Mamba 與 Transformer 在文字分類上之比較
Comparing Mamba and Transformer on Text Classification

陳御輔
資訊工程學系
國立暨南國際大學
南投,臺灣
s111321031@mail1.ncnu.edu.tw

許祐銓
資訊工程學系
國立暨南國際大學
南投,台灣
s111321027@mail1.ncnu.edu.tw

Abstract

在當前 AI 技術蓬勃發展的時代，從資料蒐集、前處理到最終模型輸出，相關技術層出不窮；而各式 AI 架構亦在短短數十年間不斷演進，從早期的 CNN、RNN，到如今國際大廠廣泛採用的 Transformer，以及近年崛起，專為 long-range dependency 所設計的 Mamba 模型。經過對 Mamba 架構的了解過後，以兩項不同的文字分類任務對 Mamba 和 Transformer 在模型參數量大約相差 10%以內對訓練成本和效能進行比較，發現了 Mamba 雖然在 accuracy 的表現上略微不及 Transformer ，但在 GPU 記憶體的使用程度上最多能降低 73.3%，訓練時間上則最低能有 15.1%的縮短，推理時間上最低能夠有 7.6%的減少。清楚地展現了 Mamba 相較於 Transformer 有更低的訓練資源消耗以及更快的訓練與推理速度。

Keywords—Mamba, Transformer, Text Classification

I. INTRODUCTION

在 AI 的領域中，有各式各樣的模型因為各種不同的目的或是任務性質而被開發出來。例如針對圖像相關任務而出現的 CNN (Convolution Neural Network) ，為了序列型資料而開發的 RNN (Recurrent Neural Network) 。近年來各種 AI 工具也不斷地出現在我們的生活中，如 ChatGPT、Gemini 等等的聊天機器人，Stable Diffusion、DALL·E 等等的圖像生成模型。

而 Transformer [1]自 2017 年推出以來，已然成為現在世界上各種大型語言模型的基本架構。隨著人類對 AI 模型的期待日益增加，Transformer 雖說仍然是 Seq2Seq 架構中絕對的霸主，但隨著序列長度增加而快速暴漲的資源消耗已經不可忽視。

Mamba 作為 Transformer 的挑戰者，更是從架構上根本的解決了 Transformer 因為序列長度而產生的資源消耗問題。透過對 Mamba 架構的研究與了解，我們嘗試透過架構的比較與實驗數據解釋為何 Mamba 能夠成為 Transformer 的挑戰者。

II. MAMBA ARCHITECHTURE

Mamba [2]是一種 Seq2Seq 的模型架構，由 SSM 和 Selective 機制所組成的模型架構。Mamba 架構的核心概念是期望在有限的矩陣大小內，盡可能地保存重要的資訊，並將不重要的資訊遺忘。高效利用記憶體再加上 SSM 所提供的平行化訓練機制與線性推理機制，使的 Mamba 具備足以成為 Transformer 挑戰者的資格。

A. SSM

SSM (State Space Model) 的概念來自訊號系統中的 Continuous-Time Linear State-Space ODE 。具體公式如下： **x**(t)：系統狀態；A、B、C、D：系統參數矩陣；u(t)：訊號輸入；**y**(t)：狀態輸出

$$\frac{d\mathbf{x}(t)}{dt} = A\mathbf{x}(t) + B\,u(t) \tag{1}$$

$$y(t) = C\mathbf{x}(t) + D\,u(t) \tag{2}$$

為了將這個描述連續狀態的微分方程改為描述一個離散狀態的公式，引入了 step size Δ 作為離散化參數。並且透過 Bilinear Transform 對 Eq.(1) 的 A 和 B 進行離散化，公式如下： A_d：離散化 A；B_d：離散化 B

$$\mathrm{A_d} = \left(\mathrm{I} - \frac{\Delta}{2}\mathrm{A}\right)^{-1}\left(\mathrm{I} + \frac{\Delta}{2}\mathrm{A}\right) \tag{3}$$

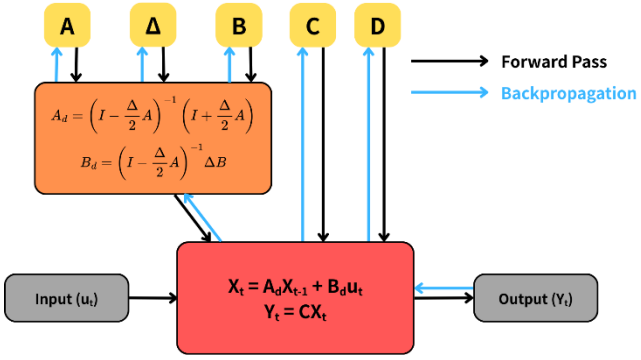
$$B_d = \left(I - \frac{\Delta}{2}A\right)^{-1}\ \Delta\, B \tag{4}$$

除了將 A 和 B 進行離散化處理之外，系統狀態 x(t) 也會變成離散化的狀態，將不再需要如 Eq.(1) 使用微分的形式進行表示。而系統參數矩陣 D 則相當於 Residual 的功能，對 SSM 的系統狀態並沒有影響，所以將 D 從系統參數矩陣中移除，至此 SSM 的核心公式如下：

$$\mathbf{x_t} = \mathrm{A_d}\mathbf{x_{t-1}} + B_d\, u_t \tag{5}$$

$$\mathbf{y_t} = C\mathbf{x_t} \tag{6}$$

在 SSM 的核心公式 Eq.(5)、(6) 中的 A_d 和 B_d 並非是 SSM 的訓練目標，SSM 中可訓練的參數是 A、B、C、D、step size Δ 。關於 SSM 如何處理資料以及訓練時是如何執行 backpropagation (藍色箭頭) 如圖一所示，黑色箭頭則表示正常進行 forward pass 時，系統參數矩陣 A、B、C、D、step size Δ 的運作方式。



圖一：SSM 架構

除了上述的基本 SSM 概念之外，SSM 在訓練時具有特殊的 CNN-like 計算方式，以及實踐階段避免系統參數矩陣 A 的不穩定而使用的特殊初始化矩陣 HiPPO matrix [3]，還有針對計算不穩定而提出的一系列數學解法。

1) CNN-like

從 SSM 的核心公式 Eq. (5)、(6)不難發現 SSM 是一個 RNN-like 的模型，而 RNN-like 模型最大的問題便是訓練速度緩慢，但 SSM 與一般 RNN 模型最大的不同便是 SSM 具有 CNN-like 的訓練方式。

假設將 x_{-1} 設為 0，並將 Eq. (5)、(6)攤開，可以得到結果如下：

$$\mathbf{x_0} = B_d\, u_0$$
$$\mathbf{y_0} = CB_d\, u_0$$

$$\mathbf{x_1} = \mathrm{A_d}B_d\, u_0 + B_d\, u_1$$
$$\mathbf{y_1} = \mathrm{CA_d}B_d\, u_0 + CB_d\, u_1$$

$$\mathbf{x_2} = \mathrm{A_d}^2 B_d\, u_0 + \mathrm{A_d}B_d\, u_1 + B_d\, u_2$$
$$\mathbf{y_2} = \mathrm{CA_d}^2 B_d\, u_0 + \mathrm{CA_d}B_d\, u_1 + CB_d\, u_2$$

透過上面的式子，能夠整理出下面的規律：

$$\mathbf{y_t} = \mathrm{CA_d}^t B_d\, u_0 + \mathrm{CA_d}^{t-1} B_d\, u_1 + \cdots + \mathrm{CA_d}B_d\, u_{t-1} + CB_d\, u_t \tag{7}$$

定義一個 $\boldsymbol{K}$ 作為 kernel，則得出：

$$\boldsymbol{K_t} = (\mathrm{CA_d}^t B_d\,,\ \cdots\ ,\ \mathrm{CA_d}B_d\,,\ CB_d) \tag{8}$$

再將 $\boldsymbol{K_t}$ 與 Eq.(7) 進行公式整理之後便會得到下面的式子：

$$\mathbf{y_t} = \boldsymbol{K_t} \times \mathrm{U_t} \tag{9}$$

U_t 代表$[u_1, u_2, \ldots, u_t]$，對於輸入 token 數量為 t 時，能夠直接根據 A_d、B_d、C、t 計算出 $\boldsymbol{K_t}$，再將 Ut 中的每個 token 對應到 $\boldsymbol{K_t}$ 的每一項。因為 kernel $\boldsymbol{K_t}$ 的存在，賦予了 SSM CNN-like 的特性，不論 t 的大小都能用一個公式直接計算出 SSM 的輸出 $\mathbf{y_t}$，大大降低了訓練所需的時間。

2) HiPPO Matrix

從 Eq. (3)、(5)中可知系統參數矩陣 A 會負責處理系統狀態 $\mathbf{x_t}$ 的保留。透過 Eq.(6) 可知 SSM 的輸出 $\mathbf{y_t}$ 與系統參數矩陣有密切的關係，因此系統參數矩陣 A 可以說是決定狀態系統 **x** 記憶長短與穩定的關鍵。

為了保證系統狀態 $\mathbf{x_t}$ 的穩定，引入了 HiPPO matrix 對系統參數矩陣 A 進行初始化。HiPPO matrix 透過 Legendre 多項式的係數對矩陣進行設定，進而達到讓系統參數矩陣 A 從訓練的一開始便具有記憶能力，維持系統狀態 $\mathbf{x_t}$ 的穩定。

3) Calculation Issue

在離散化的過程中 Eq. (3)、(4) 出現了反矩陣的運算，再加上 Eq. (7) 中系統參數矩陣 A 的多次連乘都會導致計算上的不穩定，尤其在大量的運算上容易出現錯誤，致使整個系統不穩定，輸出結果也會無法信任。

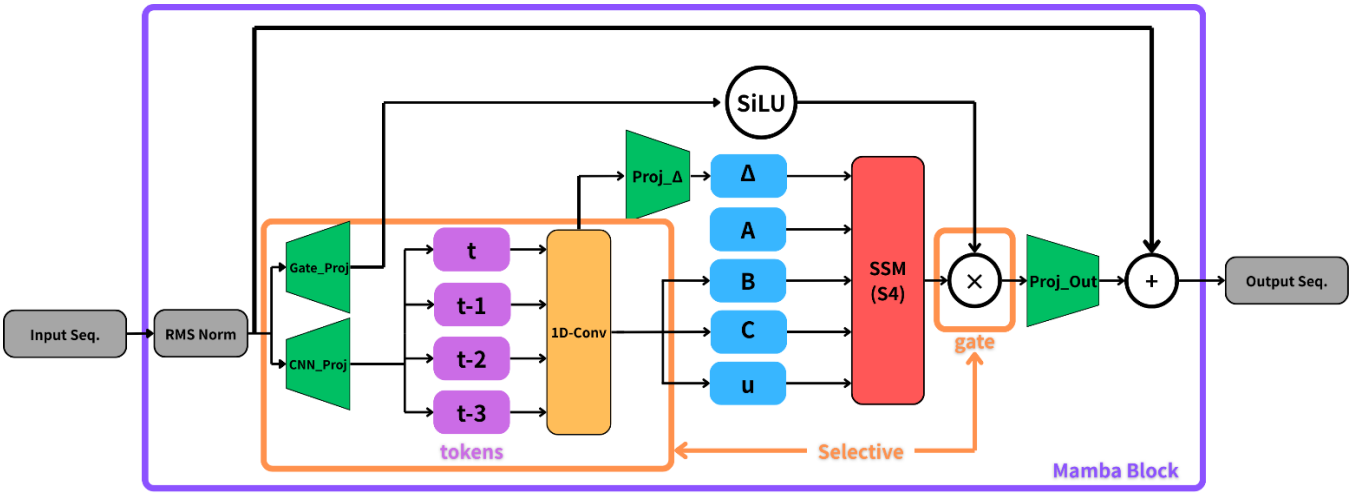
在 Structured State Space Model [4]中給出了解決辦法，透過專門為 SSM 設計的計算演算法，包含 iFFT, NPLR, DPLR ,Woodbury Identity, Cauchy kernel, FFT 保證了計算的穩定性。

B. *Selective*

即使在 Structured State Space Model(S4) 的研究中已經解決了 SSM 在實作上的問題，但仍然不足以讓 SSM 成為 Transformer 的挑戰者，因為 SSM 並沒有像 Transformer 中的 Attention 機制能對輸入進行篩選。

相較於 Attention 直接考慮整個序列，並挑選出重要資訊進行加權的做法， Selective 的做法是對輸入的資訊進行挑選並記憶，進而達到在有限的系統狀態 $\mathbf{x_t}$ 內盡可能儲存重要資訊。具體 Selective 的作法如圖二所示，由三個部分組成，分別是映射矩陣、1D-Convolution、gated 機制 。

映射矩陣是兩個可學習的矩陣，一個負責將 input 投影成 gate 所需的資訊(如圖二中的 Gate_Proj)，另一個矩陣則負責將 input 投影成後續 1D-Convolution 所需的矩陣(如圖二中的 CNN_Proj)。



圖二：Mamba 架構

1D-Convolution 中的 1D 是指在一段序列中每個 token 的每個維度所出現的順序或者說時間關係，如圖二中 token 的 t, t-1, t-2, t-3 的關係。而 Convolution 的輸入則是來自 In_Proj 的映射結果。具體這個 1D-Convolution 的 window 大小則作為超參數用以根據目標與任務進行調整。而 1D-Convolution 的輸出則會交由 SSM 進行後續運算。

在 SSM 輸出之後會進入一個 gate，而這個 gate 會決定 SSM 的輸出 $\mathbf{y}_t$ 中每個元素應該保留或是遺忘的程度。而 gate 中決定保留或是遺忘的程度的矩陣由 Gate_Proj 經過 SiLU activation function 所產生。

C. *Mamba Block*

主要由 Selective 和 SSM 所組成，但並不完整，仍需要一點調整與技術才能完整實現整個 Mamba Block (如圖二所示)

以 RMS Norm (Root Mean Square Layer Normalization) 對輸入序列進行 normalization，不僅能契合 SSM 的性質，相較其他 normalization 也有更高的計算效率，對於計算量龐大的 Mamba 來說非常適合。

接著是 1D-Convolution 和 SSM 之間的對接。已知 SSM 需要的輸入包括 A、B、C、u、Δ，而 1D-Convolution 的輸出結果會切割成 B、C、u 傳進 SSM，A 則是由 HiPPO matrix 直接進行初始化，最後剩下的 Δ 則是將 1D-Convolution 的輸出結果透過一個可學習的映射矩陣 Proj_Δ (如圖二所示) 投影得到。

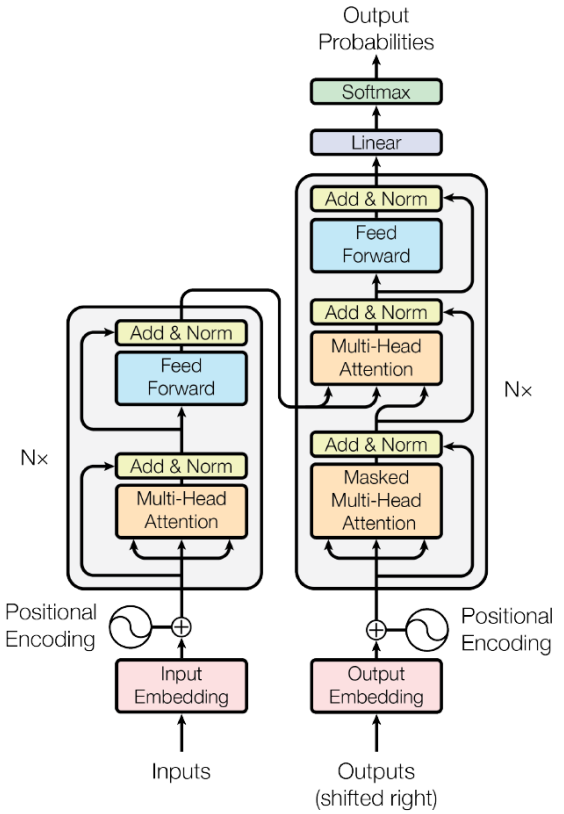
在 SSM 完成計算輸出 $\mathbf{y}_t$ 並經過 gated 計算之後由 Proj_Out 進行維度調整再加上 input sequence 提供 Residual 機制。經過上述的組合才能夠完整的呈現一個如圖二所示的 Mamba Block。

III. Comparison of Mamba and Transformer

Transformer 與 Mamba 同樣都是 Seq2Seq 的模型，不過 Transformer 由 Encoder 和 Decoder 組成，如圖三所示。Transformer 主要由 Multi-head Self-Attention, Feed Forward Network, Residual, Layer Normalization 所組成，並且以其中的 Multi-head Self-Attention 為主要核心。

Multi-head Self-Attention 由多個 Self-Attention 組成，而 Self-Attention 機制是透過三個矩陣 (如圖四中的 Q、K、V) 計算出每個 token 和其他每個 token 之間的關係 (如圖四中的 attention score)。透過 attention score 能夠找到帶有重要或者特殊資訊的 token。而 Multi-head 則代表了需要對那些資訊進行 attention。

在後續的比較中會以 Mamba 和 Transformer Encoder 做為比較對象，但由於差距最為明顯的部分是各自的核心機制，所以後續的比較會著重在 Selective SSM 和 Self-Attention 之間的差異。



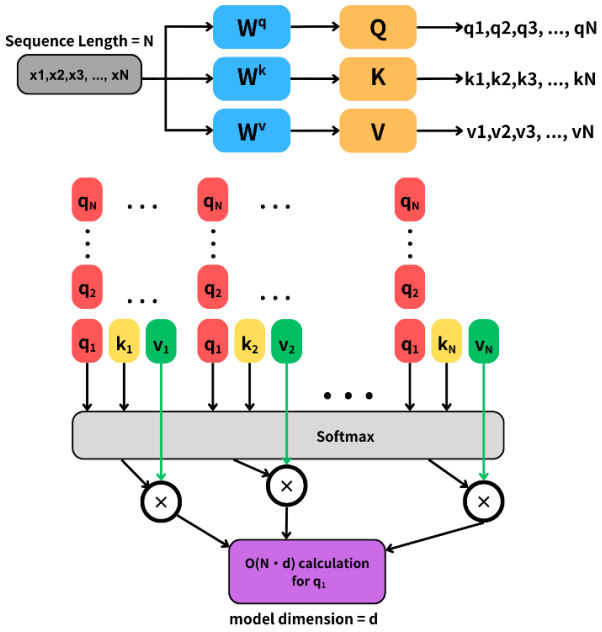
圖三：Transformer 架構 (圖片來源：[1] Attention is All you Need)

A. *Mechanism Comparison*

Selective SSM 由 Selective 機制決定資料的保留與更新程度，再由 SSM 裡的系統狀態 $\mathbf{x_t}$ 對資訊進行保存。Token 之間除了被包含在 1D-Convolution 的 window 內的 token 會在進行 Convolution 時互相影響之外，其他資訊則全部由 SSM 的系統狀態 $\mathbf{x_t}$ 負責保存。

Self-Attention 透過計算 token 之間的 attention score 獲得資訊，而這種作法能夠非常清楚的知道每個 token 之間的交互關係。

Selective SSM 和 Self-Attention 之間的差別就如同人類閱讀模式的不同。 Selective SSM 就像是輕鬆閱讀，只有正在閱讀的部份為了語意連貫與理解而特別注意，對於已經讀過的部分則是大概記得；Self-Attention 則像是字斟句酌地在閱讀，關注每個文字並且把握每個細節。



四：Self-Attention 示意圖

B. *Calculation Comparison*

計算量的比較上會聚焦在序列長度對模型的影響，首先是因為模型的其他計算量能夠透過超參數進行調整，再者是 Mamba 作為 Transformer 的挑戰者主打的便是能夠處理長序列的資料。

Mamba 和序列長度相關的計算主要在 In_Proj 和 1D-Convolution 中，序列長度為 N 、model dimension 為 d、 convolution 的 window 為 W ，而 W 會比 N 小的多且通常不大於 10，在 Mamba 官方提供的程式碼設定中 W 設為 4，所以能得到計算複雜度為 O($N{\cdot}d$ + $W{\cdot}N{\cdot}d$) ，$N{\cdot}d$ 對應 In_Proj ， $W{\cdot}N{\cdot}d$ 對應到 1D-Convolution。複雜度示意圖如圖五所示。

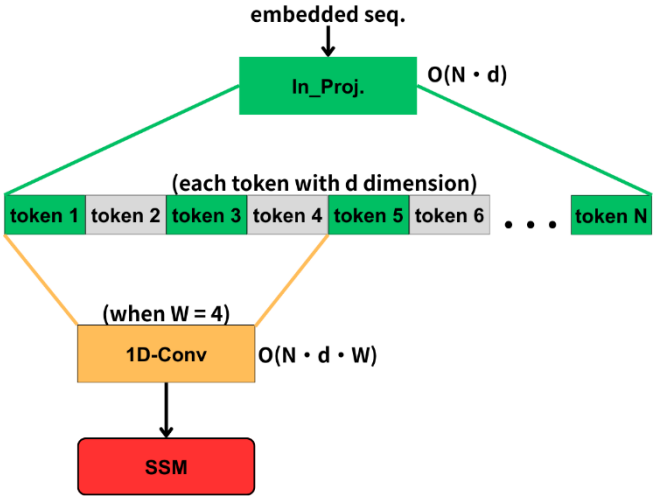
Transformer 對計算量影響最大的就是 Self-Attention 計算 attention score 的部分了，因為需要取的每個 token 和所有其他 token 之間的 attention score ，所以僅僅是 attention score 需要的計算量就已經是 O($N^2 \cdot d$) 了。

因為序列長度增加對 Mamba 的計算複雜度是線性增加的，而 Transformer 因為 Self-Attention 機制的關係導致序列長度增加對 Transformer 的計算複雜度是以平方次增加的。所以從計算量的角度來說，在相同硬體條件下 Mamba 不論在訓練還是推理的時間上都會比 Transformer 更快。

C. *Long Sequence and Stability*

在面對長序列時， Mamba 除了在計算複雜度上勝過 Transformer ，在整體架構的設計與特性上都比 Transformer 更適合處理長序列任務。

首先是 Mamba 保留資訊的方式是透過 SSM 裡的系統狀態 $\mathbf{x_t}$ 保存在模型中，且 SSM 是由訊號系統中處理連續狀態的微分方程轉變而來，所以本身面對長序列時就具有良好的穩定性。Transformer 並沒有額外的資訊儲存空間，資訊必須在層和層之間傳遞，因此更需要依賴良好的 Residual 和 Normalization 設計才能保證模型穩定。Self-Attention 的計算中使用的 Softmax 在遇到極長序列的時候可能會出現 attention score 被稀釋的問題。



圖五：Mamba 計算複雜度示意圖

IV. Implementation of Mamba and Transformer

為了實踐 Mamba 和 Transformer 之間的比較，我們選擇了兩項文本分類任務：Movie Sentiment Analysis（使用 IMDb dataset [5]）、News Classification（使用 News Category Dataset [6]），嘗試觀察在序列長度差異與資料集大小之差異上，Mamba 與 Transformer 的表現差距。

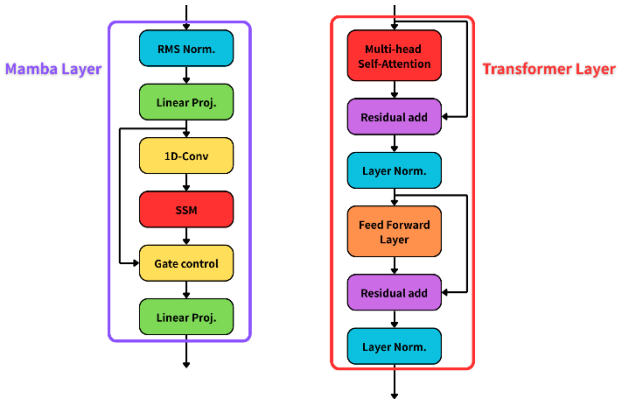
在模型的設計上我們以 Mamba Block 和 Transformer Encoder 為參考目標，定義了屬於我們這次實驗中所使用的 Mamba Layer 與 Transformer Layer (如圖六所示)。之所以參考 Mamba Block 和 Transformer Encoder 是因為我們的目標是進行文本分類，並沒有用到 Decoder 的需求，且 Mamba Decoder 目前仍處於開發階段，並沒有相關成果與程式參考。不過我們在實作 Mamba Layer 裡的 SSM 區塊時，並沒有加入 S4 在計算上的演算法，因為我們認為我們所實作的任務並不困難，序

列長度不足以觸發計算上的問題，所以我們在計算上就使用原始 SSM 的 CNN-like 的方式進行訓練，在推理的部分我們維持 RNN-like 的寄宿方式。

對於兩份資料集的分類成果評估部分，我們都使用 accuracy 作為評估依據。在訓練資源消耗的部分，記錄了模型參數量、GPU 記憶體最高使用量、訓練時間、推理時間搭配我們所定義的 Gain of Mamba 對 Mamba 和 Transformer 進行比較。透過 Gain of Mamba 的計算，我們能夠直觀的感受到 Mamba 在該比較項目上是否相較 Transformer 有較為優秀的表現，以及表現上相差的程度。Gain of Mamba 有兩種計算方式，因為我們希望能夠根據數值的正負判斷 Mamba 是否優於 Transformer，但因為 accuracy 是數值越大表現越好且具有直接參考的價值，而模型參數量、GPU 記憶體最高使用量、訓練時間、推理時間都是越小越好，所以我們會根據比較的項目不同使用不同的 Gain of Mamba 計算方式，Eq.(10)是給 accuracy 計算 Gain of Mamba 使用，而 Eq.(11)是給其餘比較項目計算 Gain of Mamba 使用。具體 Gain of Mamba 計算公式如下：

$$\text{Gain of Mamba} = \text{Mamba accuracy} - \text{Transformer accuracy} \quad (10)$$

$$\text{Gain of Mamba} = \frac{\text{Transformer value} - \text{Mamba value}}{\text{Transformer value}} \times 100\% \quad (11)$$



圖六：Mamba Layer 和 Transformer Layer

A. *Dataset Introduction*

IMDb 是一份關於電影評論的分類任務，由 25000 比訓練資料和 25000 測試資料所組成，分成正評和負評兩類。在這份資料集中平均每筆資料有 273 個單字。資料集來源直接透過 Keras 下載其內建的 IMDb 資料集[7]。由於資料集本身已經趨於完整，並且為了比較公平性我們並沒有對資料集進行處理，確認 label 與 text 沒有問題後便直接進行訓練。

News Category 是一份在 Kaggle 網站上針對新聞文字的部分進行分類，總共有大約 21 萬筆的資料，總共有 42 種分類，但這是一份不平衡的資料集，資料量最多的種類 (Politic)有 35602 筆，資料量最少的種類 (Parent)有 3955 筆。這份資料集對一筆資料的紀錄包括：category、headline、authors、link、short description、date，在這些紀錄之中我們選擇使用 category 作為 label，headline 和 short description 作為訓練資料。Headline 的平均長度為 9.6，而 short description 的平均長度為 19.67。與 IMDb 資料集相比的話是明顯較短的。資料處理的部份我們將沒有 headline 也沒有 short description 的資料刪掉，並將 headline 與 short description 連接起來成為一段文本並在 headline 和 short description 之間插入一個 SEP token 作為分界。並將處理完的資料集切出 10% 做為測試資料。

在資料集的份量上 News Category 資料集比 IMDb 資料集大了數倍，但在一筆資料的長度上 IMDb 資料集比 News Category 大了數倍，我們希望能夠藉由資料集的份量與單筆資料的序列長度差異比較出 Mamba 和 Transformer 各自的優勢。

B. *Model Design*

在模型設計的部分，我們將差異保留在 Mamba Layer 和 Transformer Layer 之間，其餘部分皆保持一樣。在 Mamba Layer 和 Transformer Layer 的堆疊層數選擇上，由於我們嘗試多層堆疊的結果未必比較好，而且近一步造成訓練時間增加，所以我們選擇使用一層進行實驗。詳細架構如表一、表二所示，關於 Mamba Layer 和 Transformer Layer 的定義請參考圖六。

表一和表二的差別位於表二 Embedding Layer 之後的 Layer Normalization 以及 Mean Layer 之後的 Linear project 表一是投影成 32 維，表二則是投影成 64 維。之所以有這樣的差異是因為 News Category 資料集份量較大且分類數量較多，所以進行這樣的調整。

	Mamba	Transformer
Input	Input	
Layer 1	Embedding Layer	
Layer 2	Mamba Layer	Transformer Layer
Layer 3	Mean Layer	
Layer 4	Linear project to 32	
Layer 5	ReLU	
Layer 6	Dropout	
Layer 7	Layer Normalization	
Layer 8	Linear project to 1	
Layer 9	Sigmoid	
Output	Class output	

表一：IMDb 資料集實驗之模型架構

	Mamba	Transformer
Input	Input	
Layer 1	Embedding Layer	
Layer 2	Layer Normalization	
Layer 3	Mamba Layer	Transformer Layer
Layer 4	Mean Layer	
Layer 5	Linear project to 64	
Layer 6	ReLU	
Layer 7	Dropout	
Layer 8	Layer Normalization	
Layer 9	Linear project to 1	
Layer 10	Softmax	
Output	Class output	

表二：News Category 資料集實驗之模型架構

C. Training Result and Comparison

訓練過程中我們透過將 k-fold 設定為 5 進行訓練，並且以 5 個 fold 所得之訓練與推理數據經平均後進行比較。在訓練資源的使用上，我們在進行 IMDb 資料集訓練時使用的是 Google Colab 所提供的 T4 GPU；在 News Category 資料集的訓練資源使用的則是 Google Colab 所提供的 A100 GPU。

在模型設定的部分，IMDb 資料集的實驗所使用的 Optimizer 為 AdamW，搭配 learning rate 設為 5e-4 和 weight decay 設為 1e-3。Loss function 的部分使用 binary cross entropy ；Optimizer 選用 AdamW，搭配 learning rate 設為 1e-3。Loss function 的部分使用 cross entropy 。

在超參數的選擇上，因為我們需要比較訓練時間的消耗，所以我們會將兩份資料集的訓練 epoch 數固定在 10 進行訓練。其餘超參數的部分，我們為了能夠完善的比較 Mamba 和 Transformer 之間的差異，我們在兩份資料集中都嘗試了相當多的超參數組合，詳細內容如表三、表四所示。在後續的實驗成果比較中，我們會在一定的範圍內尋找 accuracy 最高的參數組合之實驗結果進行比較。

	Mamba	Transformer
vocab_size	10000 / 20000	
max length	200 / 500	
batch size	32 / 64	
embedding dim	32 / 64 / 128	
d_state	16 / 32	N/A
conv_size	3 / 4	N/A
nhead	N/A	2 / 4
ff_dimension	N/A	256 / 512 / 1024

表三：IMDb 資料集實驗所嘗試的超參數

	Mamba	Transformer
vocab_size	10000 / 20000	
max length	100 / 250	
batch size	256 / 512	
embedding dim	32 / 64	
d_state	16 / 32	N/A
conv_size	2 / 3	N/A
nhead	N/A	2 / 4
ff_dimension	N/A	256 / 512 / 1024

表四：News Category 資料集實驗所嘗試的超參數

	Mb-IMDb-1	TF-IMDb-1	Gain of Mamba
模型參數量	354,018	393,121	9.9%
Best accuracy	86.3%	87.8%	-1.5%
GPU 記憶體最高使用量(MB)	244	558	52.3%
訓練時間(s)	26.48	72.69	63.5%
推理時間(s)	1.55	3.57	56.6%

表五：IMDb 資料集中，參數量 30~40 萬之間的最佳實驗結果

	Mb-IMDb-2	TF-IMDb-2	Gain of Mamba
模型參數量	740,290	663,201	-11.6%
Best accuracy	87.6%	87.7%	-0.1%
GPU 記憶體最高使用量(MB)	304	560	45.7%
訓練時間(s)	32.22	37.95	15.1%
推理時間(s)	1.69	1.83	7.6%

表六：IMDb 資料集中，參數量 60~80 萬之間的最佳實驗結果

	Mb-IMDB-1	TF-IMDb-1	Mb-IMDB-2	TF-IMDb-2
vocab_size	10000	10000	10000	20000
max length	500	200	500	200
batch size	32	64	32	64
max length	500	200	500	200
batch size	32	64	32	64
embed dim	32	32	64	32
d_state	32	N/A	16	N/A
conv_size	3	N/A	4	N/A
nhead	N/A	4	N/A	2
ff_dim	N/A	1024	N/A	256

表七：IMDb 資料集實驗中最佳實驗結果之參數組合

	Mb-News-1	TF-News-1	Gain of Mamba
模型參數量	332,395	360,234	7.73%
Best accuracy	46.3%	49.7%	-3.4%
GPU 記憶體最高使用量(MB)	1565	5865	73.3%
訓練時間(s)	88.44	193.1	54.2%
推理時間(s)	0.68	1.45	53.1%

表八：News Category 資料集中，參數量 30~40 萬之間的最佳實驗結果

	Mb-News-2	TF-News-2	Gain of Mamba
模型參數量	682,251	726,666	6.1%
Best accuracy	51.3%	52.1%	-0.8%
GPU 記憶體最高使用量(MB)	2979	7849	62.0%
訓練時間(s)	160.48	218.22	26.5%
推理時間(s)	1.04	1.66	37.3%

表九：News Category 資料集中，參數量 60~80 萬之間的最佳實驗結果

	Mb-News-1	TF-News-1	Mb-News-2	TF-News-2
vocab_size	10000	10000	10000	10000
max length	100	250	100	250
batch size	512	256	512	256
embed dim	32	32	64	64
d_state	16	N/A	32	N/A
conv_size	2	N/A	2	N/A
nhead	N/A	4	N/A	4
ff_dim	N/A	512	N/A	512

表十：News Category 資料集實驗中最佳實驗結果之參數組合

透過表五~表七針對 IMDb 資料集的實驗結果可知，在模型參數量相差不到 15%的情況下 Best accuracy 的差距非常小，甚至在 Mb-IMDb-2 和 TF-IMDb-2 之間的 Best accuracy 差距僅 0.1%，與資源消耗程度上的比較已經是微乎其微。在 GPU 記憶體最高使用量的比較上，因為 Mb-IMDb-1 和 Mb-IMDb-2 所使用的 batch size 是 32，而 TF-IMDb-1 和 TF-IMDb-2 所使用的 batch size 則是 64，由於 batch size 的增加本身就會對 GPU 記憶體使用量造成提升，所以僅憑針對 IMDb 資料集所選出的 4 個模型並沒辦法說明 Mamba 在 GPU 記憶體使用量能比 Transformer 更少。但也因為 Mb-IMDb-1 和 Mb-IMDb-2 所使用的 batch size 是 32，而 TF-IMDb-1 和 TF-IMDb-2 所使用的 batch size 則是 64，在相同的模型下，較大的 batch size 應該要有更短的訓練時間，而 Mb-IMDb-1 和 Mb-IMDb-2 卻以更小的 batch size 分別以 63.5% 和 15.1%相較於 TF-IMDb-1 和 TF-IMDb-2 的訓練時間完成訓練，藉此我們可以認定 Mamba 具有比 Transformer 更快的訓練速度。在推理時間的比較上，Mb-IMDb-1 和 TF-IMDb-1 有著 56.6% 的差距，但 Mb-IMDb-2 和 TF-IMDb-2 僅有 7.6%的差距，我們推測原因來自於超參數的選擇，因為 TF-IMDb-1 和 TF-IMDb-2 之間的模型參數量差距主要來自於 vocab size，而 nhead 卻是 4 和 2 的差別，在 III. B. Calculation Comparison 中得知，Transformer 的計算量主要落在 Self-Attention 中，所以當 nhead 降低後計算量也大幅降低，進而影響推理速度。

News Category 資料集與 IMDb 資料集最大的不同便是資料量大幅增加，但序列長度縮短，在經過表八~表十的觀察可知，在參數量差距不到 10%的情況下，Best accuracy 與 IMDb 資料集的實驗結果相仿，只是在 Mb-News-1 和 TF-News-1 之間的差距為 Mb-News-1 落後 6.88%，但因為 News Category 資料集本身資料的序列長度較短，比較適合 Transformer 發揮，所以 TF-News-1 在 Best accuracy 上有較好的表現符合我們的期望。但是在訓練資源消耗的部分 GPU 記憶體最高使用量與訓練時間上的差異因為資料集分量大幅增加導致 Mb-News 和 TF-News 之間的差距比 Mb- IMDb 和 TF-IMDb 要大得多，甚至在 Mb-News-1 和 TF-News-1 之間的 GPU 記憶體最大使用量達到了 73.3%的差距，而這樣的情況更是發生在 Mb-News-1 所使用的 batch size 為 512，而 TF-News-1 所使用的 batch size 為 256，在這樣的比較結果之下，我們可以確定 Mamba 在 GPU 記憶體的使用上會比 Transformer 更小。在推理時間的比較上，也因為資料量增加導致計算量之間的差距愈發明顯，在 Mb-News-1 和 TF-News-1 的推理時間比較上達到了 54.2%的差距，而且在超參數 nhead 在 TF-News-1 和 TF-News-2 使用 4 的情況下可以發現訓練和推理時間其實相差不多，相較於 TF-News-1 和 TF-News-2 所使用的 nhead 為 4 和 2 所造成的訓練和推理時間差距，可以知道 nhead 會很大程度的決定訓練時間和推理時間。

V. Conclusion and Future work

經過我們這次對 Mamba 的架構進行理解，以及比較 Mamba Layer 和 Transformer Layer 之間的差異實驗後，我們已經清楚地了解到為什麼 Mamba 能夠被譽為 Transformer 的挑戰者。在我們的實驗中，IMDb 資料集有著比 News Category 資料集更長的序列，而 News Category 資料集的份量比 IMDb 資料集要大得多。在模型參數量相差大約在 10%以內的情況下，雖然 Mamba Layer 在 accuracy 的表現上都落後 Transformer Layer，但差距最多為 3.4%，最少則是 0.1%。在我們所選擇的兩份資料集中 Mamba 的亮點便是訓練資源消耗與時間上的節省，使用 Mamba Layer 相較於 Transformer Layer 隨著資料量提升能夠以減少最多 73.3% 的 Transformer Layer GPU 記憶體使用量完成訓練。在訓練時間的比較上，使用 Mamba Layer 相較於 Transformer Layer 能降低至少 15.1%的訓練時間。在推理時間的比較上，使用 Mamba Layer 相較於 Transformer Layer 則是至少能夠減少 7.6%的時間。

在計算複雜度的部分，以 Selective SSM 和 Self-Attention 比較就已經能夠發現 Selective SSM 是線性的計算複雜度，而 Self-Attention 則是平方次的複雜度，隨者序列長度增加以及 nhead 的增加都會導致使用 Transformer Layer 進行訓練與推理的時間都大大增加。不過從 News Category 的實驗結果來看，即使使用的是序列長度偏短的資料集，Mamba 在計算上的優勢仍然存在，只是 Transformer 雖然在速度與記憶體使用程度不占優，但卻有著更高的 accuracy。這也符合我們對 Mamba 和 Transformer 的期待，Mamba 並不會詳細的去關心每一個 token ，而是大概記得其中的資訊；而 Transformer 則是對每個 token 都予以同等程度的注意，所以 Transformer 才能在較短的序列上有比較優秀的 accuracy 表現。

在這次的實驗中我們所選擇的資料集中，序列長度都不算長，沒有辦法表現出 Mamba 在長序列上的優勢，也沒辦法比較 Mamba Layer 和 Transformer Layer 經過堆疊之後的效能與資源消耗的比較。另外，在 Transformer Layer 的設計上沒有加上 Positional Encoding 也是我們實驗中的失誤，但即便在沒有 Positional Encoding 的狀況下，Transformer 都具有比 Mamba 更好的 accuracy 結果，所以我們更可以相信 Transformer 在短序列的表現上一定會比 Mamba 更加優秀。

雖說目前 Transformer 在 Seq2Seq 架構中仍然是霸主般的存在，再加上目前和 Transformer 相關的研究可以說是到處都是，所以 Transformer 的靈活性和泛用性

肯定是遠遠強於現在的 Mamba。不過根據 Mamba 對長序列的優勢以及較低的訓練資源消耗和較快的訓練速度，我們相信 Mamba 是有潛力成為與 Transformer 相提並論的存在。

REFERENCES

[1] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. Neural Information Processing Systems. https://arxiv.org/pdf/1706.03762

[2] Gu, A., & Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752. https://arxiv.org/pdf/2312.00752

[3] Gu, A., Dao, T., Ermon, S., Rudra, A., & Ré, C. (2020). Hippo: Recurrent memory with optimal polynomial projections. Advances in neural information processing systems, 33, 1474-1487. https://arxiv.org/pdf/2008.07669

[4] Gu, A., Goel, K., & Ré, C. (2021). Efficiently modeling long sequences with structured state spaces. arXiv preprint arXiv:2111.00396. https://arxiv.org/pdf/2111.00396

[5] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meet. Assoc. Comput. Linguistics: Human Lang. Technol.*, Portland, OR, USA, 2011, pp. 142–150.

[6] R. Misra, "News Category Dataset," Kaggle, Version 1, 2018. [Online]. Available: https://www.kaggle.com/datasets/rmisra/news-category-dataset/data [Accessed: 15-Mar-2025]

[7] TensorFlow–Keras developers, "IMDb Movie Review Dataset," Keras, 2025.[Online]. Available: https://keras.io/api/datasets/imdb/ [Accessed: 22-May-2025].