ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN


# BÁO CÁO ĐỒ ÁN CUỐI KỲ

# Building the SinoNômBERT from BERT-ancient-Chinese

## NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

**Giảng viên hướng dẫn:** PSG.TS. Đinh Điền

**Nhóm sinh viên thực hiện:**

| | |
|---|---|
| 22120068 | Nguyễn Anh Đức |
| 22120135 | Lê Quang Huy |
| 22120147 | Bùi Trần Quang Khải |
| 22120458 | Quách Hải Đăng |

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11/2024

# LỜI NÓI ĐẦU

Bài báo cáo với chủ đề **Building the SinoNômBERT from BERT-ancient-Chinese** trình bày quá trình thực hiện việc xây dựng và finetune mô hình SinoNômBERT dựa trên mô hình BERT-ancient-Chinese.

Trong đồ án này, chúng em sẽ trình bày các quy trình được thực hiện như thu thập và xử lý bộ dữ liệu, tạo tokenizer và finetune mô hình.

Chúng em xin chân thành gửi lời cảm ơn đến *thầy Đinh Điền*, *thầy Nguyễn Hồng Bửu Long*, *thầy Lương An Vinh*, *thầy Lê Thanh Tùng* đã trực tiếp giảng dạy bộ môn Nhập môn xử lý ngôn ngữ tự nhiên, dày công hướng dẫn và truyền đạt đến chúng em những kinh nghiệm lẫn kiến thức vô cùng quý giá. Chính sự hỗ trợ tận tâm của các thầy đã giúp chúng em có được nền tảng vững chắc để thực hiện dự án này.

Chúng em kính mong nhận được những đánh giá và nhận xét từ các thầy để có thể học hỏi, rút kinh nghiệm cho đồ án này và hoàn thiện hơn trong những dự án tương lai.

# MỤC LỤC

# 1 Đánh giá mức độ hoàn thành

## 1.1 Bảng phân công công việc

| MSSV | Họ Tên | Công việc | Đánh giá |
|---|---|---|---|
| 22120068 | Nguyễn Anh Đức | Thực hiện tìm kiếm dữ liệu và phát hiện lỗi, viết báo cáo | 100% |
| 22120135 | Lê Quang Huy | Thực hiện crawl data, viết báo cáo | 100% |
| 22120147 | Bùi Trần Quang Khải | Thực hiện fine tunning model, viết báo cáo | 100% |
| 22120458 | Quách Hải Đăng | Thực hiện tạo tokenizer, viết báo cáo | 100% |

Bảng 1: Bảng phân công công việc

## 1.2 Tiến độ công việc

| Số TT | Tiêu chí | Tiến độ |
|---|---|---|
| 1 | Tìm hiểu về model | 100% |
| 2 | Thu nhập dữ liệu | 100% |
| 3 | Xử lý dữ liệu | 100% |
| 4 | Tạo tokenizer, fine tunning model | 100% |

Bảng 2: Tiến độ công việc của từng bộ dữ liệu

# 2 Tìm hiểu về model

## 2.1 BERT là gì?

BERT (Bidirectional Encoder Representations from Transformers ) là một mô hình ngôn ngữ tự nhiên (NLP) tiên tiến do Google AI phát triển và công bố vào năm 2018. Đây là một trong những mô hình đầu tiên áp dụng kiến trúc **Transformers** theo cách hai chiều (Bidirectional), giúp hiểu rõ hơn ngữ cảnh của từ trong câu.

**Đặc điểm chính của BERT:**

- **Kiến trúc Transformers:** Dựa trên Transformer, BERT sử dụng cơ chế self-attention để hiểu rõ mối quan hệ giữa các từ trong một câu bất kể khoảng cách giữa chúng.
- **Hai chiều (Bidirectional):** Thay vì chỉ đọc câu theo một chiều (trái sang phải hoặc phải sang trái), BERT đọc cả hai chiều, giúp hiểu ý nghĩa

ngữ cảnh toàn diện hơn.

- **Pre-train:** BERT được huấn luyện trước trên hai tác vụ:
  - **Masked Language Modeling (MLM):** Một số từ trong câu được che (mask), và mô hình phải dự đoán các từ bị che đó.
  - **Next Sequence Prediction (NSP):** Mô hình dự đoán liệu câu thứ hai có phải câu tiếp theo hợp lí của câu thứ nhất hay không.
- **Transfer Learning:** Sau khi được huấn luyện trước, BERT có thể được fine-tune cho các bài toán NLP cụ thể như:
  - Phân loại văn bản.
  - Trả lời câu hỏi.

**Mẫu Input của BERT:**

- **Dữ liệu đầu vào:** Một chuỗi văn bản hoặc các cặp câu cần xử lí.
- **Cách mã hóa:**
  - **Token IDs:** Mỗi từ hoặc ký tự được ánh xạ thành một ID từ từ điển (vocab).
  - **Segment IDs:** Xác định câu nào thuộc câu đầu tiên (0) và câu nào thuộc câu thứ hai trong các cặp câu
  - **Attention Mask:** Xác định các token được sử dụng trong quá trình tính toán và các token nào bị bỏ qua.

```
Input 1:"[CLS] Tôi thích học NLP [SEP] BERT rất thú vị [SEP]"
Token IDs: [101, 2004, 6375, 29347, 10479, 103, 10487, 29159,
10487, 103]
Segment IDs: [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
Attention Mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

## 2.2 BERT-base-chinese

**BERT-base-chinese** là một biến thể của BERT được phát triển để phục vụ xử lí tiếng Trung hiện đại. Mô hình này kế thừa cấu trúc từ BERT gốc và được tối ưu hóa cho ngữ liệu tiếng Trung.

**Đặc điểm chính:** Mô hình thường được sử dụng cho Masked Language Model (MLM).

**Cấu hình của mô hình**

- **Số lớp ẩn (Number of layers)**: 12
- **Kích thước lớp (Hidden size)**: 768
- **Số đầu chú ý (Number of attention heads)**: 12
- **Kích thước tập từ điển (Vocabulary size)**: 21,128
- **Chiều dài chuỗi tối da (Maximum sequence length)**: 512
- **Total parameters**: khoảng 110 triệu câu

**Chức năng Fill-Mask:** Chức năng này cho phép mô hình dự đoán từ bị che trong một câu tiếng Trung hiện đại, giúp xử lý các bài toán như:

- Dự đoán từ phù hợp với ngữ cảnh.
- Sửa lỗi chính tả hoặc cú pháp.
- Hoàn thành câu tự động trong các ứng dụng viết văn bản.

## 2.3 BERT-ancient-chinese

BERT-ancient-chinese: Link to the paper.

**BERT-ancient-chinese** là mô hình được cải tiến dựa trên ý tưởng **Domain-Adaptive Pretraining** và được đào tạo dựa trên cơ sở của mô hình **BERT-base-chinese** với mục tiêu thực hiện các tác vụ đối với ngữ liệu tiếng Trung cổ.

**Mục tiêu nghiên cứu**

- **Mở rộng tập từ vựng**: Tăng kích thước từ điển lên 38,208 từ, so với 21,128 từ của BERT-base-chinese và 29,791 từ của SikuBERT. Từ điển mới bao gồm nhiều từ hiếm và ít phổ biến trong văn bản cổ, giúp cải thiện hiệu suất cho các tác vụ hạ nguồn.
- **Mở rộng tập ngữ liệu huấn luyện**: Sử dụng tập dữ liệu lớn gấp sáu lần so với *Siku Quanshu*, bao gồm các văn bản từ nhiều lĩnh vực khác nhau như Công Bộ, Đạo Giáo, Phật Giáo, Nho Giáo, Thơ Văn, Lịch Sử, Y Học, Nghệ Thuật, Dịch Học và Tử Bộ.
- **Tăng cường ngữ nghĩa cục bộ**:
  - Sử dụng BERT làm bộ mã hóa cơ bản **(Encoder)**, chuyển đổi chuỗi ký tự $X = [c_1, c_2, \ldots, c_n]$ thành biểu diễn ẩn:

$$H = \text{BERT}(X), \tag{1}$$

    trong đó $H \in \mathbb{R}^{n \times d_h}$.
  - Tích hợp các đặc trưng bigram **(Linear Bigram Layer)** bằng cách kết hợp biểu diễn ẩn của ký tự hiện tại với ký tự liền kề trước và sau. Biểu diễn bigram được tính qua lớp tuyến tính như sau:

$$b_{i1} = \text{LinearLayer1}(h_{i-1} \oplus h_i), \quad b_{i2} = \text{LinearLayer2}(h_i \oplus h_{i+1}), \tag{2}$$

    trong đó $\oplus$ là phép nối vector và $b_{i1}, b_{i2} \in \mathbb{R}^{d_b}$.
  - Kết hợp các biểu diễn bigram và biểu diễn ẩn ban đầu **(Linear Fusion Layer)** để tạo thành vector đặc trưng tổng hợp:

$$h'_i = h_i \oplus b_{i1} \oplus b_{i2}, \tag{3}$$

    trong đó $h'_i \in \mathbb{R}^{d_h + 2 \cdot d_b}$. Biểu diễn tổng hợp của cả chuỗi $H'$ được biểu diễn như sau:

$$H' = [h'_1, h'_2, \ldots, h'_n]. \tag{4}$$

Tiếp theo, sử dụng lớp tuyến tính *(LinearLayer3)* để chuyển đổi các vector tổng hợp thành biểu diễn cuối cùng:

$$L = \text{LinearLayer3}(H'), \tag{5}$$

trong đó $L \in \mathbb{R}^{n \times d_1}$.

– **Decoder**: Biểu diễn cuối $L$ được chuyển đổi thành các xác suất trên nhãn POS bằng một lớp MLP:

$$P = \text{Softmax}(WL^T + b), \tag{6}$$

trong đó $P \in \mathbb{R}^{n \times d_t}$, $d_t$ là số nhãn POS. $P_{ik}$ biểu diễn xác suất ký tự $c_i$ mang nhãn $\text{tag}_k$.

– Cuối cùng, sử dụng thuật Viterbi để giải mã chuỗi nhãn cuối cùng: Cho chuỗi nhãn cuối cùng $T = [t_1, t_2, \ldots, t_n]$, với $T \in \mathbb{R}^n$.

• **Lấy mẫu bất định (Uncertainty Sampling)**:

– Áp dụng MC-Dropout để giữ trạng thái dropout trong quá trình suy luận và tạo ra $k$ chuỗi nhãn ứng viên $\{T_1, T_2, \ldots, T_k\}$ thông qua giải mã Viterbi.

– Xác định các thành phần không chắc chắn (uncertain components) bằng cách so sánh sự khác biệt giữa các chuỗi nhãn ứng viên và chuỗi nhãn dự đoán ban đầu $T_p$. Kết hợp các từ không chắc chắn từ tất cả các nhãn ứng viên để tạo danh sách các thành phần không chắc chắn.

– Đánh giá tầm quan trọng của các thành phần không chắc chắn bằng cách so sánh độ chính xác nhãn giữa các thành phần không chắc chắn ($ACC_{\text{uncertain}}$) và các thành phần chắc chắn ($ACC_{\text{certain}}$), được định nghĩa như sau:

∗ $ACC_{\text{uncertain}}$ là độ chính xác của các nhãn dự đoán ban đầu tại các thành phần không chắc chắn.

∗ $ACC_{\text{certain}}$ là độ chính xác của các nhãn dự đoán ban đầu tại các thành phần chắc chắn.

– **Truy xuất tri thức**:

* Thu thập các câu chứa từ không chắc chắn từ kho tri thức bằng cách tìm kiếm các câu có chứa từ $w$ liên quan đến các thành phần không chắc chắn. Nếu từ $w$ chỉ có một ký tự, tạo thành hai từ ghép $w_1$ và $w_2$ bằng cách ghép với ký tự trước và sau, sau đó tìm kiếm các câu chứa $w_1$ hoặc $w_2$.
* Sắp xếp các câu theo độ tương đồng ngữ pháp với câu đầu vào $X$. Độ tương đồng giữa hai câu $P$ và $Q$ được đo bằng công thức:

$$s = \frac{\text{union}(P, Q)}{\|P\| + \|Q\|},$$

trong đó:

· $\text{union}(P, Q)$ là số ký tự chung giữa $P$ và $Q$.

· $\|P\|$ và $\|Q\|$ lần lượt là độ dài của câu $P$ và $Q$.

* Lựa chọn các câu có độ tương đồng cao nhất để làm tri thức hỗ trợ. Sau đó, kết hợp câu đầu vào $X$ với các câu tri thức để tạo chuỗi đầu vào nâng cao.

• **Khung tổng thể (Framework)**:

– **Giai đoạn 1**: Tạo chuỗi nhãn dự đoán tạm thời và xác định các thành phần không chắc chắn $U$.

* Cho chuỗi đầu vào $X = [c_1, c_2, \ldots, c_n]$, sử dụng mô hình cơ sở để tạo chuỗi nhãn dự đoán tạm thời $T_p$ và các chuỗi nhãn ứng viên.
* Áp dụng phương pháp lấy mẫu bất định để xác định các thành phần không chắc chắn $U = [c_i, c_{i+1}, \ldots, c_{i+o}]$.
* Nếu không có thành phần không chắc chắn, sử dụng $T_p$ làm chuỗi nhãn dự đoán cuối cùng $T$. Ngược lại, sử dụng $U$ để truy xuất tri thức bổ sung $K$. Nếu $X$ có nhiều thành phần không chắc chắn, các thành phần này sẽ được truy xuất và xử lý độc lập.

– **Giai đoạn 2**: Kết hợp tri thức bổ sung $K$ với chuỗi đầu vào để tái dự đoán bằng KF-BERT.

∗ Kết hợp $X$ và $K$ để tạo chuỗi đầu vào nâng cao $X' = [c_1, c_2, \ldots, c_n, \texttt{[SEP]}, k_1, k_2, \ldots, k_m]$.

∗ Xây dựng chuỗi nhãn phụ trợ $T'$ như sau:

$$t'_i = \begin{cases} t_i & \text{nếu } i \leq n \text{ và } c_i \notin U \\ \texttt{[MASK]} & \text{nếu } c_i \in U \\ \texttt{[PAD]} & \text{nếu } i > n \end{cases}$$

$$T' = [t'_1, t'_2, \ldots, t'_n, t'_{n+1}, \ldots, t'_{n+m+1}].$$

∗ Đưa $X'$ và $T'$ vào KF-BERT để tính phân phối xác suất $D$:

$$E_{T'} = \text{LabelEmbedding}(T'),$$

$$E_{X'} = \text{CharacterEmbedding}(X'),$$

$$D = \text{KF-BERT}(E_{T'} + E_{X'}),$$

trong đó $D = [d_1, d_2, \ldots, d_n]$ và $d_{ij}$ là xác suất $c_i$ được dự đoán thuộc nhãn $j$.

Embedding nhãn và ký tự là các tham số cần được huấn luyện. Chuỗi nhãn cuối cùng được xác định bằng thuật toán Viterbi. Đặc biệt, nếu có nhiều thành phần không chắc chắn trong $X$, xử lý chúng độc lập trong giai đoạn thứ hai và lấy trung bình tất cả các giá trị $D$ thu được trước khi giải mã bằng Viterbi.

**Ngữ liệu huấn luyện và đánh giá**

- Văn bản cổ chủ yếu được viết bằng chữ Hán phồn thể, chứa nhiều ký tự hiếm, đòi hỏi sự mở rộng từ điển đáng kể để đảm bảo độ phủ.
- Bộ ngữ liệu huấn luyện phong phú bao gồm các văn bản từ nhiều lĩnh vực khác nhau, tạo nền tảng vững chắc cho việc huấn luyện mô hình.

**Cấu hình, đánh giá và kết quả huấn luyện**

## Cấu hình:

- Mô hình sử dụng cấu trúc BERT với các cải tiến về từ điển và tập ngữ liệu cổ.

**Đánh giá**:

- Thử nghiệm trên các tập dữ liệu cổ điển như *Zuozhuan* và *Shiji*.
- Sử dụng BERT+CRF làm mô hình baseline để so sánh hiệu suất với SikuBERT, SikuRoBERTa và BERT-ancient-chinese.

**Kết quả**:

- Hiệu suất được đánh giá thông qua F1-score trong các tác vụ CWS và POS Tagging. BERT-ancient-chinese đạt kết quả tốt hơn so với các mô hình trước đó:

| Model | CWS (Zuozhuan) | POS (Zuozhuan) | CWS (Shiji) | POS (Shiji) |
|---|---|---|---|---|
| SikuBERT | 96.0670% | 92.0156% | 92.7909% | 87.1188% |
| SikuRoBERTa | 96.0689% | 92.0496% | 93.0183% | 87.5339% |
| BERT-ancient-chinese | **96.3273%** | **92.5027%** | **93.2917%** | **87.8749%** |

Bảng 3: So sánh hiệu suất giữa các mô hình trên tập dữ liệu Zuozhuan và Shiji.

- Việc bổ sung tri thức và xử lý thành phần không chắc chắn giúp cải thiện đáng kể độ chính xác trong các thành phần khó.

**Kết luận**:

- Mô hình BERT-ancient-chinese đã đạt được bước tiến quan trọng trong xử lý ngôn ngữ tự nhiên cho văn bản cổ.
- Các cải tiến như mở rộng từ điển, lấy mẫu bất định và truy xuất tri thức đã mang lại hiệu quả cao, mở ra hướng đi mới trong nghiên cứu Hán học số hóa và các ứng dụng liên quan.

## 2.4 Đề xuất cải tiến mô hình dành cho chữ Nôm (SinoNomBERT)

#### Mục tiêu nghiên cứu đối với chữ Nôm

Chữ Nôm là hệ thống chữ viết của người Việt, được xây dựng dựa trên chữ Hán nhưng có sự sáng tạo riêng để biểu đạt âm tiếng Việt. Mục tiêu chính của nghiên cứu là:

- Phân tích các đặc điểm ngôn ngữ và cấu trúc phức tạp của chữ Nôm.
- Giải quyết các khó khăn như: thiếu tài liệu huấn luyện, tính không đồng nhất trong ký tự và cách viết qua các thời kỳ lịch sử.
- Xây dựng một mô hình xử lý ngôn ngữ tự nhiên hiệu quả cho chữ Nôm, bao gồm các tác vụ: phân đoạn từ (CWS), gán nhãn từ loại (POS), và dự đoán các từ còn thiếu đối với các văn bản chữ Nôm (Fill-Mask).

#### Phương pháp nghiên cứu và cải tiến mô hình

Để xây dựng một mô hình hiệu quả dành riêng cho chữ Nôm, các phương pháp sau được đề xuất:

- **Mở rộng tập dữ liệu huấn luyện:**
  - Thu thập các tài liệu chữ Nôm từ các nguồn như thơ văn cổ, sắc phong, bia ký và sách chữ Nôm được số hóa.
  - Sử dụng OCR (Optical Character Recognition) để chuyển đổi văn bản chữ Nôm từ dạng hình ảnh sang dạng ký tự.
- **Áp dụng Domain-Adaptive Pretraining:**
  - Tinh chỉnh mô hình ngôn ngữ tiền huấn luyện (cụ thể là trên **BERT-ancient-chinese**), trên dữ liệu chữ Nôm để cải thiện khả năng hiểu ngữ cảnh của mô hình.
- **Xây dựng từ điển chữ Nôm điện tử:**
  - Tích hợp thông tin từ các bộ từ điển chữ Nôm hiện có, như "Tự điển Chữ Nôm Trích Dẫn" và mở rộng thêm từ vựng phổ biến từ các tài liệu lịch sử.

#### Thử nghiệm và đánh giá

- **Tập dữ liệu thử nghiệm:**
  - Sử dụng các tài liệu chữ Nôm như thơ văn của Nguyễn Du, Hồ Xuân Hương hoặc các bản sắc phong triều Nguyễn.

- **Tiêu chí đánh giá:**
  - Tác vụ dự đoán từ còn thiếu (Fill-Mask): Độ chính xác (Accuracy), Hàm mất mát (Loss), Perplexity, ...
  - F1-score cho tác vụ phân đoạn từ (CWS).
  - Độ chính xác (Accuracy) cho tác vụ gán nhãn từ loại (POS).

**Kết quả mong đợi và ứng dụng**

**Kết quả mong đợi:**

- Mô hình cải thiện hiệu suất xử lý văn bản chữ Nôm đáng kể so với các phương pháp hiện tại.
- Tạo ra một mô hình ngôn ngữ chuyên biệt dành cho chữ Nôm.

**Ứng dụng:**

- Hỗ trợ nghiên cứu và bảo tồn văn bản lịch sử Việt Nam.
- Xây dựng hệ thống dịch tự động từ chữ Nôm sang tiếng Việt hiện đại.
- Phát triển các ứng dụng phân tích văn bản lịch sử và văn học.

**Hướng phát triển tương lai**

- Xây dựng hệ thống dịch tự động từ chữ Nôm sang Quốc ngữ với độ chính xác cao hơn.
- Mở rộng nghiên cứu để xử lý các ngôn ngữ cổ khác của dân tộc Việt Nam.

Trong đồ án này, nhóm sẽ thực hiện xây dựng mô hình SinoNomBERT với tác vụ dự đoán các từ bị che trong văn bản tiếng Nôm (Fill-Mask).

# 3 Thu thập dữ liệu

## 3.1 Thông tin và nguồn dữ liệu

Quá trình Fine-tune Mô hình SinoNômBERT sẽ sử dụng các bộ dữ liệu được thu thập sau:

- **Nom_monolingual_corpus_CLC**: Bộ dữ liệu đã được làm sạch, do giảng viên cung cấp trong quá trình thực hiện đồ án, bao gồm:
  - `van_van`: Các tài liệu văn vần, thơ ca, câu đối, ...
  - `van_xuoi`: Các tài liệu văn xuôi, trích dẫn, ...
- **nomfoundation_history_of_greater_vietnam**: Bộ dữ liệu Đại Việt Sử Ký Toàn Thư về lịch sử dân tộc Việt Nam qua các thời kỳ. Nguồn dữ liệu được lấy từ: https://nomfoundation.org/nom-project/history-of-greater-vietnam/.
- **NomNaOCR**: Bộ dữ liệu bao gồm toàn bộ 5 quyển Đại Việt Sử Ký Toàn Thư. Nguồn dữ liệu được lấy từ: https://www.kaggle.com/datasets/quandang/nomnaocr.
- **ChuNom corpus**: Bộ dữ liệu bao gồm các tài liệu văn học từ sự đóng góp của cộng đồng. Nguồn dữ liệu được lấy từ: https://chunom.org/shelf/corpus/.
- **Tập vocabv4**: Bộ dữ liệu bao gồm danh sách các ký tự Hán Nôm thường dùng, do trợ giảng cung cấp trong quá trình thực hiện đồ án.
- **Strokes Hán Nôm**: Bộ dữ liệu bao gồm danh sách gần như toàn bộ các ký tự Hán Nôm đã được ghi nhận lại trong các quyển sách và tài liệu cổ do Ủy ban Phục Sinh Hán Nôm thu thập (7168 ký tự). Nguồn dữ liệu được lấy từ: https://www.asuswebstorage.com/navigate/a/#/s/4AC5951A3CCD4592B70065B38C9D0955Y.

## 3.2 Quá trình thập bộ dữ liệu History_of_greater_Vietnam trên nomfoudation

**Mô tả nguồn dữ liệu.**

- Tên trang web: nomfoundation.org

- Loại dữ liệu thu thập: Chữ Hán-Nôm (các từ, các câu, các bài thơ, ...)
- Cách dữ liệu được tổ chức trên trang web.
  - Dữ liệu từ điển được chia thành nhiều trang và nhiều danh mục (cấu trúc khá phức tạp)
  - Dữ liệu được phân trang sẵn trên từ server và được nhúng các đoạn max JavaScript để lấy dữ liệu.

**Quy trình thu thập**

- Công cụ sử dụng:
  - Ngôn ngữ Python.
  - Thư viện hỗ trợ **Beautiful Soup, Requests** và kết hợp với **Selenium** để xử lý phân trang và các đoạn mã JavaScript.
- Phương pháp thu thập:
  - Phân tích cấu trúc HTML của trang mục tiêu.
  - Lập trình một crawler để trích xuất dữ liệu cần thiết (có thể sẽ cần lập trình nhiều crawler ứng với từng cấu trúc HTML khác nhau).
  - Lưu kích thước bằng định dạng TXT
- Kich thước dự kiến: số luọng các mục ước tính trên trang web.

**Mô tả quá trình thu thập**

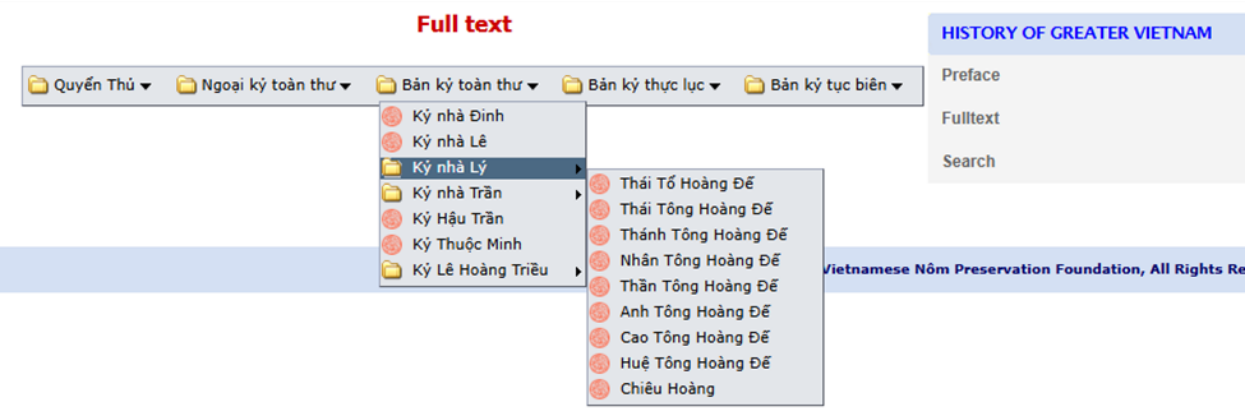
- Phân tích và trích xuất các đường link từ trang web chính (http://nomfoundation.org)
  - Ta gưi request đến trang web bằng thư viện **requests** và sử dụng **BeautifulSoup** để chuyển đổi đoạn mã phản hồi thành đoạn mã HTML.
  - Sau khi đã có được đoạn mã HTML, ta thực hiện phân tích và nhận thấy các đường link ta cần được lưu trữ trong thẻ 'a' và có class là **has_submenu**
  - Ta trích xuất đoạn mã và nhận được các đường link cần thiết.

Thắng Nghiêm Temple http://lib.nomfoundation.org/collection/2
Phổ Nhân Temple http://lib.nomfoundation.org/collection/3
Digital Library of Hán-Nôm https://nomfoundation.org/nom-project/Digital-Library-of-Han-Nom
Tale of Kiều https://nomfoundation.org/nom-project/Tale-of-Kieu
Lục Vân Tiên https://nomfoundation.org/nom-project/Luc-Van-Tien
Chinh Phụ Ngâm Khúc https://nomfoundation.org/nom-project/Chinh-Phu-Ngam-Khuc
Hồ Xuân Hương https://nomfoundation.org/nom-project/Ho-Xuan-Huong
History of Greater Vietnam https://nomfoundation.org/nom-project/History-of-Greater-Vietnam

Hình 1: Các đường link cần thiết sẽ sử dụng.

– Do một số thông tin đã có hoặc không cần thiết cho đồ án. Nên ta chỉ thu thập dữ liệu từ 3 đường link cuối.

- Thu thập dữ liêu trên **History of Greater Vietnam**
  – Dường link chứa dữ liệu chình (http://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext




Hình 2: Đường dẫn vào mục dữ liệu.

– Sau khi truy cập vào đường link trên: Ta có thể thấy rằng dữ liệu được chia ra thành nhiều quyển và mỗi quyền đều chứa các thông tin cần thiết.

– Vì vậy trước hết ta cần phân tích đoạn mã HTML từ trang web này để có thể trích xuất ra các đường link chứa dữ liêu của từng quyển.

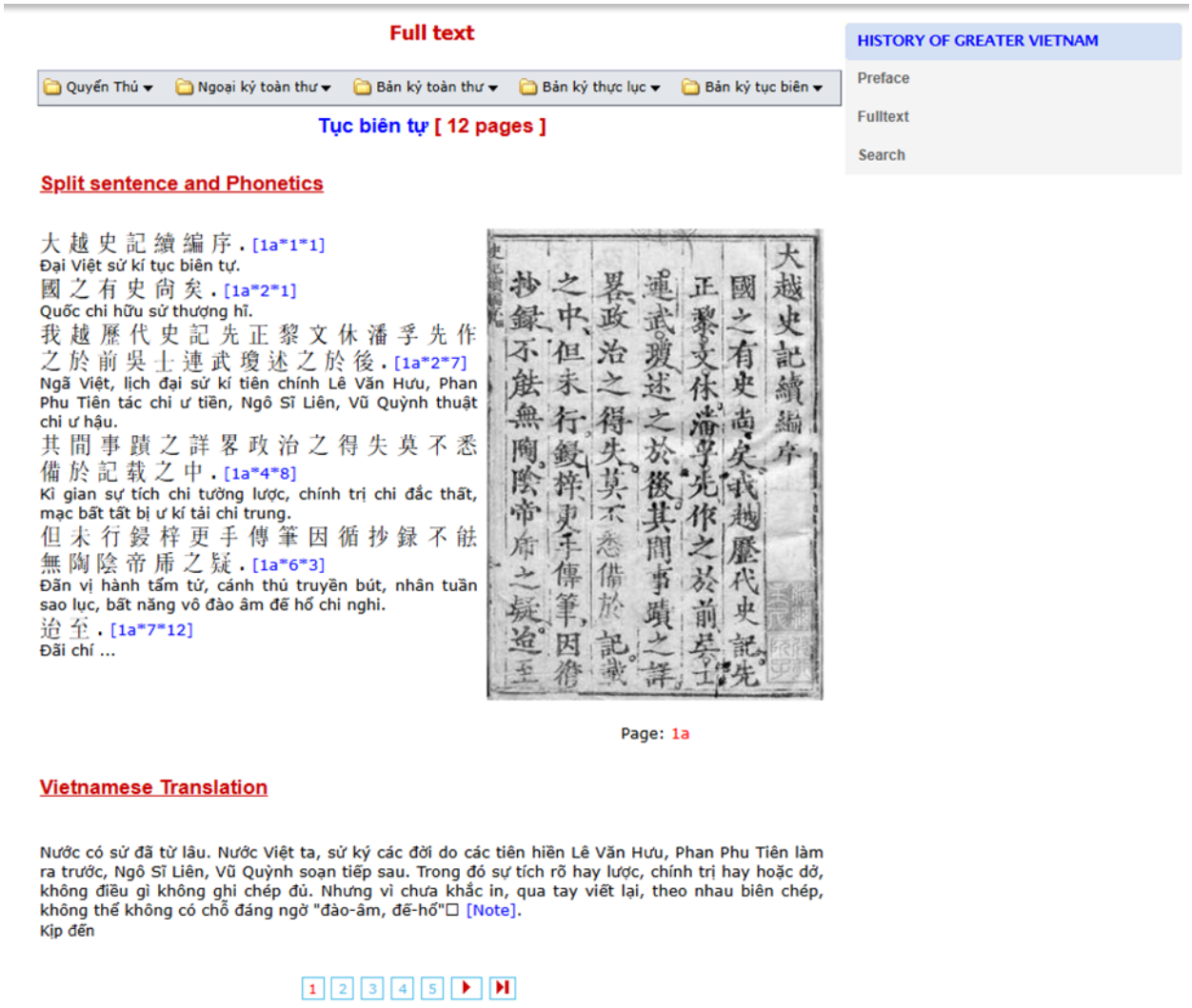
– Sau Khi phân tích đoạn mã HTML từ trang web trên thì ta nhận được các đường link và tên của từng quyển mà ta muốn thu thập.

Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/100-Tuc-bien-tu?uiLang=en, Name: Tục biên tự
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/101-Tuc-bien-thu?uiLang=en, Name: Tục biên thư
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/102-Ngoai-ky-toan-thu?uiLang=en, Name: Ngoại kỷ toàn thư
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/103-Toan-thu-bieu?uiLang=en, Name: Toàn thư biểu
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/104-Toan-thu-pham-le?uiLang=en, Name: Toàn thư phàm lệ
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/105-Ky-nien-muc-luc?uiLang=en, Name: Kỷ niên mục lục
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/106-Khao-tong-luan?uiLang=en, Name: Khảo tổng luận
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/1-Ky-Hong-Bang-thi?uiLang=en, Name: Kỷ Hồng Bàng thị
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/2-Ky-nha-Thuc?uiLang=en, Name: Kỷ Nhà Thục
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/3-Ky-nha-Trieu?uiLang=en, Name: Kỷ Nhà Triệu
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/4-Ky-thuoc-Tay-Han?uiLang=en, Name: Kỷ Thuộc Tây Hán
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/5-Ky-Trung-Nu-Vuong?uiLang=en, Name: Kỷ Trưng Nữ Vương
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/6-Ky-thuoc-Dong-Han?uiLang=en, Name: Kỷ Thuộc Đông Hán
Link: https://nomfoundation.org/nom-project/History-of-Greater-Vietnam/Fulltext/7-Ky-Si-Vuong?uiLang=en, Name: Kỷ Sĩ Vương

Hình 3: Các đường dẫn trích xuất được và tên các quyển cần thu thập.

– Tiếp theo ta cần tiến hành phân tích từng đường link dữ liệu để lấy được dữ liệu cần thiết.

– Ta truy cập vào đường link cụ thể để tiến hành phân tích.




Hình 4: Dữ liệu trong mỗi đường link cần phân tích.

– Sau khi truy cập vào đường link, quan sát sơ bộ ta có thể dễ dạng nhận thấy được mỗi quyển được phân thành nhiều trang. Và để có dữ liệu

của từng trang ta cần truy cập vào các trang kế tiếp để lấy được hoàn toàn.

– Tuy nhiên ta lại gặp một vấn đề Tại đây sự phân trang không sử dụng các đường link như các page khác mà sử dụng các đoạn mã javascript được nhúng vào trang và khi gọi các đoạn mã thì server sẽ trả về các trang khác javascript:GotoPage(??).

– Vì thế ta cần sự hỗ trợ của công cụ Selenium để giả lập trình duyệt để truy cập và thực hiện

```
def create_driver():
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')

    # Khởi tạo WebDriver mới
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
    return driver
```

Hình 5: Hàm create_driver để giả lập trình duyệt.

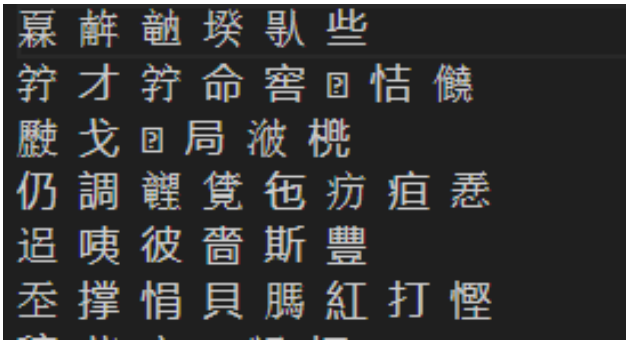
– Sau khi hoàn thành xử lí trên 1 đường link cụ thể ta tiến hành cài đặt để chạy trên các đường link còn lại và lưu chúng dưới định dạng TXT.

# 4 Tạo Tokenizer mới cho mô hình SinoNomBert

## 4.1 Xử lý dữ liệu

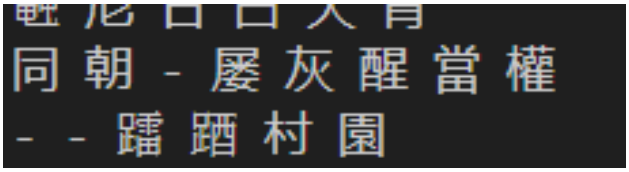
**Gom nhóm dữ liệu**

- **Dữ liệu Nom_monolingual_corpus_CLC và chunom.org**: Đưa các tập dữ liệu này vào chung với nhau vì chúng đang cùng format và ghi mỗi file sau khi chuẩn hóa (normalize) vào cùng một thư mục là `TokenNom`.



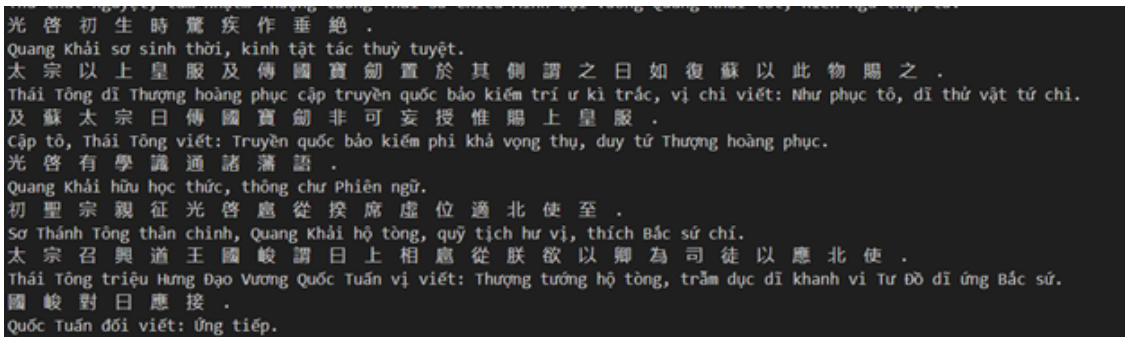
Hình 6: Dữ liệu chữ Nôm với các kí tự ? nằm trong vùng PUA của unicode.

- Các hàng của hai tập dữ liệu này đã được xử lý để mỗi dòng chỉ gồm các ký tự Hán. Tuy nhiên, vẫn còn trường hợp dính chữ và dữ liệu có chứa dấu "-" giữa các ký tự.



Hình 7: Dữ liệu gây nhiễu.

- **Dữ liệu từ web Nomfoundation**: Đặc biệt hơn khi có chứa cả dòng chữ quốc ngữ nên sẽ cần lọc ra và bỏ dấu chấm cuối câu.

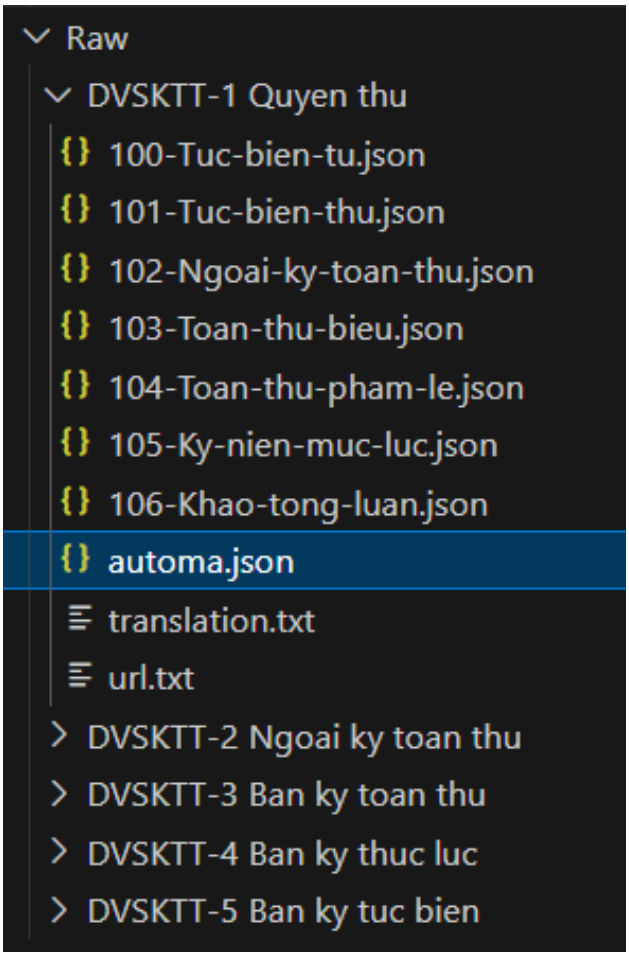



Hình 8: Dữ liệu được lấy từ web nomfoundation với các câu chữ Nôm và Việt cách nhau.

- **Dữ liệu từ NomNaOCR**: Chỉ sử dụng 5 quyển Đại Việt Sử Ký Toàn Thư. Các phần khác trong `NomNaOCR` như Truyện Kiều hoặc Chinh Phụ Ngâm đã xuất hiện trong `Nom_monolingual_corpus_CLC` nên không lấy lại.

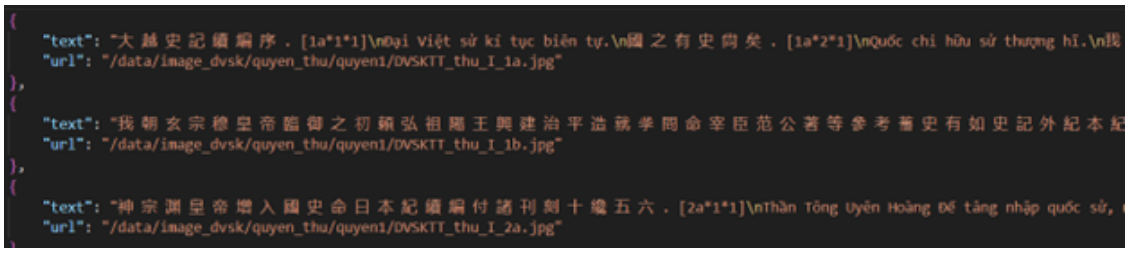
- Với dạng dữ liệu này sau khi tải thư mục `RAW` về thì sẽ làm việc trên file

`automa.json` trong mỗi thư mục chứa toàn bộ phần text OCR được từ sách.




Hình 9: File automa.json trong thư mục raw chứa các chữ đã được ocr.

- Để xử lý dữ liệu dạng này thì với mỗi value của key text, phần code sẽ chia các câu ra với nhau bởi dấu chấm và chỉ giữ lại những câu không phải chữ quốc ngữ.




Hình 10: Dữ liệu trong file automa.json

- Sau khi đã có đầy đủ dữ liệu trong thư mục `TokenNom`, ta sẽ xử lý lại toàn bộ data để xử lý các trường hợp đặc biệt.

**Xử lý các trường hợp đặc biệt**

- Xóa dấu - dư thừa vì nó gây nhiễu và làm mất nghĩa câu khi thực hiện `fill-mask` nếu `[MASK]` được đánh cho ký tự này sẽ gây nhiễu. Và việc bỏ - khiến câu vẫn có nghĩa đúng với bình thường.
- Xử lý ký tự đặc biệt ở đầu và cuối dòng. Lý do là vì ký tự này theo tìm hiểu của nhóm trên các web từ điển hán nôm thì nó là dấu chấm hoặc số không nếu nằm giữa nên việc giữ nguyên có thể gây nhiễu.
- Xóa nội dung trong `[]` do trong quyển 3 của Đại Việt Sử Ký Toàn Thư có xuất hiện các dấu ngoặc này dùng để ghi chú cho tên gọi hoặc tên tác giả nên sẽ xóa các trường hợp này để giảm thiểu nhiễu khi train.
- Xử lý các ký tự đặc biệt trong data và các dòng trùng nhau bị loại bỏ.

Kết quả được lưu ở file `extremely_clean_data.txt` với 69,746 dòng dữ liệu và dung lượng 3,116 KB (khoảng 3 MB).

**Tạo tập vocab_Han_Nom**

Sau khi đã hoàn thành bộ dữ liệu chính, ta sẽ tiến hành xây dựng bộ từ vựng từ tập dữ liệu ban đầu. Nhưng để dễ mở rộng cho các trường hợp sau này, nhóm đã dùng thêm các bộ từ vựng được giảng viên cung cấp và `strokes Hán Nôm` (bộ từ vựng do cộng đồng thu thập) để tạo 1 bộ từ vựng đầy đủ nhất có thể.

- Lấy toàn bộ ký tự Hán Nôm từ các file: `QuocNgu_SinoNom_Dic.xlsx`, `SinoNom_Similar_Dic_v2.xlsx`, bộ dữ liệu ở trên, và `strokes_Han_Nom.xlsx`.
- Gôm các ký tự và dùng tập hợp (`set`) để loại bỏ các ký tự trùng lặp và giữ lại những ký tự duy nhất trong bộ từ vựng.

Kết quả: Bộ từ vựng được xây dựng bao gồm 32,361 ký tự (cả Hán và Nôm).

## 4.2 Mở rộng vocab của bert-ancient-chinese và tạo tokenizer mới

- Ban đầu, phần vocab chỉ đơn giản được mở rộng bằng cách thêm các ký tự trong `vocab_Han_Nom` chưa xuất hiện trong vocab gốc. Tuy nhiên khi training ở bước sau thì ta không thể decode các ký tự ?, tức là những ký tự có mã unicode nằm trong vùng mở rộng PUA (Private Use Area) của UTF-8.

- Thực hiện điều chỉnh tokenizer:
  Nhóm đã có 4 lần điều chỉnh lớn để đưa ra được cách tạo tokenizer phù hợp:
  **Lần 1**: Chỉnh sửa lại tokenizer_config.json bằng cách

  – Tắt `do_basic_token` để bỏ qua tách các ký tự đặc biệt như vùng PUA, nhưng dữ liệu đầu vào phải tách sẵn bằng dấu cách.

  – Chuyển `[UNK]` thành `false` để tokenizer không tự động dán nhãn `[UNK]` cho các ký tự không xác nhận được.

  **Lần 2**: Thêm từ vựng vào special tokens

  – Nhưng như thế này vẫn chưa tốt trong việc huấn luyện dữ liệu và chưa thực sự toàn diện. Vì thế nhóm chuyển sang mở rộng bằng cách thêm các từ vựng mới này vào vùng `special_tokens`.

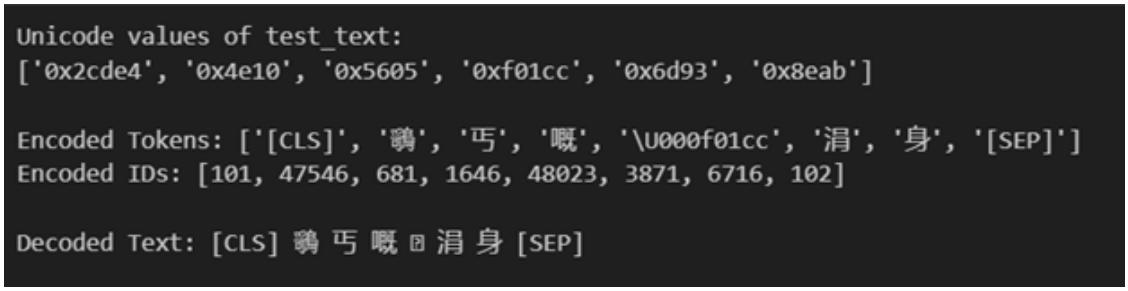
```
# Define your special tokens (e.g., [UNK], [PAD], [CLS], etc.)

special_tokens = {'additional_special_tokens': special_list}

# Add the special tokens to the tokenizer
tokenizer.add_special_tokens(special_tokens_dict=special_tokens)

# Resize the model embeddings to account for the new tokens
model.resize_token_embeddings(len(tokenizer))
```

Hình 11: Thêm chữ Nôm vào tokenizer bằng cách add_special_tokens

  – Tổng số từ vựng mới thêm vào (đã bỏ những kí tự trùng) là 12,219. Đây cũng là con số gần đúng với con số hiện tại tìm được.(Nguồn

tham khảo: https://nomfoundation.org/nom-tools/Tu-Dien-Chu-Nom-Dan_Giai/Introduction?uiLang=vn)

– Kết quả: Sử dụng tokenizer mới và thêm vào model bert-ancient-chinese thì tokenizer này đã có thể encode và decode được các ký tự PUA.




Hình 12: Các kí tự ? nằm trong vùng PUA đã encode và decode đúng câu đầu vào

**Lần 3**: Thêm từ vựng bằng add_tokens

– Mặc dù dùng phương pháp add_special_tokens có thể giúp encode và decode được chữ Nôm nằm trong vùng PUA nhưng vấn đề mới xuất hiện khi trong thư viện DataCollator không hỗ trợ việc MASK các ký tự nằm trong add_special_tokens tức là khi xây dựng data cho task fill_mask thì chữ Nôm sẽ không được đánh dấu.

– Vì thế nhóm đã thay thế bằng cách sử dụng hàm add_tokens. Nhưng add_tokens lại không tương thích với bert ancient chinese (phần này nhóm vẫn chưa rõ) khi mà nhóm sử dụng thư viện **AutoTokenizer** có sử dụng fast hay không thì đều không thể encode được các từ mới thêm vào mà sẽ xuất hiện các lỗi như: Nếu chỉ thêm mỗi chữ Nôm (12k từ) thì sẽ xuất hiện lỗi không decode được.


Hình 13: Lỗi AddedVocabulary bad split khi sử dụng AutoTokenizer

Nếu thêm toàn bộ (38k từ của chữ Hán và Hán cổ cùng với 12k từ của chữ Nôm) thì fast sẽ decode nhầm những kí tự pua thành khoảng trắng và chỉ có thể khắc phục nếu điều chỉnh do_basic_tokenize = False


Hình 14: Chữ cái nằm trong vùng PUA sẽ bị decode thành khoảng trắng tương ứng với id: 343

**Lần 4**: Thay đổi thư viện sang BertTokinizer

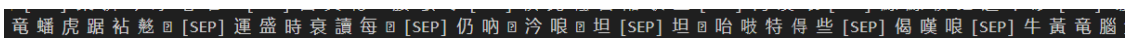
* Khi chuyển sang BertTokenizer và với việc thêm toàn bộ gần 50k từ vựng bằng cách add_tokens thì tokenizer_config.json trở nên rất lớn.
* Tuy nhiên đây là cách tính đến hiện tại có thể dùng để encode và decode được các kí tự PUA cũng như DataCollator có thể mask được các từ này

– Kết quả cuối cùng


Hình 15: Ký tự ? ở trên đã decode được với id là 29685

## 4.3 Xây dựng lại data từ tokenizer mới

- Trong quá trình huấn luyện với bộ data được lấy từ data gốc là các câu trong bài thơ hoặc các câu trong đoạn văn thì nhóm nhận thấy hiệu suất của mô hình chưa cao và vì mỗi câu đều được gán max_length = 256 trong khi độ dài mỗi câu quá ngắn dẫn đến không tận dung hết tài nguyên.
- Vì thế nhóm đã gom các câu trong cùng 1 đoạn văn lại với nhau và đặt [SEP] ở giữa để phần tách các câu. (Ý tưởng nhóm em tham khảo từ https://aclanthology.org/2023.alt-1.3.pdf)




Hình 16: Các câu được ghép lại và cách nhau bởi [SEP]

- Việc ghép các câu trong cùng 1 đoạn văn hay bài thơ giúp cho mô hình học hiểu tốt ngữ cảnh hơn các câu đầu vào dài hơn giúp tiết kiệm tài nguyên

khi giảm từ gần 70k câu xuống còn 8k câu.

- Và đặc biệt khi thay đổi cách tiếp cận này thì trong quá trình finetune hiệu suất đã tăng rõ rệt.

# 5 Quá trình fine-tune

## 5.1 Môi trường thực hiện huấn luyện

**Nền tảng:** Colab
**Phần cứng:** GPU A100

## 5.2 Chuẩn bị dữ liệu

- **Tải dữ liệu** và thực hiện xáo trộn với seed=42 để đảm bảo kết quả tái lập.
- **Chia dữ liệu** thành 3 tập:
  - **Train** chiếm 90% dữ liệu, dùng để huấn luyện mô hình.
  - **validation** chiếm 5 % dữ liệu dùng để đánh giá mô hình trong quá trình huấn luyện.
  - **Test:** Chiếm 5 % dữ liệu dùng để kiểm tra hiệu năng của mô hình sua khi huấn luyện.
- **Lưu trữ dữ liệu** đã chi vào folder my_dataset theo định dạng thư viện datasets

## 5.3 Tải lên Hugging Face

- Nếu muốn chia sẻ dữ liệu hoặc lưu trữ trực tuyến, tải dataset đã chuẩn bị lên nền tảng hugging Face bằng tài khoản cá nhân.

## 5.4 Chuẩn bị mô hình

- **Tải mô hình tiền huấn luyện:** bert-ancient-chinese. Tải từ Hugging Face: https://huggingface.co/Jihuai/bert-ancient-chinese.

- **Nếu sử dụng tokenizer tùy chỉnh:**
  - Tích hợp `tokenizer` với mô hình.
  - Thực hiện `resize_token_embeddings` để mở rộng embedding, đảm bảo tương thích giữa `tokenizer` và mô hình.

### 5.5 Load dataset và tokenize

Thực hiện load dataset từ my_dataset và tokenize làm đầu vào cho mô hình huấn luyện.

### 5.6 Data Collator cho Mô hình Masked Language Model

Sử dụng `DataCollatorForLanguageModeling` trong quá trình huấn luyện, với tỷ lệ 20% các token bị che (masked).

### 5.7 Cấu hình huấn luyện

**Các tham số quan trọng:**

- `data_collator`: `DataCollatorForLanguageModeling` được sử dụng với cấu hình như trên
- `per_device_train_batch_size`: Batch size cho tập train, thiết lập giá trị 32 (có thể điều chỉnh dựa trên bộ nhớ GPU).
- `per_device_eval_batch_size`: Batch size cho tập validation và test, cũng là 32.
- `learning_rate`: Tốc độ học là 5e-5 để đảm bảo hiệu quả tối ưu hóa.
- `num_train_epochs`: Số lượng epoch là 20.
- `evaluation_strategy`: Đánh giá mô hình sau mỗi epoch.
- `save_strategy`: Lưu mô hình tại mỗi epoch nếu có cải thiện.
- `save_total_limit`: Lưu tối đa 2 checkpoint tốt nhất.

- `early_stopping_patience`: Dừng sớm nếu không có cải thiện trong 2 epoch liên tiếp.
- `early_stopping_threshold`: Ngưỡng cải thiện tối thiểu là 0.01.
- `logging_steps`: Ghi log sau mỗi 100 bước.
- `fp16`: Sử dụng floating-point 16-bit nếu có GPU.
- `logging_dir`: Thư mục `./logs` để lưu trữ thông tin huấn luyện.
- `report_to`: Kích hoạt TensorBoard để theo dõi quá trình huấn luyện.
- `optim`: Sử dụng `adamw_torch` làm thuật toán tối ưu.

### 5.8 Thực hiện huấn luyện

- **Sử dụng Trainer để huấn luyện:**
  - **Tập train:** Dùng để cập nhật trọng số mô hình.
  - **Tập validation:** Đánh giá hiệu năng sau mỗi epoch.
- Sau khi huấn luyện, lưu mô hình đã tinh chỉnh vào thư mục `./fine_tuned_model` cùng với `tokenizer`.

### 5.9 Đánh giá mô hình

Tính toán **perplexity** (càng thấp càng tốt, giống loss) để đánh giá độ phức tạp của mô hình và **Accuracy mask token** (càng cao càng tốt) để đánh giá mức độ chính xác của mô hình trong việc dự đoán token mask.

- **Trên tập validation:**
  - Dùng để kiểm soát overfit của tập huấn luyện, giúp đánh giá khả năng tổng quát của mô hình.
- **Trên tập test:**
  - Tính toán **perplexity** để kiểm tra hiệu năng mô hình trên dữ liệu chưa từng thấy, từ đó đánh giá độ phức tạp và khả năng dự đoán của mô hình.

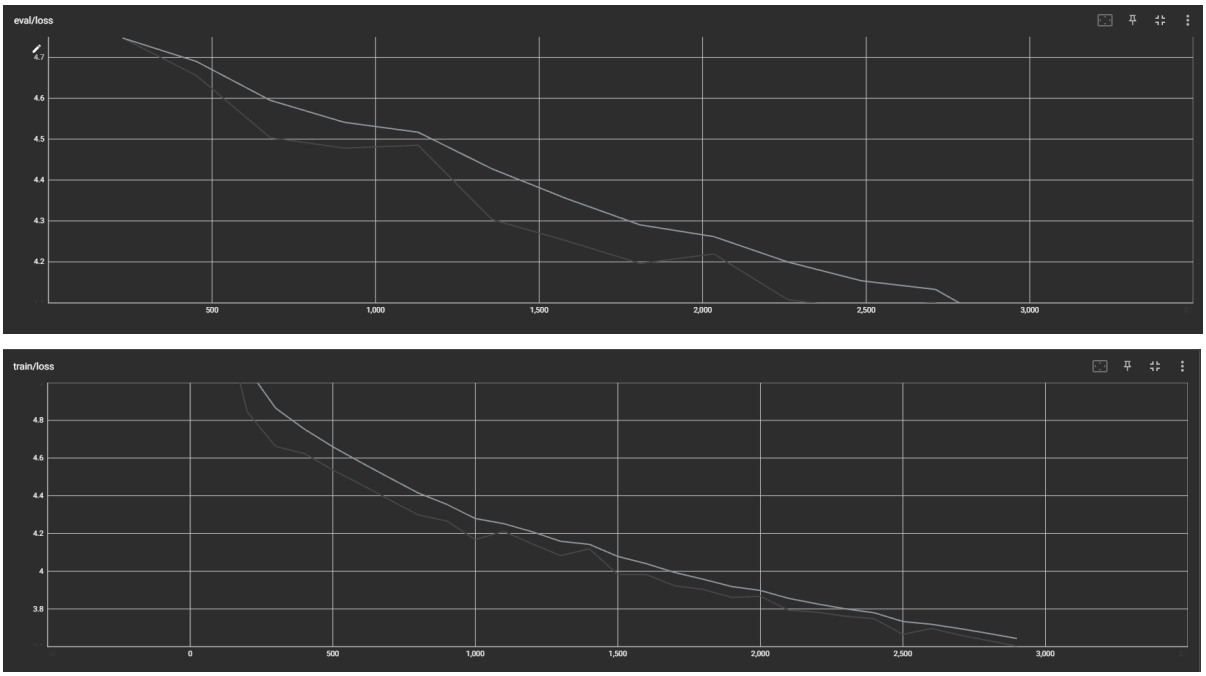
– Tính **Accuracy mask token** để đánh giá độ chính xác của mô hình trong việc dự đoán các token bị mask trong tập test.

## 5.10 Theo dõi kết quả huấn luyện

- Ghi lại thông tin huấn luyện chi tiết vào thư mục `./logs`.
- Sử dụng `TensorBoard` để trực quan hóa các chỉ số như **loss**.
- Mở TensorBoard bằng lệnh:

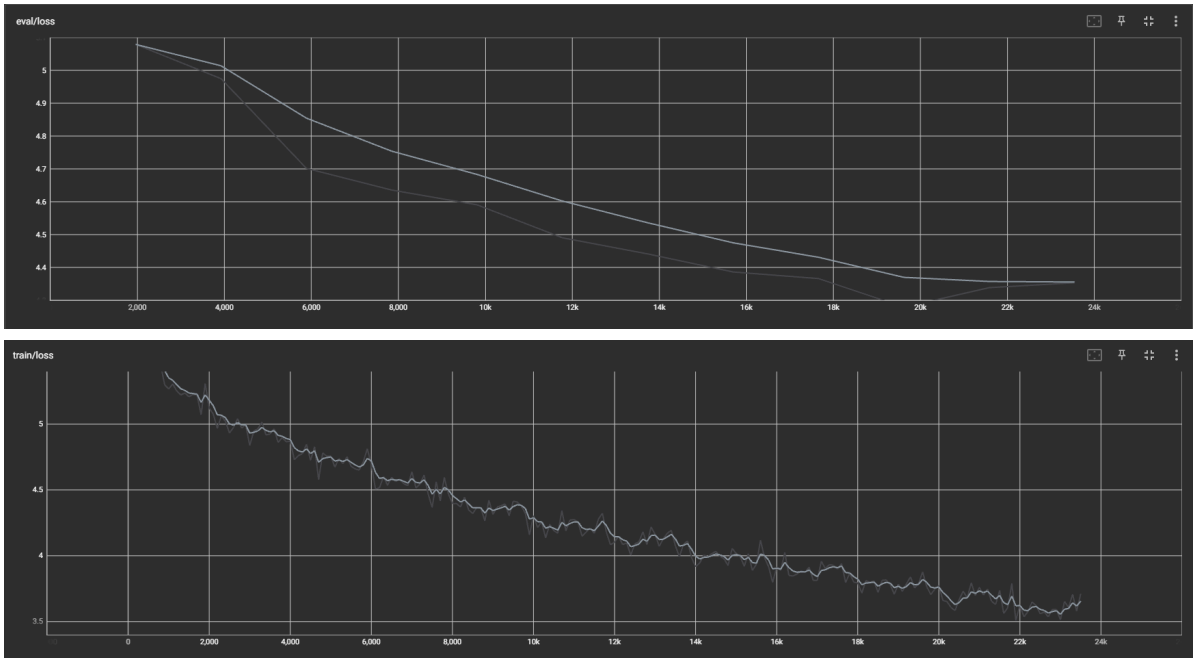
```
tensorboard --logdir ./logs
```

### Thời gian training: 4 phút cho dataset được ghép câu




Hình 17: Biểu đồ đường của train và val loss trong quá trình huấn luyện của tập dataset được ghép câu.

### Thời gian training: 21 phút cho dataset gốc




Hình 18: Biểu đồ đường của train và val loss trong quá trình huấn luyện của tập dataset gốc.

## 5.11 Kết quả thu được trên tập test

**Dataset concat:**

- **Perplexity:** 48.25
- **Accuracy:** 39.33%

**Dataset original:**

- **Perplexity:** 77.73
- **Accuracy:** 33.32%

## 5.12 Tải lên Hugging Face và Demo:

Demo inference có sẵn sau khi đánh giá bộ dữ liệu trong file jupyter notebook (content/concate_version.ipynb, content/original_version.ipynb) của thư mục training.

Model huấn luyện dataset ghép câu được tải lên tại: https://huggingface.co/btqkhai/SinoNomBERT.

## 5.13 Cho phép huấn luyện lại với:

- Chạy công cụ tiền xử lý data với bất kỳ dataset nào.
- Chạy file `split_hf.py` để chia bất kỳ dataset nào.
- Chạy huấn luyện dùng `jupyter notebook` đã cung cấp với dataset đã được chia.
- Có thể thay bất kỳ tokenizer, mô hình tiền huấn luyện nào từ Hugging Face.
- Tái hiện kết quả: Cung cấp sẵn tokenizer, dataset đã được chia nằm trong tất cả 1 thư mục content chỉ cần chạy lại file jupyter notebook tương ứng trong thư mục training.

## 5.14 Ưu điểm và khuyết điểm

- **Ưu điểm:**
  - **Dataset:** Đa dạng từ nhiều nguồn đã được tiền xử lý sạch sẽ.
  - **Tokenizer:** Phát triển thêm từ tokenizer base model, nhận diện được các ký tự đặc biệt của chữ Nôm và vocab gần như đủ các ký tự chữ Nôm hiện có.
  - **Model:** Học tốt trên tập train, hội tụ đều.
  - **Framework:** Tổng quan, thực hiện lại được với mọi dataset.
- **Khuyết điểm:**
  - **Model:** Accuracy dự đoạn chưa quá cao.
  - **Dataset:** Kích thước nhỏ, dễ dẫn đến overfit.
  - **Dataset phức tạp:** Các thơ văn có cấu trúc đặc biệt, mỗi câu riêng lẻ có thể không liên quan đến nhau nhiều.
  - **Tokenizer lớn**: Do đặc điểm riêng của base model mà chỉ có thể thêm vào toàn lại toàn bộ vocab vào tokenizer_config khiến tokenizer trở nên rất lớn làm tốn nhiều thời gian finetune hơn.

# Tài liệu tham khảo


[1] Hugging Face, Masked Language Modeling, https://huggingface.co/docs/transformers/tasks/masked_language_modeling.

[2] Hugging Face, Trainer Main Class, https://huggingface.co/docs/transformers/main_classes/trainer.

[3] Hugging Face, Data Collator Main Class, https://huggingface.co/docs/transformers/main_classes/data_collator.

[4] Hugging Face, Tokenizer Main Class, https://huggingface.co/docs/transformers/main_classes/tokenizer.

[5] Hugging Face, BERT Base Chinese Model, https://huggingface.co/google-bert/bert-base-chinese.

[6] Hugging Face, BERT Ancient Chinese Model, https://huggingface.co/Jihuai/bert-ancient-chinese.

[7] Hugging Face, Fine-tuning a Masked Language Model, https://huggingface.co/learn/nlp-course/chapter7/3.

[8] The Uncertainty-based Retrieval Framework for Ancient Chinese CWS and POS, https://aclanthology.org/2022.lt4hala-1.25.pdf.