TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KÌ**

**MÔN NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN**

**ĐỒ ÁN CUỐI KÌ MÔN NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN**

*Người hướng dẫn*: **GV. LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN THỊ VẸN – 52100674**

**TRƯƠNG BỈNH THUẬN - 52100322**

**NGUYỄN ĐÌNH DANH – 52100878**

Lớp: **21050301**

Khoá: **25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KÌ**

**MÔN NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN**

**ĐỒ ÁN CUỐI KÌ MÔN NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN**

*Người hướng dẫn*: **GV. LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN THỊ VẸN – 52100674**

**TRƯƠNG BỈNH THUẬN - 52100322**

**NGUYỄN ĐÌNH DANH – 52100878**

Lớp: **21050301**

Khoá: **25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

LỜI CẢM ƠN

Trong suốt quá trình học tập và rèn luyện, chúng em đã nhận được rất nhiều sự giúp đỡ tận tình, sự quan tâm, chăm sóc của thầy Lê Anh Cường. Ngoài ra, chúng em còn được thầy truyền đạt những kiến thức về xử lý ảnh hay ho và thú vị, thầy cô còn giúp sinh viên có được nhiều niềm vui trong việc học và cảm thấy thoải mái, … Chúng em xin chân thành cảm ơn các thầy cô rất nhiều trong suốt quá trình học tập này!

Bởi lượng kiến thức của chúng em còn hạn hẹp và gặp nhiều vấn đề trong quá trình học nên báo cáo này sẽ còn nhiều thiếu sót và cần được học hỏi thêm. Chúng em rất mong em sẽ nhận được sự góp ý của quý thầy cô về bài báo cáo này để chúng em rút kinh nghiệm trong những môn học sắp tới. Cuối cùng, chúng em xin chân thành cảm ơn quý thầy cô.

TP Hồ Chí Minh, ngày 20 tháng 05 năm 2024

Sinh viên:

TRẦN THỊ VẸN – 52100674

TRƯƠNG BỈNH THUẬN - 52100322

NGUYỄN ĐÌNH DANH - 52100878

**BÁO CÁO CUỐI KÌ ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm của riêng tôi/chúng tôi và được sự hướng dẫn của thầy Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 20 tháng 05 năm 2024*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Trần Thị Vẹn*

*Nguyễn Đình Danh*

*Trương Bỉnh Thuận*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

# TÓM TẮT

* Giới thiệu về Byte-Pair Encoding (BPE)
* Khái niệm
* Nguyên lý hoạt động
* Ví dụ về các mô hình sử dụng BPE

GPT

BERT

RoBERTa

* So sánh mô hình sử dụng BPE và không sử dụng BPE. Chọn bài toán cụ thể:
* Tiến hành train và test trên cùng một bộ dữ liệu
* So sánh kết quả và hiệu suất
* Xây dựng mô hình để tạo một trợ lý ảo (agent) làm nhiệm vụ chăm sóc khách hàng trong một lĩnh vực

# MỤC LỤC

[TÓM TẮT 4](#_Toc167134012)

[MỤC LỤC 5](#_Toc167134013)

[DANH MỤC HÌNH ẢNH, BẢNG BIỂU VÀ ĐỒ THỊ 6](#_Toc167134014)

[CHƯƠNG 1 - LÝ THUYẾT TOKENIZATION 7](#_Toc167134015)

[1.1 Định nghĩa và mục tiêu Tokenization 7](#_Toc167134016)

[1.2 Các phương pháp tokenization 7](#_Toc167134017)

[1.2.1 Word-based tokenization 9](#_Toc167134018)

[1.2.2 Character-based tokenization 10](#_Toc167134019)

[1.2.3 Subword-based tokenization 11](#_Toc167134020)

[1.3 Chuẩn hoá và tiền tokenize 12](#_Toc167134021)

[1.3.1 Normalization 13](#_Toc167134022)

[1.3.2 Pre-tokenization 14](#_Toc167134023)

[CHƯƠNG 2 - LÝ THUYẾT BPE 16](#_Toc167134024)

[2.1 Tổng quan 16](#_Toc167134025)

[2.2 Khái niệm về BPE (Byte-Pair Encoding) 17](#_Toc167134026)

[2.3 Thuật toán huấn luyện BPE 17](#_Toc167134027)

[2.4 Thuật toán tokenize BPE 19](#_Toc167134028)

[2.5 Một số kĩ thuật khác 20](#_Toc167134029)

[2.5.1 WordPiece tokenization 21](#_Toc167134030)

[2.6 Sự khác nhau giữa BPE và WordPiece 22](#_Toc167134031)

[TÀI LIỆU THAM KHẢO 24](#_Toc167134032)

# DANH MỤC HÌNH ẢNH, BẢNG BIỂU VÀ ĐỒ THỊ

**Danh mục hình ảnh**

[Hình 1.1 Tokenization 7](#_Toc167134007)

[Hình 1.2 Word-based tokenization 9](#_Toc167134008)

[Hình 1.3 Character-based tokenization 10](#_Toc167134009)

[Hình 1.4 Subword-based tokenization 12](#_Toc167134010)

[Hình 1.5 Chuẩn hoá và tiền tokenize 13](#_Toc167134011)

**Danh mục bảng biểu**

# LÝ THUYẾT TOKENIZATION

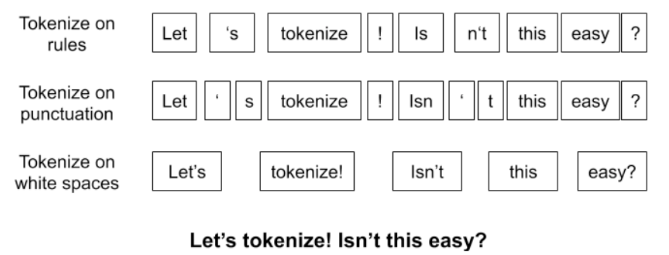
## Định nghĩa và mục tiêu Tokenization

Tokenization (tách từ) là một trong những bước quan trọng nhất trong quá trình tiền xử lý văn bản. Cho dù bạn đang làm việc với các kỹ thuật NLP truyền thống hay sử dụng các kỹ thuật học sâu nâng cao thì vẫn không thể bỏ qua bước này. Nói một cách đơn giản, tokenization là quá trình tách một cụm từ, câu, đoạn văn, một hoặc nhiều tài liệu văn bản thành các đơn vị nhỏ hơn. Mỗi đơn vị nhỏ hơn này được gọi là Tokens.

Trong xử lý ngôn ngữ tự nhiên, token là đơn vị nhỏ nhất để biểu diễn từ ngữ. Token có thể là một từ, là một ký tự hay cũng có thể là tập hợp từ, tập hợp ký tự. Thuật toán tách token là chia các văn bản thành nhiều đơn vị nhỏ (token) nhằm số hóa chúng sau đó đưa dữ liệu này vào đào tạo các mô hình.

Trong các tác vụ NLP, dữ liệu thường được xử lý là văn bản thô. Đây là một ví dụ về văn bản như vậy: **Jim Henson was a puppeteer**

Tuy nhiên, các mô hình chỉ có thể xử lý số, vì vậy chúng ta cần tìm cách chuyển văn bản thô thành số. Đó là những gì mà tokenizer làm, và có rất nhiều cách để thực hiện điều này. Mục tiêu đề ra là tìm ra cách biểu diễn có ý nghĩa nhất - nghĩa là cái có ý nghĩa nhất đối với mô hình - và, nếu có thể, là cách biểu diễn nhỏ nhất.



Hình 1.1 Tokenization

## Các phương pháp tokenization

* Whitespace Tokenization: Phương pháp đơn giản nhất, chia văn bản dựa trên khoảng trắng.

Ví dụ: "Tôi yêu NLP" -> ["Tôi", "yêu", "NLP"]

* Punctuation-based Tokenization: Chia văn bản bằng cách loại bỏ dấu câu và khoảng trắng.

Ví dụ: "Tôi yêu NLP, và bạn?" -> ["Tôi", "yêu", "NLP", "và", "bạn"]

* Word Tokenization: Chia văn bản thành các từ sử dụng các quy tắc cụ thể, có thể là phức tạp hơn các phương pháp trên. Thường sử dụng các thư viện như NLTK trong Python.

Ví dụ: "I can't do this." -> ["I", "can't", "do", "this", "."]

* Sentence Tokenization: Chia văn bản thành các câu, thường dựa trên các dấu câu như dấu chấm, dấu chấm than, dấu chấm hỏi.

Ví dụ: "Hello world! How are you?" -> ["Hello world!", "How are you?"]

* Subword Tokenization: Chia từ thành các đơn vị con nhỏ hơn, hữu ích cho các ngôn ngữ có cấu trúc từ phức tạp hoặc khi xử lý từ mới, từ ghép.

+ Byte Pair Encoding (BPE): Lặp lại quá trình kết hợp các cặp byte phổ biến nhất để tạo thành các token.

Ví dụ: "low" và "lowest" -> ["low", "est"]

+ WordPiece: Sử dụng trong BERT, tương tự như BPE nhưng với thuật toán khác.

+ Unigram Language Model: Dựa trên mô hình ngôn ngữ để chọn các token tối ưu.

* Character Tokenization: Chia văn bản thành các ký tự riêng lẻ.

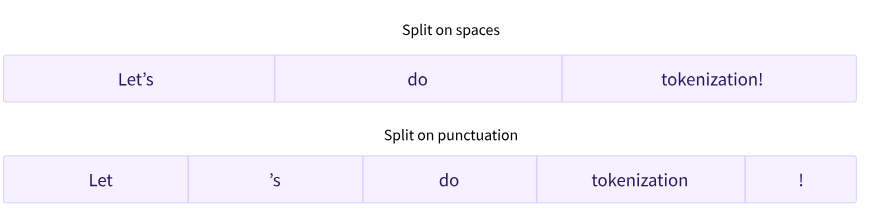
Ví dụ: "hello" -> ["h", "e", "l", "l", "o"]

* Morpheme-based Tokenization: Phân tích từ thành các morpheme (đơn vị ngữ nghĩa nhỏ nhất). Thường được sử dụng cho các ngôn ngữ như tiếng Hàn, tiếng Nhật.
* Dictionary-based Tokenization: Sử dụng từ điển để xác định các từ trong văn bản. Phương pháp này hiệu quả với các ngôn ngữ không có khoảng trắng như tiếng Trung.
* Machine Learning-based Tokenization: sử dụng các mô hình học máy để học cách tokenization từ dữ liệu được gán nhãn.

Ví dụ: Các mô hình neural network hoặc CRF.

### Word-based tokenization

Đây là kĩ thuật tokenization được sử dụng phổ biến trong phân tích văn bản. Nó chia một đoạn văn bản thành các từ (ví dụ tiếng Anh) hoặc âm tiết (ví dụ tiếng Việt) dựa trên dấu phân cách. Dấu phân cách hay được dùng chính là dấu cách trắng. Tuy nhiên, cũng có thể tách văn bản không theo dấu phân cách. Ví dụ tách từ trong tiếng Việt vì một từ trong tiếng Việt có thể chứa 2 hoặc 3 âm tiết được nối với nhau bởi dấu cách trắng.



Hình 1.2 Word-based tokenization

Có nhiều cách khác nhau để tách văn bản. Ví dụ: chúng ta có thể sử dụng khoảng trắng để tokenize văn bản thành các từ bằng cách áp dụng hàm split() của Python:

**tokenized\_text = "Jim Henson was a puppeteer".split()**

**print(tokenized\_text)**

**['Jim', 'Henson', 'was', 'a', 'puppeteer']**

Ngoài ra còn có các biến thể của tokenize mức từ với các quy tắc bổ sung cho dấu câu. Với loại tokenizer này, chúng ta có thể đúc kết với một bộ “từ vựng” khá lớn, trong đó từ vựng được xác định bằng tổng số token độc lập mà chúng ta có trong corpus (kho ngữ liệu) của mình.

Mỗi từ được gán một ID, bắt đầu từ 0 và tăng dần theo kích thước của bộ từ vựng. Mô hình sử dụng các ID này để xác định từng từ. Nếu chúng ta muốn bao phủ hoàn toàn một ngôn ngữ bằng tokenize mức từ, chúng ta sẽ cần phải có một chỉ số nhận dạng cho mỗi từ trong ngôn ngữ, điều này sẽ tạo ra một lượng lớn token.

Ví dụ: có hơn 500,000 từ trong tiếng Anh, vì vậy để xây dựng bản đồ nối mỗi từ đến một ID đầu vào, chúng ta cần theo dõi ngần đó ID. Hơn nữa, các từ như “dog” được biểu diễn khác với các từ như “dogs”, và ban đầu mô hình sẽ không có cách nào để biết rằng “dog” (chó) và “dogs” là tương tự nhau: nó sẽ xác định hai từ này không liên quan. Điều này cũng áp dụng cho các từ tương tự khác, như “run” (chạy) và “running”, mà ban đầu mô hình sẽ không thấy là tương tự.

Cuối cùng, chúng ta cần một token tùy chỉnh để đại diện cho các từ không có trong vốn từ vựng của chúng ta. Mã này được gọi là token “không xác định”, thường được biểu thị là ”[UNK]” hoặc ”<unk>”.

Tách từ có thể được thực hiện dễ dàng bằng cách sử dụng phương thức split () của RegEx hoặc Python. Ngoài ra, có rất nhiều thư viện Python – NLTK, spaCy, Keras, Gensim, có thể giúp bạn thực hiện việc này một cách thuận tiện.

Hạn chế của kỹ thuật này là nó dẫn đến một kho ngữ liệu khổng lồ và một lượng từ vựng lớn, khiến mô hình cồng kềnh hơn và đòi hỏi nhiều tài nguyên tính toán hơn. Bên cạnh đó, một hạn chế nữa là liên quan đến các từ sai chính tả. Nếu kho ngữ liệu có từ “knowledge” viết sai chính tả thành “knowldge”, mô hình sẽ gán token OOV cho từ sau đó. Do đó, để giải quyết tất cả những vấn đề này, các nhà nghiên cứu đã đưa ra kỹ thuật mã hóa dựa trên ký tự. (character-based tokenization)

### Character-based tokenization

Mã hóa dựa trên ký tự chia văn bản thô thành các ký tự riêng lẻ. Logic đằng sau mã hóa này là một ngôn ngữ có nhiều từ khác nhau nhưng có một số ký tự cố định. Điều này dẫn đến một lượng từ vựng rất nhỏ. Có ít token ngoài bộ từ vựng (không xác định) hơn nhiều, vì mọi từ đều có thể được xây dựng từ các ký tự. Nhưng ở đây cũng có một số câu hỏi nảy sinh liên quan đến dấu cách và các dấu câu:



Hình 1.3 Character-based tokenization

Ví dụ tiếng Anh có 256 ký tự khác nhau (chữ cái, số, ký tự đặc biệt) trong khi chứa gần 170.000 từ trong vốn từ vựng. Do đó, mã hóa dựa trên ký tự sẽ sử dụng ít token hơn so với mã hóa dựa trên từ.

Một trong những lợi thế chính của mã hóa dựa trên ký tự là sẽ không có hoặc rất ít từ không xác định hoặc OOV. Do đó, nó có thể biểu diễn các từ chưa biết (những từ không được nhìn thấy trong quá trình huấn luyện) bằng cách biểu diễn cho mỗi ký tự. Một ưu điểm khác là các từ sai chính tả có thể được viết đúng chính tả lại, thay vì có thể đánh dấu chúng là mã thông báo OOV và làm mất thông tin.

Loại mã hóa này khá đơn giản và có thể làm giảm độ phức tạp của bộ nhớ và thời gian. Ngoài ra, tuy kỹ thuật này giúp giảm kích thước từ vựng nhưng lại làm tăng độ dài chuỗi trong mã hóa dựa trên ký tự. Mỗi từ được chia thành từng ký tự và do đó, chuỗi mã hóa dài hơn nhiều so với văn bản thô ban đầu. Vì vậy, có thể thấy, dù đã giải quyết được rất nhiều thách thức mà mã hóa dựa trên từ gặp phải, mã hóa dựa trên ký tự vẫn có một số vấn đề nhất định. (Subword-based tokenization)

### Subword-based tokenization

Một kỹ thuật phổ biến khác là mã hóa dựa trên từ khóa phụ. Đây là một giải pháp nằm giữa mã hóa dựa trên từ và ký tự. Ý tưởng chính là giải quyết đồng thời các vấn đề của mã hóa dựa trên từ (kích thước từ vựng rất lớn, có nhiều tokens OOV, sự khác biệt trong ý nghĩa của các từ rất giống nhau) và mã hóa dựa trên ký tự (chuỗi rất dài và token riêng lẻ ít ý nghĩa hơn).

Các thuật toán Subword-based tokenization dựa trên nguyên tắc rằng các từ được sử dụng thường xuyên không được chia thành các từ phụ nhỏ hơn, nhưng các từ hiếm phải được phân tách thành các từ phụ có ý nghĩa.

Ví dụ: “annoyingly” (khó chịu) có thể được coi là một từ hiếm và có thể được chuyển thành “annoying” và “ly”. Cả hai đều có khả năng xuất hiện thường xuyên hơn dưới dạng các từ phụ độc lập, đồng thời nghĩa của “annoying” được giữ nguyên bởi nghĩa kết hợp của “annoying” và “ly”.

Dưới đây là một ví dụ cho thấy cách một thuật toán Subword-based tokenization sẽ tokenize chuỗi “Let’s do tokenization!”:



Hình 1.4 Subword-based tokenization

Những từ phụ này cung cấp rất nhiều ý nghĩa về mặt ngữ nghĩa

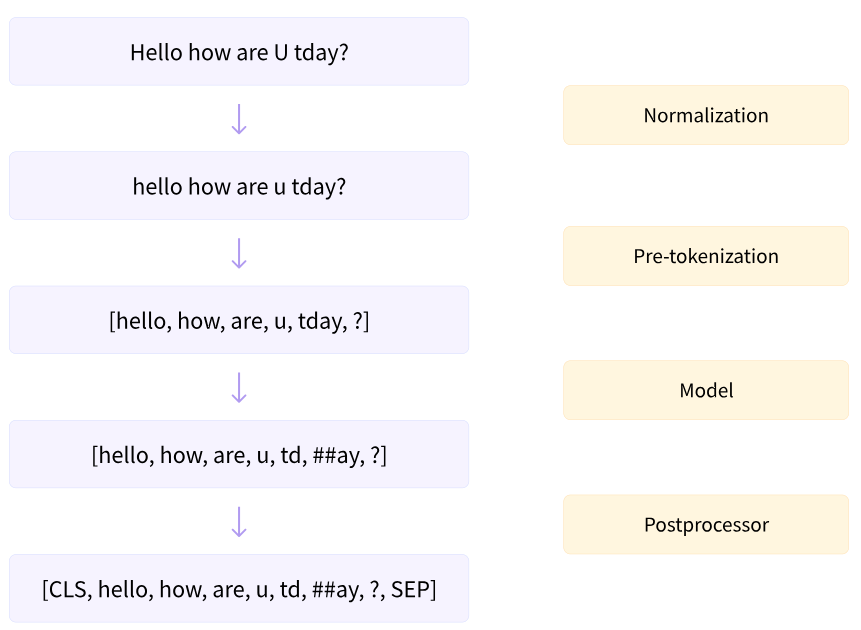
Ví dụ: trong ví dụ ở trên “tokenization” được chia thành “token” và “ization”, hai token đều có ý nghĩa về mặt ngữ nghĩa đồng thời tiết kiệm không gian (chỉ cần hai token để biểu thị một từ dài). Điều này cho phép chúng ta có thể bao quát tương đối tốt với các từ vựng nhỏ và gần như không có token nào không xác định.

* Thuật toán mã hóa dựa trên từ (word-based tokenization algorithm) sẽ chia câu thành các từ: [“Let”, “us”, “learn”, “tokenization.”]
* Thuật toán mã hóa dựa trên từ phụ (subword-based tokenization algorithm) sẽ chia câu thành các từ khóa phụ: [“Let”, “us”, “learn”, “token”, “ization.”]
* Thuật toán mã hóa dựa trên ký tự (character-based tokenization algorithm) sẽ chia câu thành các ký tự, ở đây là từng chữ cái một.

Hầu hết các mô hình tiếng Anh đều sử dụng các dạng thuật toán của mã hóa từ phụ, trong đó, phổ biến là WordPiece được sử dụng bởi BERT và DistilBERT, Unigram của XLNet và ALBERT, và Bye-Pair Encoding của GPT-2 và RoBERTa.

## Chuẩn hoá và tiền tokenize

Trước khi đi sâu hơn vào ba thuật toán tokenize từ phụ phổ biến nhất được sử dụng với các mô hình Transformer (Mã hóa theo cặp [BPE], WordPiece và Unigram), trước tiên chúng ta sẽ xem xét tiền xử lý mà mỗi trình tokenize áp dụng cho văn bản. Dưới đây là tổng quan cấp cao về các bước trong pipeline tokenize:



Hình 1.5 Chuẩn hoá và tiền tokenize

Trước khi tách một đoạn văn bản thành các token phụ (dựa theo mô hình)tokenizer sẽ thực hiện 2 bước: normalization (chuẩn hoá) và pre-tokenization (tiền tokenize).

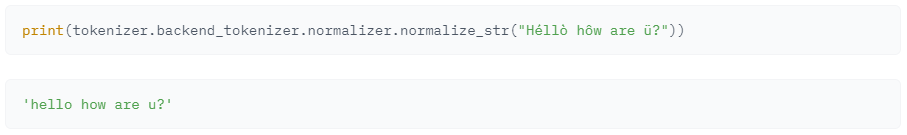
### Normalization

Bước chuẩn hóa bao gồm một số thao tác dọn dẹp, chẳng hạn như loại bỏ khoảng trắng không cần thiết, viết thường tất cả các chữ, và/hoặc xóa dấu. Nếu bạn đã quen với chuẩn hóa Unicode (chẳng hạn như NFC hoặc NFKC), thì đây cũng là điều mà tokenizer có thể áp dụng.

Tokenizer của Transformers có một thuộc tính gọi là backend\_tokenizer cung cấp quyền truy cập vào tokenizer bên dưới từ thư viện Tokenizers:



Thuộc tính normalizer của đối tượng tokenizer có phương thức normalize\_str() mà ta có thể dùng để thấy cách bước chuẩn hoá được thực hiện:

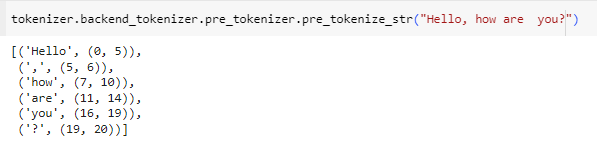


Trong ví dụ này, vì chúng ta chọn checkpoint bert-base-uncased, bước chuẩn hoá sẽ thực hiện viết thường và loại bỏ các dấu.

### Pre-tokenization

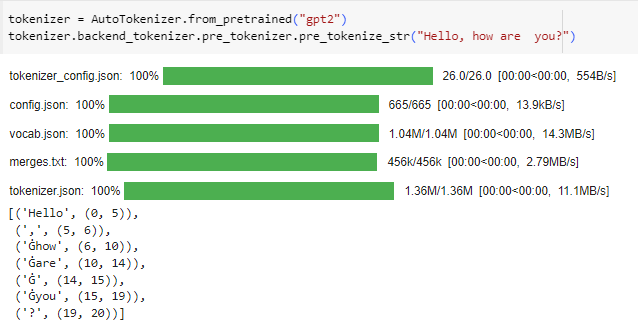
Như chúng ta sẽ thấy trong các phần tiếp theo, một tokenizer không thể được huấn luyện trên văn bản thô. Thay vào đó, trước tiên chúng ta cần chia các văn bản thành các thực thể nhỏ, như các từ. Đó là khi bước pre-tokenization bắt đầu. Như chúng ta đã thấy trong Chương 2, trình tokenize dựa trên từ có thể chỉ cần tách một văn bản thô thành các từ dựa trên khoảng trắng và dấu câu. Những từ đó sẽ là ranh giới của các token con mà tokenizer có thể học được trong quá trình huấn luyện của nó.

Để xem cách một tokenizer nhanh thực hiện pre-tokenization, chúng ta có thể sử dụng phương thức pre\_tokenize\_str() của thuộc tính pre\_tokenizer của đối tượng tokenizer:



Lưu ý cách tokenizer đã theo dõi các offset, đó là cách nó có thể cung cấp cho chúng ta ánh xạ offset mà ta đã sử dụng trong phần trước. Ở đây tokenizer bỏ qua hai khoảng trắng và thay thế chúng bằng chỉ một, nhưng các offset xen giữa are và you để giải thích điều đó.

Vì chúng ta đang sử dụng BERT tokenizer, pre-tokenization liên quan đến việc phân tách dựa trên khoảng trắng và dấu chấm câu. Các tokenizer khác có thể có các quy tắc khác nhau cho bước này. Ví dụ: nếu sử dụng GPT-2 tokenizer và nó sẽ tách dựa trên dấu cách và dấu câu, nhưng sẽ giữa dấu cách và thay thế chúng bởi kí hiệu Ġ, cho phép nó khôi phục không gian ban đầu nếu chúng tôi giải mã các token:



Cần lưu ý thêm rằng không như BERT tokenizer, tokenizer này bỏ qua dấu cách kép. Ở ví dụ cuối, hãy cùng xem T5 tokenizer dựa trên thuật toán SentencePiece:



Giống như GPT-2 tokenizer, phương pháp này giữ các dấu cách và thay thế chúng bởi một tí tự đặc biệt (\_), nhưng T5 tokenizer chỉ tách dựa theo dấu cách, không dựa theo dấu câu. Một lưu ý nữa đó là nó cũng mặc định thêm dấu cách ở phía đầu câu (trước Hello) và bỏ qua những dấu cách kẹp ở giữa are và you.

Bây giờ chúng ta đã biết một chút về cách một số loại tokenizers khác nhau để xử lý văn bản, chúng ta có thể bắt đầu tự khám phá các thuật toán cơ bản. Chúng ta sẽ bắt đầu bằng một cái nhìn nhanh về SentencePiece được áp dụng rộng rãi; sau đó, trong ba phần tiếp theo, chúng ta sẽ xem xét cách thức hoạt động của ba thuật toán chính được sử dụng để mã hóa từ phụ.

# LÝ THUYẾT BPE

## Tổng quan

Trong các phần tiếp theo, chúng ta sẽ đi sâu vào ba thuật toán tokenize từ phụ tiêu biểu: BPE (được sử dụng bởi GPT-2 và các thuật toán khác), WordPiece (được sử dụng bởi BERT) và Unigram (được sử dụng bởi T5 và các thuật toán khác). Trước khi chúng ta bắt đầu, đây là tổng quan nhanh về cách hoạt động của từng loại. Đừng ngần ngại quay lại bảng này sau khi đọc từng phần tiếp theo nếu bạn chưa hiểu hết.

|  |  |  |  |
| --- | --- | --- | --- |
| Mô hình | BPE | WordPiece | Unigram |
| Huấn luyện | Bắt đầu với một bộ từ vựng nhỏ và học bộ quy tắc hợp nhất token | Bắt đầu với một bộ từ vựng nhỏ và học bộ quy tắc hợp nhất token | Bắt đầu với một bộ từ vựng lớn và học bộ quy tắc để loại bỏ token |
| Bước huấn luyện | Gộp các token liên quan đến cặp phổ biến nhất | Gộp các token liên quan đến cặp có điểm cao nhất dựa trên tần suất của cặp, with the best score based on the frequency of the pair, ưu tiên các cặp mà mỗi token cá nhân tần suất thấp hơn | Loại bỏ tất cả các token trong bộ từ điển giảm thiểu tối đa độ mất mát được tính trên toàn bộ kho ngữ liệu |
| Học | Gộp bộ quy tắc và bộ từ vựng | Chỉ bộ từ vựng | Một bộ từ vựng với điểm cho mỗi token |
| Mã hoá | Chia từ thành các kí tự và áp dụng bước gộp từ quá trình huấn luyện | Tìm ra chuỗi từ phụ dài nhất bắt đầu từ phần bắt đầu có trong bộ từ vựng, sau đó làm tương tự với các phần còn lại của từ | Tìm từ có khả năng chia thành token cao nhất sử dụng điểm có được từ quá trình huấn luyện |

## Khái niệm về BPE (Byte-Pair Encoding)

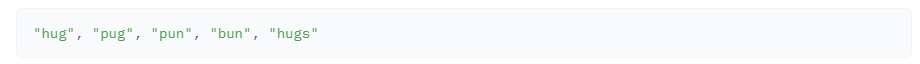
Cách đơn giản nhất đó là tách câu theo từ tuy nhiên cách này có một điểm yếu rất lớn. Đó là vấn đề từ không biết (unknown words) và để khắc phục vấn đề này chúng ta sẽ sử dụng thuật toán Byte-Pair Encoding, viết tắt là BPE.

Byte-Pair Encoding (BPE) là một thuật toán nén dữ liệu và biểu diễn từ vựng được sử dụng phổ biến trong các mô hình xử lý ngôn ngữ tự nhiên (NLP). BPE hoạt động bằng cách lặp lại việc tìm và thay thế cặp byte xuất hiện nhiều nhất trong văn bản với một token duy nhất. Điều này giúp giảm số lượng token cần thiết để biểu diễn văn bản, đồng thời cải thiện hiệu quả học của các mô hình NLP.

Nó được sử dụng bởi rất nhiều mô hình Transformer, bao gồm GPT, GPT-2, RoBERTa, BART và DeBERTa.

## Thuật toán huấn luyện BPE

Huấn luyện BPE bắt đầu bằng cách tính toán tập hợp các từ duy nhất được sử dụng trong kho ngữ liệu (sau khi hoàn thành các bước chuẩn hóa và pre-tokenization), sau đó xây dựng từ vựng bằng cách lấy tất cả các ký hiệu được sử dụng để viết những từ đó. Ví dụ rất đơn giản, giả sử kho dữ liệu của chúng ta sử dụng năm từ sau:



Từ vựng cơ sở khi đó sẽ là ["b", "g", "h", "n", "p", "s", "u"]. Đối với các trường hợp trong thực tế, từ vựng cơ sở đó sẽ chứa tất cả các ký tự ASCII, ít nhất và có thể là một số ký tự Unicode. Nếu một mẫu bạn đang tokenize sử dụng một ký tự không có trong kho dữ liệu huấn luyện, thì ký tự đó sẽ được chuyển đổi thành token không xác định. Đó là một lý do tại sao nhiều mô hình NLP rất kém trong việc phân tích nội dung bằng biểu tượng cảm xúc.

GPT-2 và RoBERTa tokenizer (khá giống nhau) có một cách thông minh để giải quyết vấn đề này: chúng không xem các từ được viết bằng các ký tự Unicode mà là các byte. Bằng cách này, từ vựng cơ sở có kích thước nhỏ (256), nhưng mọi ký tự bạn có thể nghĩ đến sẽ vẫn được bao gồm và không bị chuyển đổi thành token không xác định. Thủ thuật này được gọi là BPE cấp byte.

Sau khi có được bộ từ vựng cơ bản này, chúng ta thêm các token mới cho đến khi đạt được kích thước từ vựng mong muốn bằng cách học hợp nhất, đây là các quy tắc để hợp nhất hai yếu tố của từ vựng hiện có với nhau thành một từ mới. Vì vậy, lúc đầu sự hợp nhất này sẽ tạo ra các token có hai ký tự và sau đó, khi quá trình huấn luyện tiến triển, các từ phụ sẽ dài hơn.

Tại bất kỳ bước nào trong quá trình huấn luyện token, thuật toán BPE sẽ tìm kiếm cặp token hiện có thường xuyên nhất (theo “cặp”, ở đây có nghĩa là hai token liên tiếp trong một từ). Cặp thường xuyên nhất đó là cặp sẽ được hợp nhất, và chúng ta xả và lặp lại cho bước tiếp theo.

Quay trở lại ví dụ trước, giả sử các từ có tần số như sau:



nghĩa là "hug" có mặt 10 lần trong kho ngữ liệu, "pug" 5 lần, "pun" 12 lần, "bun" 4 lần và "hug" 5 lần. Chúng ta bắt đầu huấn luyện bằng cách tách từng từ thành các ký tự (những ký tự hình thành từ vựng ban đầu của chúng ta) để có thể xem mỗi từ như một danh sách các token:

 Sau đó, chúng ta xem xét các cặp. Cặp ("h", "u") có trong các từ "hug" và "hugs", vì vậy tổng cộng là 15 lần trong ngữ liệu. Tuy nhiên, đây không phải là cặp thường xuyên nhất: vinh dự đó thuộc về ("u", "g"), có trong "hug", "pug", và "hugs", với tổng cộng 20 lần xuất hiện trong bộ từ vựng.

Do đó, quy tắc hợp nhất đầu tiên được học bởi tokenizer là ("u", "g") -> "ug", có nghĩa là "ug" sẽ được thêm vào từ vựng và cặp này sẽ được hợp nhất trong tất cả các từ của ngữ liệu. Vào cuối giai đoạn này, từ vựng và ngữ liệu sẽ giống như sau:



Bây giờ chúng ta có một số cặp dẫn đến một token dài hơn hai ký tự: ví dụ: cặp ("h", "ug"), (hiện diện 15 lần trong kho ngữ liệu). Cặp thường gặp nhất ở giai đoạn này là ("u", "n"), xuất hiện 16 lần trong kho ngữ liệu, vì vậy quy tắc hợp nhất thứ hai đã học là ("u", "n") -> "un". Thêm nó vào bộ từ vựng và hợp nhất tất cả các lần xuất hiện hiện có sẽ dẫn chúng ta đến:



Giờ thì cặp xuất hiện nhiều nhất là ("h", "ug"), nên chúng ta hợp nhất ("h", "ug") -> "hug", trả về cho chúng ta token gồm ba ký tự đầu tiên. Sau sự hợp nhất này, kho ngữ liệu sẽ như sau:

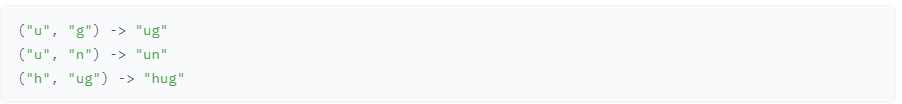


## Thuật toán tokenize BPE

Tokenize tuân thủ chặt chẽ quá trình huấn luyện, theo nghĩa là các đầu vào mới được tokenize bằng cách áp dụng các bước sau:

* Chuẩn hoá
* Pre-tokenization
* Tách các từ thành các ký tự riêng lẻ
* Áp dụng các quy tắc hợp nhất đã học theo thứ tự trên các phần tách đó

Lấy ví dụ mà ta đã sử dụng trong quá trình huấn luyện, với ba quy tắc hợp nhất đã học:



Từ "bug" sẽ được tokenize thành ["b", "ug"]. "mug", tuy nhiên, sẽ tokenize thành ["[UNK]", "ug"] vì kí tự "m" không có trong bộ tự vựng gốc. Tương tự, từ "thug" sẽ được tokenize thành ["[UNK]", "hug"]: kí tự "t" không có trong bộ tự vựng gốc, và áp dụng quy tắc hợp nhất ở "u" và "g" và sau đó "hu" và "g".

RoBERTa (Robustly optimized BERT approach)

Mô tả: RoBERTa là một biến thể của mô hình BERT, được tối ưu hóa bằng cách huấn luyện trên nhiều dữ liệu hơn và với các kỹ thuật tăng cường dữ liệu khác nhau.

Ứng dụng: RoBERTa được sử dụng trong các tác vụ như phân loại văn bản, trả lời câu hỏi, và phân tích ngữ nghĩa.

Tokenizer: RoBERTa sử dụng BPE để token hóa văn bản, giúp cải thiện khả năng học của mô hình trên các từ và cụm từ phức tạp.

T5 (Text-to-Text Transfer Transformer)

Mô tả: T5 là một mô hình ngôn ngữ tổng quát có thể chuyển đổi từ đầu vào văn bản này sang đầu ra văn bản khác, áp dụng cho nhiều tác vụ NLP như dịch máy, tóm tắt văn bản, và trả lời câu hỏi.

Ứng dụng: T5 được sử dụng rộng rãi trong nhiều nhiệm vụ NLP khác nhau nhờ khả năng chuyển đổi linh hoạt giữa các nhiệm vụ.

Tokenizer: T5 sử dụng BPE để token hóa văn bản, giúp tối ưu hóa quá trình biểu diễn văn bản đầu vào.

## Một số kĩ thuật khác

### WordPiece tokenization

WordPiece là một thuật toán tokenize được Google phát triển để huấn luyện trước BERT. Nó đã được tái sử dụng trong một vài mô hình Transformer dựa trên BERT, như DistilBERT, MobileBERT, Funnel Transformers, và MPNET. Nó khá tương tự với BPE về mặt huấn luyện, nhưng tokenize thực sự được thực hiện hoàn toàn khác.

**Thuật toán huấn luyện**

Giống như BPE, WordPiece bắt đầu từ một từ vựng nhỏ bao gồm các token đặc biệt được sử dụng bởi mô hình và bảng chữ cái đầu tiên. Vì nó xác định các từ phụ bằng cách thêm tiền tố (như ## cho BERT), ban đầu mỗi từ được tách bằng cách thêm tiền tố đó vào tất cả các ký tự bên trong từ. Vì vậy, ví dụ, "word" được chia như thế này: w ##o ##r ##d. Vì vậy, bảng chữ cái chứa tất cả các kí tự xuất hiện ở đầu của một từ và các kí tự xuất hiện bên trong của từ được thêm một tiền tố của WordPiece phía trước.

Sau đó, một lần nữa, giống như BPE, WordPiece học các quy tắc hợp nhất. Sự khác biệt chính là cách chọn cặp được hợp nhất. Thay vì chọn cặp phổ biến nhất, WordPiece tính điểm cho từng cặp, sử dụng công thức sau:



Bằng cách chia tần suất của cặp cho tích tần suất của từng con của nó, thuật toán ưu tiên hợp nhất các cặp mà các bộ phận riêng lẻ ít thường xuyên hơn trong từ vựng. Ví dụ: nó sẽ không nhất thiết phải hợp nhất ("un", "##able") ngay cả khi cặp đó xuất hiện rất thường xuyên trong từ vựng, vì hai cặp "un" và "##able" mỗi từ có thể sẽ xuất hiện bằng nhiều từ khác và có tần suất cao. Ngược lại, một cặp như ("hu", "##gging") có thể sẽ được hợp nhất nhanh hơn (giả sử từ “hugging” xuất hiện thường xuyên trong từ vựng) vì "hu" và "##gging"có khả năng ít xuất hiện hơn với từng cá thể.

Hãy cùng nhìn vào cùng bộ từ vựng chúng ta sử dụng cho BPE:



Nó sẽ được chia ra như sau:

("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5) vì vậy từ vựng ban đầu sẽ là ["b", "h", "p", "##g", "##n", "##s", "##u"] (nếu ta tạm quên các token đặc biệt). Cặp thường gặp nhất là ("##u", "##g") (xuất hiện 20 lần), nhưng tần suất xuất hiện riêng của "##u" rất cao, vì vậy điểm của nó không phải là cao nhất (đó là 1 / 36). Tất cả các cặp có "##u"thực sự có cùng điểm (1 / 36), vì vậy điểm tốt nhất thuộc về cặp ("##g", "##s") — cặp duy nhất không có "##u" — là 1 / 20, và phép hợp nhất đầu tiên đã học là ("##g", "##s") -> ("##gs").

Lưu ý rằng khi hợp nhất, chúng ta loại bỏ ## giữa hai token, vì vậy chúng ta thêm "##gs" vào từ vựng và áp dụng hợp nhất trong các từ của ngữ liệu:

* Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]
* Corpus: ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)

Tại thời điểm này, "##u" nằm trong tất cả các cặp có thể có, vì vậy tất cả chúng đều có cùng điểm. Giả sử trong trường hợp này, cặp đầu tiên được hợp nhất, vì vậy ("h", "##u") -> "hu". Điều này đưa chúng ta đến:

* Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu"]
* Corpus: ("hu" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)

Sau đó, điểm số tốt nhất tiếp theo được chia sẻ bởi ("hu", "##g") and ("hu", "##gs") (với 1/15, so với 1/21 của các cặp khác), vì vậy cặp đầu tiên có điểm lớn nhất được hợp nhất:

* Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]
* Corpus: ("hug", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)

và tiếp tục như vậy cho đến khi chúng ta đạt được kích thước bộ từ vựng mong muốn.

## Sự khác nhau giữa BPE và WordPiece

Tokenize của WordPiece khác BPE ở chỗ WordPiece chỉ lưu từ vựng cuối cùng, không lưu các quy tắc hợp nhất đã học. Bắt đầu từ từ cần tokenize, WordPiece tìm từ con dài nhất có trong từ vựng, sau đó tách từ đó ra. Ví dụ: nếu chúng ta sử dụng từ vựng đã học trong ví dụ trên, đối với từ "hugs" từ phụ dài nhất bắt đầu từ đầu mà bên trong từ vựng là "hug", vì vậy chúng ta tách ở đó và nhận được ["hug","##s"]. Sau đó, chúng ta tiếp tục với "##s", trong từ vựng, vì vậy tokenize của "hugs" là ["hug","##s"].

Với BPE, chúng ta sẽ áp dụng các phép hợp nhất đã học theo thứ tự và token điều này thành ["hu", "##gs"], do đó, cách mã hoá sẽ khác.

Ví dụ khác, hãy xem từ "bugs" sẽ được tokenize như thế nào. "b" là từ phụ dài nhất bắt đầu từ đầu của từ có trong từ vựng, vì vậy chúng tôi tách ở đó và nhận được ["b", "##ugs"]. Sau đó, "##u" là từ con dài nhất bắt đầu ở đầu "##ugs" có trong từ vựng, vì vậy chúng ta tách ở đó và nhận được ["b", "##u, "##gs"]. Cuối cùng, "##gs" có trong từ vựng, vì vậy danh sách cuối cùng này là mã hóa của "bugs".

Khi quá trình tokenize đến giai đoạn không thể tìm thấy một từ khóa phụ trong từ vựng, toàn bộ từ được tokenize thành không xác định - vì vậy, ví dụ: "mug" sẽ được tokenize là ["[UNK]"], cũng như "bum" (ngay cả khi chúng ta có thể bắt đầu bằng "b" và "##u", "##m" không phải thuộc bộ từ vựng và kết quả tokenize sẽ chỉ là ["[UNK]"], không phải ["b", "##u", "[UNK]"]). Đây là một điểm khác biệt so với BPE, chỉ phân loại các ký tự riêng lẻ không có trong từ vựng là không xác định.

# TÀI LIỆU THAM KHẢO

[1] "Natural Language Processing with Python" by Steven Bird, Ewan Klein, and Edward Loper

[2] "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition" by Daniel Jurafsky and James H. Martin

[3] "Foundations of Statistical Natural Language Processing" by Christopher D. Manning and Hinrich Schütze

[4] "Deep Learning for Natural Language Processing" by Palash Goyal, Sumit Pandey, Karan Jain, and Karthik Raman

[5] "Introduction to Information Retrieval" by Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze