼器（Autoregressive Decoder）的優勢，兼具生成能力與語義理解能力。BART 的主要特點是作為一種降噪自編碼器（denoising autoencoder），能夠從損壞的文本中恢復原始文本，特別適合處理生成與重建類的任務。在摘要生成、機器翻譯等自然語言處理任務中，BART 都展現了出色的性能。

BART 的結構使用了標準的編碼器-解碼器架構。其編碼器與解碼器均都是由多層 Transformer 組成的，保留了 Transformer 高效並行處理和自注意力機制的優勢。在編碼器部分，BART 會把帶有損傷的文本（如刪減、遮蔽字詞、重新排列等）進行編碼處理，提取出其中的語義資訊；接著，在解碼器部分，模型會根據這些語義表示逐步生成恢復後的完整文本。這種處理方式，強化了模型的重建能力，適用於多種生成類的任務。其架構的核心特性之一是做為一種去噪自編碼器，目的是通過重建被損傷的文本來學習語義和結構，損傷的類型包括：

- Token Masking：隨機遮蔽部分單詞。
- Token Deletion：隨機刪除單詞，讓模型學習在不完整信息下恢復文本。
- Text Infilling：隨機替換一段連續的單詞為特殊標記，要求模型填補空白。
- Sentence Permutation：打亂句子的順序，使模型學會重新排列。
- Document Rotation：改變段落或文檔的起始位置。

BART 的設計結合了雙向編碼與自回歸解碼的優勢。對於編碼器的部分，借鑒了 BERT 的設計，能夠同時考慮單詞的前後文語境，這讓模型對整體文本的理解更加全面。而在解碼器的部分，BART 又採用了像 GPT 的自回歸生成方式，每次生成一個單字，然後把這個單詞作為下一步的輸入來預測接下來的單詞。這樣的結合設計，讓 BART 既擅長理解文本，也擅長生成高質量的內容。其應用場景例如：

[1] https://huggingface.co/docs/transformers/model_doc/bart

- 生成任務：機器翻譯、文本摘要、自動對話生成。
- 理解任務：情感分析、句子分類、文本填空等。

BART 提供了多種規模的模型以適應不同的應用場景，其中最具代表性的是 bart-base 和 bart-large：

- bart-base：包含 6 層編碼器與 6 層解碼器，總參數量約為 140M。由於他的模型規模相對較小，因此能夠在計算資源有限的環境中高效運行，適合需要快速開發、實時應用的場景。
- bart-large：包含 12 層編碼器與 12 層解碼器，總參數量約為 406M。由於他更大的模型容量，bart-large 在生成任務和理解任務中展現出了更強的性能，尤其是在需要處理高複雜度語言結構或長文本生成的場景中效果特別突出，如文本摘要、機器翻譯和自然語言生成任務。

這兩種模型分別適應了不同資源條件下的需求：bart-base 為資源受限場景提供了高效選擇，而 bart-large 則在追求卓越性能的應用中展現了強大的實力。基於這些架構，BART 的各種變體被廣泛用於多樣化的自然語言處理任務，擁有高度的是適應性與實用性。

2.2.1 facebook-bart-large-cnn 介紹

BART 模型在文本摘要任務上的應用，主要以微調模型的形式呈現，以適應不同的摘要需求。Facebook AI Research 團隊針對不同的應用場景，開發了基於 BART 的多種微調模型，其中最具代表性的模型為 facebook/bart-large-cnn，該模型是基於 CNN/DailyMail 資料集微調的，用於生成長文本摘要。CNN/DailyMail 資料集以新聞報導為主，提供了高度結構化的文本與摘要對，使得模型能在生成多句式摘要時展現出優異的性能。facebook/bart-large-cnn 特別適合處理篇幅較長的文章，並在語法與語義上保持高度的自然性與一致性。根據官方釋出的結果，該模型在 CNN/DailyMail 資料集上達到了以下 ROUGE 分數[2]：

[2]url:https://huggingface.co/facebook/bart-large-cnn

ROUGE-1	ROUGE-2	ROUGE-L
42.949	20.815	30.619

此外，針對生成極短摘要的應用場景，Facebook 團隊也開發了基於 XSum 資料集微調的 facebook/bart-large-xsum。相比於長摘要模型，該模型聚焦於資訊濃縮，能生成單句式的高密度摘要。此外，BART 還擁有多種不同的變體，專門針對不同的任務場景進行調整，例如：針對推理的 bart-large-mnli 模型和問答任務的 bart-large-squad 模型等。展現出了對各類自然語言處理任務的廣泛適用性。

2.3 LLaMA 3.1 模型介紹

LLaMA 3.1（Large Language Model Meta AI）[3] 是由 Meta AI 開發免費提供研究社群使用的開源 (open source) 大型語言模型，其設計目的是通過精簡的架構實現高效的語言理解與生成，同時降低計算資源消耗。該模型延續 LLaMA 系列的設計理念，專注於提供高效的語言處理能力，並保持相對較小的參數規模 (8B)，以降低計算資源的需求。LLaMA 3.1 特別針對長文本處理進行了優化，在自然語言生成、對話系統和長文本摘要等任務中表現出色。相較於 GPT-4 (參數規模約為 175B 到 1.8T) 等其他同性質的大型語言模型，LLaMA 3.1 雖然使用了較少的參數，卻能夠提供相似甚至更高效的語言理解與生成能力。

2.3.1 模型架構

LLaMA 3.1 採用了基於自注意力機制的 Transformer 架構，因此可以高效地處理長文本。Transformer 的結構主要由兩個關鍵部分組成：

1. 多頭自注意力機制：這個機制讓模型在處理文本時，能靈活地調整每個詞之間的權重，根據上下文捕捉詞語之間的關聯性，並根據整體上下文來預測下一個詞的機率。

[3] https://huggingface.co/meta-llama/Llama-3.1-8B

2. 前饋神經網絡（FFN）：在每一層的注意力機制後，會使用前饋神經網絡進行非線性轉換，幫助模型進一步提取和處理語言特徵。

LLaMA 3.1 在架構上基於多層的自注意力和前饋層，使其能夠學習到更加豐富的語言模式。針對長文本處理，LLaMA 3.1 對上下文窗口進行了擴展，使得它在處理長篇文章、生成摘要或應對複雜對話時，能夠保持更好的上下文連續性。

2.3.2 訓練與優化

LLaMA 3.1 採用大規模的自監督學習方法，通過從涵蓋各種語言、主題和文本風格的大量高質量數據中學習，掌握語言結構和語意關聯。在優化過程中，模型使用了參數壓縮技術和稀疏性機制，有效降低記憶體使用的需求和運算成本，來提高推理效率。此外，LLaMA 3.1 在微調方面非常靈活，允許用戶根據特定任務或應用場景進行訓練，以達到更專業的表現。

2.3.3 模型的應用

LLaMA 3.1 在各類型的 NLP 任務中都表現得很出色，以下是幾個典型的應用領域：

- 長文本摘要生成：能生成語義豐富且連貫的摘要，特別適用於新聞、學術文章和技術報告等長篇文檔的總結或概括。
- 問答系統：由於強大的語言理解與生成能力，LLaMA 3.1 能根據用戶的問題生成準確、相關且自然流暢的回答，提升對話系統的交互體驗。
- 內容創作：生成的文本具備邏輯連貫性和語義深度，廣泛應用於創作小說、社群平台文章等內容生成任務中，滿足不同文本風格的生成需求。
- 機器翻譯：支持高準確度的多語言翻譯，能處理不同語言之間的語義轉換，提升翻譯質量和流暢度。

LLaMA 3.1 提供靈活的微調能力，允許針對特定領域或任務進行訓練，能夠應用於法律、醫療等專業領域的文本處理需求。

LLaMA 3.1 是一個強大且高效的語言模型，基於精簡的 Transformer 架構，結合先進的訓練技術與優化方法，提供了卓越的語言理解與生成能力。其開源的特性使得它成為學術研究、技術開發和商業應用中的理想工具。無論是在文本摘要、對話系統、專業報告生成還是創意內容創作中，LLaMA 3.1 都展現出了廣泛的適應力和應用潛力。

2.3.4 Ollama 平台介紹

本研究中使用的 LLaMA 3.1 模型來自開放平台 Ollama[4]。由於，LLaMA 系列模型需要專業的部署環境與硬體資源，對一般使用者而言使用門檻較高。Ollama 平台基於 Meta 的開源許可，將這些模型封裝為易於操作的工具，並對模型運行進行優化，讓使用者可以輕鬆地在本地端下載、安裝和使用這些模型，不需要依賴雲端服務。

Ollama 的最大特色是操作簡單，幾乎不需要技術背景即可上手。例如，本研究使用的 LLaMA 3.1-8B 模型，只需執行兩個指令即可完成整個安裝與啟動流程：

1. 安裝模型：輸入 **ollama install llama-3.1-8b**，系統會自動下載並配置模型。
2. 啟動模型：輸入 **ollama run llama3.1** 即可啟動模型進行推理或測試。

即使在硬體資源有限的情況下，也能讓使用者快速運行 LLaMA 模型，大幅降低了技術門檻。

除了 LLaMA，Ollama 平台還支援多種其他高性能模型，並提供靈活的選擇以滿足不同需求。例如，Phi 1.5 是一款專注於中文任務的模型，特別適合處理與中文相關的自然語言理解和生成問題，對需要高品質中文語言處理的任務來說是一個不錯的選擇。而對於硬體資源有限的用戶，Ollama 平台提供了多種輕量化且高效能的模型，例如 Mistral 7B，他能夠在輕量化的設計下有很出色的生成性能。另外，像

[4] https://ollama.com/

Falcon 7B 和 GPT-J 6B 等模型，也提供了使用者在更多資源限制下的選擇，這些模型能以較低的硬體需求完成高效推理。

除此之外，Ollama 還允許用戶導入其他格式的模型，例如 Hugging Face 模型格式，讓研究者和開發者能夠根據自身需求與應用場景靈活調整工具和資源，進一步提升平台的靈活性與擴展性。

第三章　多層次摘要演算法

多層次摘要演算法的核心目的是在有效利用模型資源的同時，逐層壓縮和濃縮文本資訊，以生成最終的摘要。這種方法不僅克服了長文本處理中的技術限制，還能在資訊壓縮與語義保持之間取得平衡，是大語言模型（LLMs）在處理長文本應用中很重要的一種技術。

3.1　演算法步驟

多層次摘要演算法的框架，如表3.1所示，主要分成三個步驟：文本段落的拆分、摘要生成和摘要合併。

layer	
original	$[T_1^{(1)}],\quad [T_2^{(1)}],\quad [T_3^{(1)}],\quad [T_4^{(1)}],\ldots,\quad [T_{m_1}^{(1)}]$
summary 1	$[S_1^{(1)}],\quad [S_2^{(1)}],\quad [S_3^{(1)}],\quad [S_4^{(1)}],\ldots,\quad [S_{m_1}^{(1)}]$
combine 1	$[C_1^{(1)}],\quad [C_2^{(1)}],\quad [C_3^{(1)}],\quad \ldots,\quad [C_{m_2}^{(1)}]$
summary 2	$[S_1^{(2)}],\quad [S_2^{(2)}],\quad [S_3^{(2)}],\quad \ldots,\quad [S_{m_2}^{(2)}]$
combine 2	$[C_1^{(2)}],\quad [C_2^{(2)}],\quad \ldots,\quad [C_{m_3}^{(2)}]$
⋮	⋮
summary n	$[S_1^{(n)}],\quad [S_2^{(n)}]$
combine n	$[C_1^{(n)}]$

表 3.1: 多層次摘要框架

1. 段落拆分

在多層次摘要演算法中，語言模型會將長文本進行適當的拆分，確保後續摘要生成過程的有效性。拆分的過程通常基於文本的結構特性，例如自然段落或語意邊界，來拆分適合處理的片段。這樣的設計能確保每個段落在生成摘要的過程中保留足夠的關鍵內容，同時也能避免長文本導致模型效能下降的問題。對於拆分的策略，會根據不同語言模型的特性和處理能力進行調整，來優化長文本摘要的結果。

2. 摘要生成

當段落拆分完成後，下一步是對每個段落生成摘要。這個過程目的在於濃縮段落中的關鍵信息，同時保持語意清晰和內容的連貫性。而為了避免一次性壓縮造成的關鍵信息流失，採用了多層次摘要的壓縮方法，分段逐步減少文本的長度。這樣的設計能夠在每次摘要的過程中保留重要內容，達到關鍵訊息和摘要簡潔性的平衡。

總體來說，多層次摘要的方法通過逐步壓縮文本長度，在處理長文本的過程中展現了更強的靈活性與可靠性。不僅能同時兼顧關鍵資訊的完整性也能保持摘要文本的簡潔性。

3. 摘要合併

在完成每個段落的摘要生成後，接下來的步驟是對這些摘要進行逐步的合併，並進行多層次的壓縮，確保關鍵信息被保留下來，避免文本冗長。這個步驟的核心在於如何有效的管理文本長度，使其在後續的摘要壓縮過程中仍能保持核心內容的完整性。

具體來說，合併後的摘要可能仍然超過預期的長度，這時候會將合併的結果視為一個新的文本段落，並重新進行摘要生成與壓縮的步驟。這個循環過程會持續進行，直到文本被壓縮到所需的長度，最終生成一個符合需求的精簡摘要。

這樣的設計在長文本的處理中有著重要作用，尤其是針對有 token 限制的語言模型。在通過多層次的摘要合併與壓縮，能確保每一輪生成的內容既簡潔又保留了核心信息，從而達到高效摘要生成的目標，並避免信息丟失或壓縮不足的情況。

3.2 演算法實現

根據表3.1中的多層次摘要框架，Algorithm1詳細描述了每個步驟的具體實現。在這個演算法中，我們首先將文章拆分為符合 token 限制的段落（Step 1）。這一個步驟能夠將長文章分解成可管理的部分，確保每個段落不會超過模型的處理限制。接下來，對每個文本塊進行摘要生成（Step 2）。這些摘要將被合併成較少的

段落，並重複這個過程直到剩下一個段落，也就是該文本在此演算法中濃縮出的最終摘要。此演算法的核心目的在於每一層的壓縮與合併過程，它有效的平衡了文本長度與信息保留，特別適用於長文本的處理，並能在模型資源有限的情況下達到高效的摘要生成。

Algorithm 1 Multi-level Summarization Algorithm

Input: Article A, Maximum token limit per chunk, Token range for summaries, Target token count

Output: Final summarized article S_{final}

Step 1: Split the article A into chunks $T_1, T_2, \ldots, T_n$ such that each chunk fits within the token limit

while more than one chunk remains **do**

for each chunk T_i **do**

Generate summary S_i for T_i ensuring it stays within a reasonable length range

end for

Merge summaries $S_1, S_2, \ldots, S_n$ into fewer chunks, ensuring the length remains manageable

end while

Step 2: Generate the final summary S_{final} from the remaining chunk

Ensure the final summary is within the desired length range relative to the target token count

return S_{final}

第四章　評估指標

4.1　數據集介紹

本研究使用 CNN/DailyMail 數據集是由 Kyunghyun Cho 和 Alexander Rush 等人於 2015 年發表的論文 [12] 中首次提出的，並說明了其在生成式摘要任務的背景與動機。最初是用於抽取式摘要 (Extractive Summarization) 後來逐漸擴展到生成式摘要 (Abstractive Summarization)。該數據集匯集了來自 CNN 和 DailyMail 兩大新聞網站的文章，用於訓練模型能夠生成新聞文章的摘要。分為三個版本，初版是由”問題-回答”對的形式構成，之後改進為”文章-摘要”對的形式，延續至今，被廣泛應用於新聞理解、文本壓縮等 NLP 任務。

本研究使用 CNN/DailyMail 數據集第三個版本的測試 (test) 集，主要資料內容為：唯一識別碼 (id)、新聞文本 (article) 和新聞摘要 (highlights)，如圖4.1。其中 article 為完整新聞報導的文章，highlights 是針對報導正文撰寫的摘要或關鍵點。


```
{
  id : 062f78c2922d4050190dbba10f5d65eeff25e1ed
  article : Bayern Munich have an interest in Chelsea defender Branislav Ivanovic ...
  highlights : Branislav Ivanovic's contract at Chelsea expires ...
}
```


圖 4.1: CNN/DailyMail 資料集

在此資料集當中，article 的最大 token 數為 3112 tokens，最小 token 數為 66 tokens; highlights 的最大 token 數為 847 tokens，最小為 12 tokens。

其應用場景主要包括文本摘要生成（Text Summarization）、問答系統（Question Answering, QA）、新聞分析與信息抽取（News Analysis and Information Extraction）、生成式預訓練模型的微調（Fine-Tuning Pre-trained Models）和其他自然語言理解中的長文本處理等。

在 Papers with Code 平台上[1]針對 CNN/Daily Mail 資料集，列出了不同研究工作

[1]Papers with Code 平台 (https://paperswithcode.com/sota/document-summarization-on-cnn-daily-mail) 為匯總和展示機器學習領域最新研究成果的網站，提供不同資料集上的基準比較。

上使用 ROUGE 指標進行評估的結果。本文的實驗結果亦可與平台上的相關工作進行比較，以評估本方法的效能。

4.2 ROUGE 評估指標介紹

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) 是一組用於文本摘要和翻譯系統評估的指標，由 Chin-Yew Line 在 2004 年提出 [8]，主要是通過比較機器生成文本 (如摘要或翻譯) 與通常為人工撰寫的參考摘要之間的相似性，來量化摘要的質量。

ROUGE 是基於統計分析，衡量生成文本與參考文本之間內容的重疊程度。這些重疊可以是單詞級別 (詞彙)、短語級別 (n-gram)、句子或段落的語法結構。根據比較範圍的不同，ROUGE 被設計為多種變體 (如 ROUGE-1、ROUGE-2 和 ROUGE-L)，用於評估生成文本在不同細節層次的表現。這個指標是近期自然語言處理領域中極為流行的評估工具，被廣泛應用於新聞摘要、法律文件的重點提取以及機器翻譯結果的評估。ROUGE 的基本公式都圍繞三個核心概念進行計算：

- 精確率 (Precision)：生成摘要中與參考摘要匹配的比例，表示生成文本的相關性。

$$\text{Precison} = \frac{\text{重疊的單位數量}}{\text{生成文本的單位數量}} \tag{4.1}$$

- 召回率 (Recall)：參考摘要中被生成摘要捕捉到的比例，表示生成文本的覆蓋度。

$$\text{Recall} = \frac{\text{重疊的單位數量}}{\text{參考文本的單位總數}} \tag{4.2}$$

- F1-score：Precision 和 Recall 的調和平均數，平衡兩者之間的影響

$$\text{F1-score} = \frac{2 \cdot \text{Precion} \cdot \text{Recall}}{\text{Precision+Recall}} \tag{4.3}$$

Precision 和 Recall 是信息檢索領域中的核心指標，最初用於評估搜索引擎的準確性。這些指標被引入到文本摘要評估中，目的在於平衡摘要的簡潔性與全面性，分別表示為高的精確率和高的召回率。而 F1-score 的引入是為了解決僅考慮 Precision 或 Recall 時可能造成的偏差。其中本研究使用的 ROUGE 變體為：

1. ROUGE-1：捕捉核心內容，適合概括性摘要。

2. ROUGE-2：衡量短語級別的準確性，適用於高語法依賴的生成摘要。

3. ROUGE-L：考慮句子結構整體相似性，對於長文本和抽象式摘要效果較好。

廣義的 ROUGE-n 被定義為任意 n-gram 的匹配，代表連續 n 個單詞組成的片段。其中，最常使用的變體為 ROUGE-1 和 ROUGE-2，分別用於衡量 unigram (單個詞) 和 bigram (相鄰兩個詞) 層級的重疊情況。

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{RefSumm}} \sum_{\text{n-gram} \in S} \text{Count}_{\text{match}}(\text{n-gram})}{\sum_{S \in \text{RefSumm}} \sum_{\text{n-gram} \in S} \text{Count}(\text{n-gram})} \tag{4.4}$$

參數解釋：

- RefSumm (參考摘要)：人工撰寫的參考文本，作為生成摘要質量的基準。
- Count(n-gram)：在參考文本中某個特定 n-gram 的出現次數。
- $\text{Count}_{\text{match}}$(n-gram)：在生成文本與參考文本中共同出現的 n-gram 次數。
- n-gram：由 n 個連續單詞組成的片段。

ROUGE-L 是基於最長公共子序列（Longest Common Subsequence, LCS）[2]的指標，考慮句子整體結構的相似性，LCS 為生成摘要與參考摘要中，按序排列的最長詞序列，允許匹配的詞之間存在間隔，但要求順序需要一致。適合用於評估文本摘要和翻譯結果的語言流暢性，因為 LCS 考慮了詞序的自然連貫性。因此，相較於 ROUGE-N，ROUGE-L 更關注語言的結構性匹配，而非單純的詞頻匹配。

$$\text{ROUGE-L} = \text{F}_\beta = \frac{(1+\beta^2) \cdot P \cdot R}{\beta^2 \cdot P + R}. \tag{4.5}$$

參數符號解釋：

- Precision(P)：在生成文本中匹配到的 LCS 部分佔生成文本的比例。

$$P = \frac{\text{LCS}(X, Y)}{\text{Len}(X)}. \tag{4.6}$$

[2]最長公共子序列（Longest Common Subsequence, LCS）指兩段文本中按順序排列最長子序列，子序列不必連續。

LCS(X, Y)：生成文本 (X) 與參考文本 (Y) 的最長公共子序列長度。

Len(X)：生成文本 (X) 的長度。

- Recall(R)：在參考文本中匹配到的 LCS 部分佔參考文本的比例。

$$R = \frac{\text{LCS}(X, Y)}{\text{Len}(Y)}. \tag{4.7}$$

Len(Y)：參考文本 Y 的長度。

- F 值 (F_β)：將精確率與召回率結合，提供一個平衡的指標。β 是調節精確率與召回率相對重要性的權重，通常設置為 1，表示二者重要程度相同。

此評估指標的數值範圍在 0 到 1 之間，1 表示生成摘要完全覆蓋參考摘要的 n-gram，0 表示沒有任何 n-gram 匹配。

第五章　實驗設計

在面對長文本摘要的挑戰時，本研究目的在探索如何在資源受限的情境下，實現與最先進的大語言模型 (如基於 Transformer Decoder 架構的 GPT 以及結合 Transformer Encoder 與多模態設計的 Gemini 等）相近的摘要效果。我們設計了一套的實驗方法，結合 LLaMA 3.1 模型的文本壓縮能力與 BART 的多層次摘要演算法。

本章節將詳細介紹實驗設計，首先分析 LLaMA 3.1 模型的文本壓縮提示詞設計，目的在於如何移除冗餘內容並同時保留關鍵資訊；接者介紹本實驗的 BART 多層次摘要演算法、參數選取和實現過程。

5.1　LLaMa 3.1 文本壓縮

Prompt 是用戶與大語言模型進行互動的核心工具，分為系統提示詞 (system prompt) 和使用者提示詞 (user prompt)：

- System Prompt（系統提示詞）是由系統預先設置的，定義了 AI 的行為方式、角色身份，以及回答範圍，對使用者來說，System Prompt 通常是不可見的，設置的指令在與使用著互動之前就已經生效，負責引導整體對話框架。
- User Prompt (使用者提示詞）是使用者明確輸入的指令或問題，用來表達目前的需求，是 AI 生成回應的直接依據，靈活性高，可以根據當下的情境調整 Prompt。

為了探討大語言模型在文本壓縮任務中，使用不同提示設計方式和壓縮比設定的效率與精確度差異，我們在實驗中採用了兩種設置方法 (如表5.1)：**ollama.create** 和 **ollama.generate**。這兩種方法分別代表靈活性低與高的模型設置，適用於不同的應用場景，並在壓縮效果和輸出一致性上各有特點，表5.2為兩種設置方式的特性比較：

設置方式	ollama.create (固定系統)	ollama.generate (動態系統)
系統提示詞 (system prompt)	O	X
使用者提示詞 (user prompt)	O	O

表 5.1: 固定系統 vs. 動態系統

設置方式	**ollama.create (固定系統)**	**ollama.generate (動態系統)**
靈活性	低：提示詞與規則固定，所有處理遵循預設設定	高：提示詞可隨需求動態調整
應用場景	適合高度一致性需求，如：法律文件或技術文檔	適合多變的應用場景，如：新聞或社交媒體內容
控制程度	精確控制，保證結果一致	更具彈性，但結果可能略微波動
操作自由度	嚴格：如：僅可刪除詞彙，不允許調整結構或新增內容	自由：如：可調整壓縮比例、文本結構

表 5.2: ollama.create 和 ollama.generate 的比較

5.1.1 固定系統提示詞設計

在本實驗中，我們針對 LLaMA 3.1 模型設計了一個名為 **"compressor"** 的文本壓縮系統，其目標是透過嚴格的提示詞設置，確保文本壓縮結果的穩定性與一致性。此設置方式使用了 **ollama.create** 的方法，將固定提示詞嵌入系統中，使模型在每次處理文本時均能遵循相同的規則。提示詞設計的具體內容基於 Pan 和 Wu 等人的研究 [11]，其設計的主要目標在減少文本詞彙量的同時不改變原文的語意和結構，並針對 LLaMa 3.1 模型的使用狀況增加了第六項指令，限制大語言模型回答的內容僅包含結果，不附加任何註解 (如圖5.1)：

Our compression instructions:

1. You can ONLY remove unimportant words.
2. Do not reorder the original words.
3. Do not change the original words.
4. Do not use abbreviations or emojis.
5. Do not add new words or symbols.
6. Provide ONLY the result text, without any explanations or additional information.

圖 5.1: 固定系統提示詞設計

上述固定提示詞當中，規則 1～ 規則 3 嚴格限制大語言模型的生成功能，只能以原文中的文字在不改變文字次序的條件下做 Extractive Summarization 方式的資料壓縮。

在文本壓縮實驗中，為測試提示詞設計對文本壓縮精度與效率的影響，我們將上述提示詞應用於以下四個資料壓縮任務：compress, highlight, summarize, abstract 指令; 比較哪個提示壓縮的結果更符合我們的目標，以下是這四個指令的特性：

1. Compress (文本壓縮)：減少文本長度，保留最重要的訊息和邏輯結構。壓縮後的文本會比原文短，基本上會保留主要觀點、訊息和原有的句子結構。
2. Highlight (重點標記)：模型會標記出文本中最具信息價值的關鍵句，過程僅移除次要信息，以便在後續處理過程中能夠迅速識別並抽取核心信息，也能使讀者快速瀏覽到重要資訊。
3. **Summarize (摘要)：模型生成針對文本的簡要總結，涵蓋文本的主要內容和重要信息。摘要的過程會去除冗長描述，但保留核心事件和關鍵細節。**
4. **Abstract (概述)：概述是對文章核心思想的高度濃縮，要求模型生成比摘要更高層次的內容總結，通常只有一到兩句，側重主題、目標或重要觀點，而非具體細節，適合快速了解全文意圖。**

在固定提示詞嚴格限制大語言模型生成功能的條件下，上述 Compress (文本壓縮) 或 Highlight (重點標記) 的資料壓縮方式主要在減少文本長度，且生成或保留與文

本主要觀點的關鍵句內容近似的壓縮結果。此外，在此限制條件下預計 Abstract (概述) 將無法對文章核心思想做高度濃縮，有可能會得到與 Compress (文本壓縮) 或 Highlight (重點標記) 類似的結果。至於 Summarize (摘要) 會利用原文文字與結構得到多麼**精簡**的摘要則有待實驗驗證。

在本章中，我們考慮以下兩種壓縮比計算方式對模型壓縮效果的影響：

1. 使用 Token 計算壓縮比：Token 是模型內部的基本處理單位，也是語言模型進行運算和生成的核心單位。因此我們以 token 數作為壓縮比計算基準，理論上較能與模型的內部運作保持一致。
2. 使用 Word 作為壓縮比：Word 是更接近人類日常語言表達的單位，也是常見的篇幅控制標準。在摘要生成或篇幅限制中，對最終文本字數進行限制更為直觀，因此以 word 數作為基準能更好地滿足實際需求。

本實驗中，根據上述提示詞，針對 CNN/DailyMail 資料集分別執行 4 個壓縮任務 (無壓縮比限制條件)，其結果摘錄在表5.3。其中，值得注意的是，用 summarize 指令做壓縮，無論是用 token 或 word 計算的壓縮比平均壓縮率皆在 0.22 左右，遠小於其他三個指令，並且與 CNN/DailyMail 的 highlights 與 article 的平均壓縮比

$$\frac{\text{highligths}}{\text{article}}$$

的值 0.1003 (token)，0.1048 (word) 相近。Highlight 指令得到的兩種壓縮比都是最高，似乎保留較多原 article 的訊息。

task	test item	average token ratio	average word ratio
1	Compress	0.7124	0.6956
2	Highlight	0.8111	0.8068
3	Summarize	0.2196	0.2199
4	Abstract	0.6979	0.6821

表 5.3: 4 個任務的平均壓縮比

這些任務的設計能全面檢驗提示詞設置在不同應用場景中的效果，並為後續評估壓縮準確性和模型表現提供基準。

5.1.2 動態系統使用者提示詞設計

在 Liu 等人 [10] 的研究設計中，提出以壓縮比當作優化參數的策略，來減少冗餘信息並降低記憶體的使用量。然而，Pan 和 Wu 等人 [11] 的研究指出，大語言模型（LLMs）在處理文本壓縮任務時，常難以遵守提示詞中指定的壓縮比，導致輸出文本的長度偏離預期。

Our user prompt:

1. Please summarize the following text to approximately **{ratio}** of the original length.
2. The output should be a direct, concise summary without any extra commentary or introductory text.
3. original text: {...}

圖 5.2: ollama.generate 動態系統提示詞

在5.1.1的實驗中並未對壓縮比進行嚴格限制，因此本小節使用 **ollama.generate** 的方法。依據圖5.2的系統提示詞對給定的壓縮比 (圖5.2中的 **{ratio}**) 進行文本壓縮任務。本小節評估用 token 以及 word 兩種壓縮比計算方式對模型壓縮效果的影響，並針對這兩種壓縮比分別執行兩個壓縮任務。首先是使用 summarize 指令，執行以下任務 task A：

- task A-1：token 的目標壓縮比設為 0.1003。
- task A-2：word 的目標壓縮比設為 0.1048。

這個任務是測試 summarize 指令在設定固定壓縮比的指引下，其結果是否跟 CNN/DailyMail 的 highlights 類似。

接著執行以下任務 task B：

- task B-1：token 的目標壓縮比設為 0.8111。
- task B-2：word 的目標壓縮比設為 0.8068。

這個任務是測試 summarize 指令在設定固定壓縮比的指引下，其結果是否跟 CNN/DailyMail 的 article 類似。

task	test item	target average ratio	results average ratio	R-1	R-2	R-L
A-1	token ratio	0.1003	0.1636	0.3685	0.1424	0.2161
A-2	words ratio	0.1048	0.1714	0.3692	0.1430	0.2165
B-1	token ratio	0.8111	0.2478	0.3574	0.2137	0.2409
B-2	words ratio	0.8068	0.2549	0.3558	0.2129	0.2406

表 5.4: task A 與 task B 壓縮比測試平均結果

task A 的結果顯示 summarize 指令得到的壓縮比較目標壓縮比分別高出 6.33% 和 6.66%，task B 的結果則是遠小於目標壓縮比。其次，表5.4結果顯示，在 task B 中，當目標壓縮比設定得相對較大（例如 0.8 左右）時，模型生成的摘要實際壓縮比卻明顯偏低 (分別為 0.2478 和 0.2549)，結果集中在約 0.2 的範圍。同樣地，在 task A 中，即使目標壓縮比設定得較小（例如 0.1 左右），模型生成的壓縮比也仍然偏向 0.2 附近。

這種現象顯示出此大語言模型的 summarize 指令在執行預設固定壓縮比的任務時，存在顯著的偏差。無論目標壓縮比是高還是低，模型的生成行為均呈現出一種傾向，即壓縮比會集中在一個固定的範圍內。這可能反映出模型對指令中 **summarize** 的解讀方式：模型傾向於將文本壓縮到一個相對較短的長度，甚至超出使用者指令的實際需求。

這些結果表明，儘管使用者提示詞明確要求模型根據目標壓縮比進行壓縮，但模型生成的摘要無法準確遵守提示，並且表現出一種對壓縮比的內建偏好。而造成此現象的原因可能包括：

- 大語言模型內部對 **summarize** 指令的固有傾向，即模型偏好生成較短的文本。
- 提示詞本身可能缺乏足夠的約束力，導致模型未能完全理解或執行目標壓縮比的要求。

圖5.3 進一步展現了這些偏差的分布。對於 task A 的壓縮比（A-1 與 A-2），紅色柱狀圖代表的實際生成壓縮比分布明顯右偏，代表模型傾向於生成較長文本，出現與藍色柱狀圖代表的目標壓縮比還要高的現象，由於 task A 的限制目標壓縮比主要集中在 0.1 附近，因此我們僅繪製壓縮比在 0 到 0.5 範圍內的結果。而在 task B 的壓縮比（B-1 與 B-2）中，紅色柱狀圖則明顯左偏，代表模型壓縮過度且未能達到目標值。此外值得注意的是，在 task B 實驗中有極少數 (不到 10 個) 的文本壓縮比超過 1 ，因此在動態系統中此大語言模型並未遵循壓縮比的限制。

結合表5.4和圖5.3的結果，可以得出與 Pan 和 Wu 等人 [11] 對於壓縮比相關探討的觀點一致，大語言模型（LLMs）在壓縮比上的表現存在不穩定性，難以精確遵守提示詞中指定的壓縮比限制，導致生成文本長度與目標值之間存在顯著偏差。

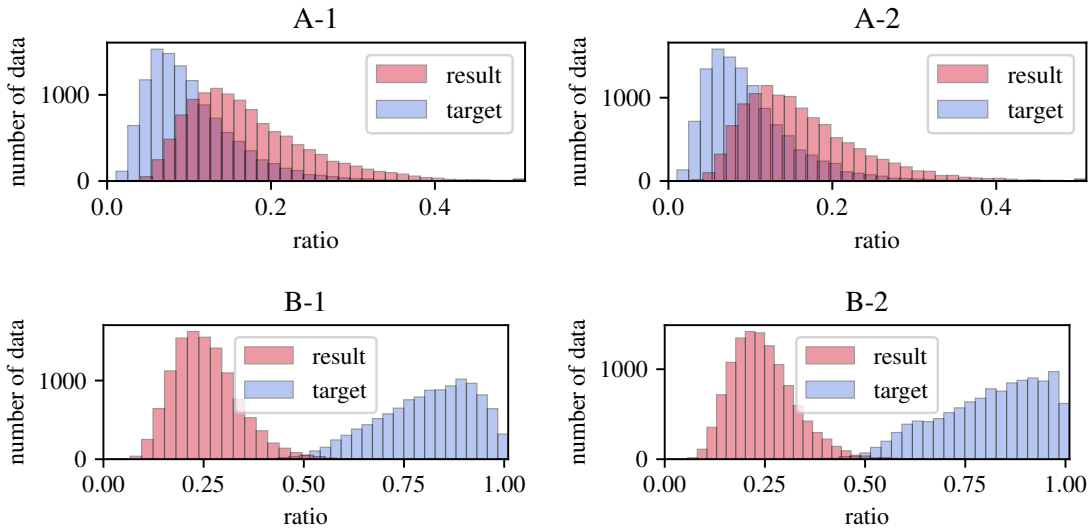

圖 5.3: 4 個任務對應的壓縮比測試比較結果：藍色代表限制每一篇文本的壓縮比，紅色代表實際上生成出來摘要結果的壓縮比。縱軸為資料的總數，橫軸為壓縮比的比率。

5.2 BART 多層次摘要

本研究的多層次摘要演算法使用 BART 作為提取摘要的模型，具體選擇用 facebook/bart-large-cnn 模型來生成高效且精確的摘要，本論文使用 spaCy 語意分割模型進行段落的分割，根據句子的結束符號來判斷句子的邊界，對於文本裡容易遇到的人名、縮寫或是數字等特殊情況，spaCy 模型能夠準確的判斷哪些並非句子的結

尾，根據上下文的語義做精準的分割，保留完整語義。

5.2.1 參數的選取

參數的選取對本研究的多層次摘要演算法結果至關重要。為了確保最終摘要的精確性與文本長度，選擇適當的參數對於摘要生成的質量有直接的影響。

雖然 BART 模型的上限為 1024 tokens，為了避免注意力機制分散的問題，我們設定 512 tokens 為輸入 token 上限。每個段落的最小和最大長度的設定，我們選擇了將每段摘要長度控制在 230 到 256 tokens 之間。這個範圍的選擇是基於對 BART 模型生成摘要結果的觀察。根據實驗結果，BART 模型通常會生成接近最小長度的摘要，因此為了不過度壓縮關鍵信息，將摘要的最小長度設定為目標 token 數量的 90%（即 230 tokens），而最大長度則是 256 tokens，即原始目標的 100%。這樣的長度範圍能夠在壓縮文本的同時，保留住關鍵信息，避免重要內容的丟失。

在最終的摘要長度控制中，為了確保摘要長度的合理性，我們參考了 CNN/DailyMail 資料集中的 highlight token 作為基準。該資料集的 highlight 是專門針對新聞文章設計的，長度限制在信息摘要領域具有極高的代表性。因此，我們設置了目標長度為資料集中 highlight 部分的 90% 到 120% 之間。這個長度控制是整個多層次摘要過程中最關鍵的一個步驟。由於我們使用的相似度評估是通過 ROUGE（Recall-Oriented Understudy for Gisting Evaluation）來進行的，該指標專門用來比較原始摘要與生成摘要之間的相似度，包括字數和內容的匹配度。因此，為了使生成的摘要與原始的 highlight 長度差異不要過大，確保生成的摘要在評估指標上達到理想效果，因此設計了這樣的限制條件。

我們為了適應 CNN/DailyMail 資料集的特性以及 ROUGE 評估指標的要求，特別設計了一個額外的 final summary 步驟。該步驟的引入目的在於使最終生成的摘要更接近理想的長度與語義範圍，提升在 ROUGE 指標上的表現。

Algorithm 2. BART 多層次摘要演算法

Input: Article A, Maximum tokens per chunk $T_{\text{max}} = 512$, Token range for summaries $S_{\text{min}} = 256 \times 0.9$, $S_{\text{max}} = 256$, Target token count T_{target}
Output: Final summarized article S_{final}

Step 1: Split the article A into chunks $T_1, T_2, \ldots, T_n$ such that each $|T_i| \leq T_{\text{max}}$
while $n > 1$ **do**
    for each chunk T_i **do**
        Generate summary S_i for T_i using the BART model
        Ensure the token count of each summary is within the range: $S_{\text{min}} \leq |S_i| \leq S_{\text{max}}$
    end for
    Merge summaries $S_1, S_2, \ldots, S_n$ into fewer chunks $C_1, C_2, \ldots$, ensuring token count $|C_i| \leq T_{\text{max}}$
end while
Step 2: Generate final summary S_{final} from the remaining chunk
Ensure $T_{\text{target}} \leq |S_{\text{final}}| \leq 1.2 \times T_{\text{target}}$
return S_{final}

第六章　實驗結果

6.1　軟/硬體環境設置

本實驗的電腦軟體與硬體設置如下：

處理器 (CPU)	Intel Core i7-14700 @ 2.10 GHz
顯示卡 (GPU)	NVIDIA GeForce RTX 4090 (24GB)
主記憶體 (RAM)	64 GB
作業系統	Microsoft windows 11 Pro
軟體版本	Python 3.12.7 + CUDA 12.4

6.2　實驗設計

6.2.1　單獨模型實驗的觀察

本研究使用兩種模型進行文本摘要的任務，分別測試 LLaMA 3.1 模型與 BART 多層次摘要演算法在摘要任務中的效能結果，其中，LLaMA 3.1 使用了圖5.1的系統提示詞，在使用者提示詞的部分命令其進行摘要 (不設定壓縮比)，而 BART 使用了多層次摘要演算法。

Model		R-1	R-2	R-L
1.	LLaMA 3.1	0.3350	0.1280	0.2054
2.	**BART+ 多層次摘要**	**0.4754**	**0.2334**	**0.4150**

表 6.1: 單一模型摘要性能評估

表6.1測試中使用 LLaMA 3.1 模型進行摘要生成花費時間約為 5 小時 7 分鐘，在效率上具一定優勢。但結果顯示了 LLaMa 3.1 的 ROUGE 分數顯著較低，特別是在 ROUGE-2 和 ROUGE-L 上的表現不如預期。這可能是由於此大語言模型在生成

摘要時傾向於加入額外推斷或偏離原文的內容，導致生成的摘要長度與目標 highlighlts 的長度差異較大。此外，LLaMA 3.1 的摘要生成雖然受益於較大的 token 上限 (128K)，但並未充分發揮其對長文本的處理能力，因此與其設計目標存在了很大的偏差。

相較之下，BART 多層次摘要演算法 (預設 token 上限為 512) 在三項 ROUGE 指標上的表現均優於 LLaMA 3.1，特別是在 ROUGE-1 和 ROUGE-L 上展現了更高的摘要質量。表示 BART 的多層次摘要在信息提取與語義表達方面更精確且忠實於原文，適合作為高品質摘要生成的工具，但缺點是其花費時間高達 39 小時 40 分鐘 (如圖 6.2所示) 會花費較多的時間，資源的消耗非常龐大。

基於以上的觀察，我們進一步設計了改進的實驗方法，解決效率與精確度間的平衡問題，結合兩個模型的優勢，構建分階段處理的流程，將實驗分為兩個階段，文本壓縮和 BART 多層次摘要。

6.2.2 實驗一：文本壓縮

此階段使用圖5.1系統提示詞，結合不同的壓縮指令對長文本進行預處理完成初步壓縮，為後續的 BART 多層次摘要減輕負擔。表 6.2 為 LLaMA 3.1 在此階段針對表 5.3中的四個不同壓縮任務（Abstract、Highlight、Summarize、Compress）所取得的 ROUGE 分數結果。由於此階段目的是希望將文本中冗餘的資訊做一些初步的過濾，因此壓縮的結果將會和原始文本 (article) 做相似度比對。

task		R-1	R-2	R-L
I.	compress	0.8271	0.6775	0.8237
II.	highlight	0.8906	0.7501	0.8900
III.	**summarize**	**0.3415**	**0.2241**	**0.2803**
IV.	abstract	0.8117	0.7228	0.8031

表 6.2: 壓縮任務與原始資料 article 的 ROUGE 比較

從表6.2來看，可以知道 Summarize (task III.) 的 ROUGE 分數顯著偏低，這個結果反映了 LLaMA 3.1 在執行「摘要 (Summarize)」的生成行為。

summarize 會讓模型生成過於簡短的內容，導致壓縮比例過高的問題(如圖 6.1)。在本研究實驗設計5.1.1章節中也有提到，「summarize」這個指令，本身的語意會引導大語言模型偏向簡化生成，而忽略了原文的細節或部分的語意匹配，因此這可能是造成 ROUGE 分數的表現不理想的主要原因。

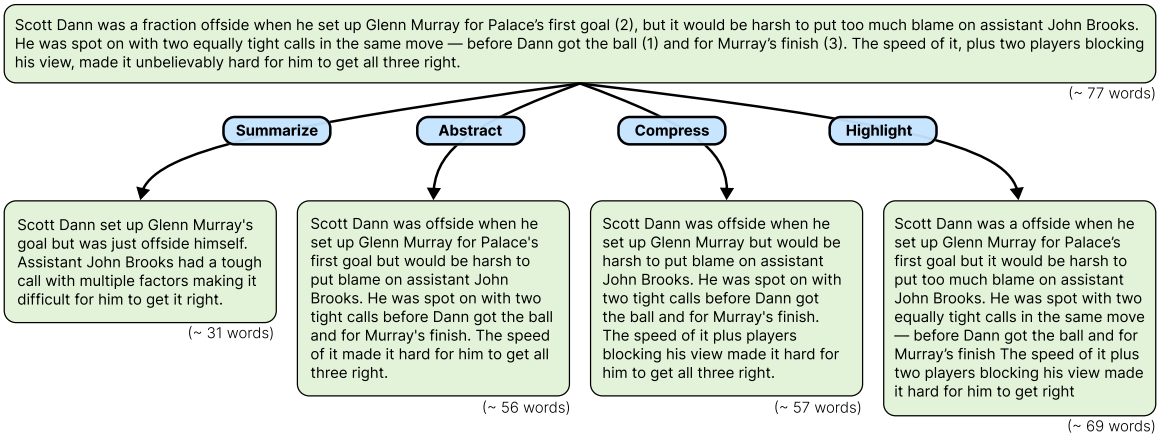

圖 6.1: 文本壓縮結果示意圖

相較之下，於 Compress (task I.) 的 ROUGE 指標相對均衡。而 Abstract (task IV.) 和 Highlight (task II.) 在此壓縮的任務上表現突出，特別是在 ROUGE-1 和 ROUGE-L 均有很不錯的分數，表示模型在執行這兩項任務時，均能夠有效捕捉文本中的重要資訊。其中，Highlight 指令的優秀表現，原因可能在於其任務設計為聚焦於標記文本中的重要資訊，使模型生成內容的完整性較高，因此讓生成的壓縮文本與 article 的語意匹配度高。

綜合本階段的實驗結果，儘管四項任務的 ROUGE 分數有所差異，但它們共同反映了模型在不同壓縮目標下的行為特性。這樣的提示詞設計與不同的壓縮指令能讓下一個實驗提供多樣化的輸入基礎，也能進一步探討不同壓縮方式對最終生成摘要質量的影響。

6.2.3 實驗二：BART 多層次摘要

在完成第一階段的壓縮文本預處理後，本研究將 LLaMA 3.1 生成的壓縮結果作為輸入，交由 BART 模型進行多層次摘要處理，以節省計算時間。將第一階段的壓縮文本作為基礎，第二階段的目標在於進一步提取文本核心的資訊，同時確保摘要

的語意完整性與結構連貫性。在此階段中，我們根據 BART 模型的特性以及所使用的資料集，調整參數並優化框架，這些多層次摘要的結果摘錄在表 6.3 中。

從表6.3來看，summarize+bart (task VII.) 的結果顯示，雖然在實驗一中，因為生成文本過於簡短導致資訊流失，造成表6.2中的任務分數明顯不理想，但透過多層次摘要的結合，其 ROUGE 分數有一定程度的提升。這可能是因為多層次摘要的演算法具備更強的語意提取能力，也發揮了模型的忠實性，因此對原始文本的語意提取能力較強，在後續的摘要階段有效彌補前期壓縮過程中因資訊缺失而導致的語意偏離問題，進而改善了整體文本的品質。

task		R-1	R-2	R-3
V.	compress+bart	0.4217	0.1902	0.3750
VI.	**highlight+bart**	**0.4395**	**0.2002**	**0.3886**
VII.	summarize+bart	0.3781	0.1464	0.2495
VIII.	abstract+bart	0.4344	0.2019	0.3835

表 6.3: 摘要結果與 CNN/DailyMail highligths 的 ROUGE 比較

值得關注的是，highlight + bart (task VI.) 的結果表現特別突出，顯示相較於其他指令，highlight 的方法在保留文本語意與結構完整性方面具有明顯優勢。同時，其餘兩個任務 compress + bart (task V.）和 abstract + bart (task VIII.) 的結果也表現良好，但相比之下，task VI. 在分數上的優勢更為顯著。雖然此任務的分數仍然低於表6.1 中 BART 模型單獨摘要實驗（task 2）的評估分數，平均分數大約相差了 3%，但在花費的資源上卻有顯著改善，將文本處理與摘要生成的時間大幅縮短至約 30 小時 24 分鐘（如圖 6.2 所示），大大縮短了文本處理與摘要生成所需的時間。

綜合本研究的實驗結果，我們通過結合 LLaMA 3.1 壓縮模型與 BART 多層次摘要模型，提出了一種高效的長文本摘要方法。不僅顯著提升了摘要生成的準確性，還成功解決了單一模型在長文本處理中面臨的效率與精確度挑戰。這種方法成功驗證了多層次處理框架的能力，為未來的長文本處理研究提供了一個參考方向。

此兩階段的實驗，由於實驗一中的文本預處理壓縮，使得實驗二的 BART 多層次壓縮可以在較精闢的文本上進行運算，顯著降低了處理時間與資源需求。我們結

合了 LLaMA 3.1 與 BART 的優勢，有效的平衡了在單一模型在使用中面臨的資源消耗和摘要精確度之間的問題，同時也解決了大語言模型在處理長文本時，會面對的注意力機制分散問題，展現了此方法在實際應用中的潛在價值。

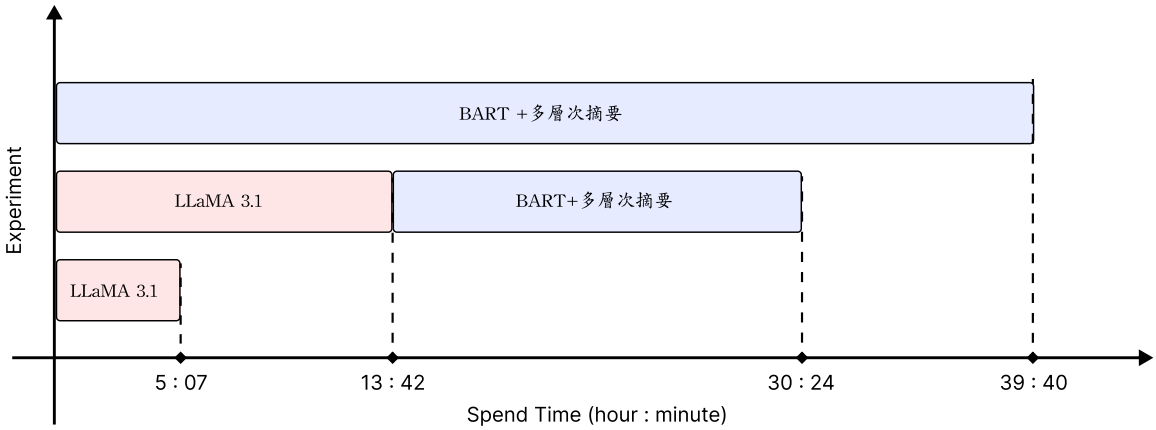

圖 6.2: spend-time 比較

第七章　結論與未來展望

7.1　結論

本研究針對長文本摘要中常見的注意力機制分散的問題，提出一種兩階段的多層次摘要演算法：首先，利用 LLaMA 3.1 模型對長文本進行初步壓縮，以減少冗餘的詞彙；接著，使用 BART 模型進行多層次摘要 (限制輸入上限為 512 tokens)，逐步提取出語義完整且準確的摘要。這種設計不僅能有效處理長文本，還能避免大語言模型在自動調整壓縮比時忽略關鍵內容，使其保持摘要的語義完整性與品質。

在 CNN/Daily Mail 資料集上的實驗結果顯示，我們的方法在 ROUGE-1(表7.1)、ROUGE-2(表7.2) 和 ROUGE-L(表7.3) 指標上[1]均取得了不錯的表現。我們也將該方法與目前在 Papers with Code 平台上公開的摘要結果進行比較，發現我們的方法不僅在效能和資源效率之間達成了良好的平衡，也展現出多層次摘要在長文本處理中的穩定性與競爭力。

Rank	Model	ROUGE-1
1.	Scrambled code + broken (alter)	48.18
2.	**BART 多層次摘要**	**47.54**
3.	PEGASUS + SummaReranker	47.16
⋮	⋮	⋮
8.	MatchSum (BERT-base)	44.22
9.	**LLaMA 3.1 + BART 多層次摘要**	**43.95**
10.	BertSumExt	43.85
⋮	⋮	⋮

表 7.1: **ROUGE-1** in Papers with Code

Rank	Model	ROUGE-2
1.	**BART 多層次摘要**	**23.34**
2.	PEGASUS + SummaReranker	22.55
3.	Fourier Transformer	21.55
⋮	⋮	⋮
12.	BERTSUM+Transformer	20.24
13.	**LLaMA 3.1 + BART 多層次摘要**	**20.02**
14.	Scrambled code + broken (alter)	19.84
⋮	⋮	⋮

表 7.2: **ROUGE-2** in Papers with Code

另外，除了量化指標的提升外，該框架能根據不同需求調整壓縮與摘要的層次，在實際應用上有很多的靈活性。例如，用於處理法律文件或醫療記錄等專業領域中的長文本摘要，由於多層次分段處理策略，該框架能在低資源環境下高校運

[1] 表7.1、表7.2 和表7.3 中，黑色粗體標記代表本研究中實驗表現最佳的 task VI (highlighted + BART) 的 ROUGE 評估分數；紅色粗體標記則為僅使用 BART 多層次摘要演算法的 ROUGE 評估分數。本網站僅顯示使用 CNN/DailyMail 做摘要任務的前 26 名模型分數。

行，因此適合在計算資源有限的環境。

Rank	Model	ROUGE-L
1.	Scrambled code + broken (alter)	45.35
2.	PEGASUS + SummaReranker	43.87
3.	HAT-BART	41.52
4.	**BART 多層次摘要**	**41.50**
⋮	⋮	⋮
13.	BERTSUM+Transformer	39.63
14.	**LLaMA 3.1 + BART 多層次摘要**	**38.86**
15.	Selector+Pointer Generator	38.79
⋮	⋮	⋮

表 7.3: **ROUGE-L** in Papers with Code

7.2 未來展望

本論文採用了 ROUGE 作為主要的評估指標，但 ROUGE 的計算主要是基於詞語的重疊率，可能沒辦法完全反映生成摘要在語義理解上的表現。未來，我們可以加入更多不同的評估方式，例如 BLEU、METEOR 或是基於語義匹配的指標，來更全面的評估摘要生成的品質。

雖然我們目前是用 BART 模型結合多層次摘要演算法，但未來可以考慮使用其他更複雜的大語言模型作多層次摘要，甚至將方法擴展到多模態資料，例如結合圖像、表格與文本等，應用在需要多樣資訊處理的情境中。也可以將本研究的成果應用於法律、醫療等需要處理長文本的專業領域，進一步驗證其通用性與穩定性。

本論文是採用限制輸入 token 上限的方式結合多層次摘要演算法來改進處理長文本的注意力偏差問題，在 CNN/DailyMail 資料集上的實驗初步測試顯示，這種設計能夠在處理長文本時保持摘要的語意完整性，然而它是否能夠完全避免因位置偏差而更傾向於關注文本開頭與結尾忽略中間內容的現象，仍需要進一步測試諸如 Liu 等人 [9] 等人的 NaturalQuestions-Open 數據集來驗證。

參考文獻

[1] Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. Walking down the memory maze: Beyond context limit through interactive reading, 2023.

[2] Junqing He, Kunhao Pan, Xiaoqun Dong, Zhuoyang Song, Yibo Liu, Qianguo Sun, Yuxin Liang, Hao Wang, Enming Zhang, and Jiaxing Zhang. Never lost in the middle: Mastering long-context question answering with position-agnostic decompositional training, 2024.

[3] Cheng-Yu Hsieh, Yung-Sung Chuang, Chun-Liang Li, Zifeng Wang, Long T. Le, Abhishek Kumar, James Glass, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Found in the middle: Calibrating positional attention bias improves long context utilization, 2024.

[4] Xijie Huang, Li Lyna Zhang, Kwang-Ting Cheng, Fan Yang, and Mao Yang. Fewer is more: Boosting llm reasoning with reinforced context pruning, 2024.

[5] Maor Ivgi, Uri Shaham, and Jonathan Berant. Efficient long-text understanding with short-text models, 2022.

[6] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore, December 2023. Association for Computational Linguistics.

[7] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[8] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[9] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[10] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[11] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression, 2024.

[12] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization, 2015.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[14] Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. Retrieval meets long context large language models, 2024.

[15] 郭子浩, 孙由之, 张梦林, 王欣然, and 陈雨洁. 大语言模型背景下提示词工程赋能英语口语学习研究, 2023.