

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC KINH TẾ TP. HỒ CHÍ MINH
TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ



ĐỒ ÁN MÔN HỌC
DỮ LIỆU LỚN VÀ ỨNG DỤNG

**ĐỀ TÀI: Xây dựng hệ thống hỏi đáp sử dụng BERT và FAISS trên data
SQuADv2**

Giảng viên hướng dẫn: ThS. Nguyễn Mạnh Tuấn

Học phần: Dữ liệu lớn và ứng dụng

Nhóm sinh viên: Nhóm 6

Họ tên SV	MSSV
Phạm Minh Hiền	31211021654
Phan Thị Thanh Hoa	31211027176
Nguyễn Phú Hưng	31211023521
Trương Thị Hồng Mai	31211026726
Đào Thị Phương Quỳnh	31211027667
Vũ Nguyễn Thảo Vi	31211027686
Phan Dương Hoàng Vũ	31211022533

TP Hồ Chí Minh, ngày 29 tháng 5 năm 2024

MỤC LỤC

DANH MỤC HÌNH ẢNH	4
DANH MỤC BẢNG BIỂU	5
DANH MỤC TỪ VIẾT TẮT	6
LỜI CẢM ƠN	8
CHƯƠNG 1: TỔNG QUAN.....	9
1.1. Lý do chọn đề tài.....	9
1.1.1. Giới thiệu về Big Data và sự bùng nổ của Big Data	9
1.1.2. Tầm quan trọng của việc phân tích và xử lý Big Data tại các doanh nghiệp	12
1.2. Mục tiêu của đề tài	14
1.3. Đối tượng và phạm vi nghiên cứu.....	14
1.4. Phương pháp thực hiện	14
1.5. Bố cục của đề tài	14
1.6. Phân công công việc	15
CHƯƠNG 2: ỨNG DỤNG DỮ LIỆU LỚN Ở FACEBOOK.....	17
2.1. Giới thiệu về Facebook	17
2.2. Dữ liệu lớn từ Facebook	17
2.3. Các công nghệ và nền tảng xử lý dữ liệu lớn được sử dụng tại Facebook	19
2.3.1. Scribe	19
2.3.2. Puma	21
2.3.3. Stylus	22
2.3.4. Swift	22
2.3.5. Laser	23
2.3.6. Scuba	23
2.3.7. Hive	26
2.3.8. Apache Hadoop	27
2.3.9. HDFS	29
2.3.10. HBase	31
2.4. Cài đặt và sử dụng HBase	33
2.4.1. Mô tả.....	33
2.4.2. Cài đặt.....	33
CHƯƠNG 3: ỨNG DỤNG DỮ LIỆU LỚN Ở SPOTIFY	39
3.1. Giới thiệu doanh nghiệp Spotify	39
3.2. Quy trình thu thập dữ liệu ở Spotify	41
3.3. Quy trình xử lý và lưu trữ dữ liệu ở Spotify	41
3.4. Cài đặt và sử dụng MapReduce	44

3.4.1. Mô tả.....	44
3.4.2. Cài đặt.....	44
CHƯƠNG 4: CƠ SỞ LÝ THUYẾT VÀ Q&A.....	48
4.1. Tokenization.....	48
4.2. Cơ chế Attention Mechanism và Transformers	49
4.3. Mô hình BERT.....	51
4.3.1 Giới thiệu về mô hình BERT.....	51
4.3.1.1. Định nghĩa.....	51
4.3.1.2. Hướng xử lý ngôn ngữ hai chiều của BERT (bidirectional approach).....	51
4.3.1.3. Kiến trúc của BERT	52
4.3.2. Cách hoạt động của mô hình BERT	52
4.3.2.1.1. Text - preprocessing.....	52
4.3.2.1.2. Pre-training	54
4.3.3. Đánh giá và ứng dụng của mô hình BERT	56
4.3.3.1. Ưu điểm	56
4.3.3.2. Nhược điểm.....	56
4.3.3.3. Ứng dụng	57
4.4. XLM-RoBERTa.....	57
4.5. RoBERTa	58
4.6 Độ đo Exact Match và F1	59
4.7. Mô hình Question & Answering (QA Model).....	60
4.7.1 Mô hình hỏi đáp - Question & Answering Model	60
4.7.1.1. Định nghĩa.....	60
4.7.1.2. Ứng dụng	60
4.7.1.3. Phân loại.....	61
4.7.2. Extractive Question & Answering	61
4.7.2.1 Tìm hiểu mô hình.....	61
4.7.2.2 Đánh giá mô hình.....	62
4.7.3. Closed Generative Question & Answering	63
4.7.3.1 Tìm hiểu mô hình.....	63
4.7.3.2 Đánh giá mô hình.....	64
CHƯƠNG 5: GIỚI THIỆU BỘ DỮ LIỆU.....	65
5.1. Giới thiệu bộ dữ liệu	65
5.2. Phân tích dữ liệu khám phá (EDA).....	66
CHƯƠNG 6: XÂY DỰNG MÔ HÌNH & GIAO DIỆN	73
6.1. Thiết lập thực nghiệm	73
6.2. Đánh giá kết quả mô hình.	73

6.3. Xây dựng hệ thống QA System	74
6.3.1. Giao diện	74
6.3.2. Extractive Q&A.....	75
6.3.3. Closed Generative Q&A	77
CHƯƠNG 7: KẾT LUẬN.....	80
TÀI LIỆU THAM KHẢO.....	81
PHỤ LỤC	84

DANH MỤC HÌNH ẢNH

Hình 1.1: Các giai đoạn phát triển của Big Data (Mohammad Shahid Husain et al., 2023)	9
Hình 1.2: Thống kê dữ liệu được tạo ra trong từng phút của năm 2023 (Heitman, 2022)	11
Hình 1.3: Thống kê dữ liệu được tạo ra từ năm 2010-2020 và dự đoán cho giai đoạn 2021-2025 (Statista, 2023)	12
Hình 2.1: Hệ thống xử lý dữ liệu realtime của Facebook. Nguồn: Chen et al. (2016)	18
Hình 2.2: Kiến trúc hệ thống Scuba. Nguồn: (Abraham et al., 2013)	24
Hình 2.3: Kiến trúc HDFS. Nguồn: (Ryan, 2012)	31
Hình 3.1: Quy trình thu thập dữ liệu của Spotify (Baer, 2015)	41
Hình 3.2: Quy trình xử lý dữ liệu ở Spotify (Mishra & Brown, 2015)	43
Hình 3.3: Sơ đồ minh họa hệ thống đề xuất cho người dùng của Spotify (Baer, 2015)	44
Hình 4.1: Minh họa cách tạo thành của các vector biểu diễn ngữ cảnh	54
Hình 4.2: Minh họa cách nhận diện Next Sentence Prediction	55
Hình 4.3: Cơ chế hoạt động của mô hình hỏi đáp trích xuất	61
Hình 4.4: Cơ chế hoạt động của mô hình hỏi đáp tạo sinh đóng	63
Hình 5.1: Số lượng bài viết, đoạn văn và câu hỏi trong bộ dữ liệu SQuADv2	66
Hình 5.2: Phân phối câu hỏi có câu trả lời và câu hỏi không có câu trả lời	67
Hình 5.3: Phân tích câu hỏi không có câu trả lời	68
Hình 5.4: Phân tích câu hỏi có câu trả lời	69
Hình 5.5: Phân phối độ dài đoạn văn và câu trả lời	70
Hình 5.6: Phân phối số câu hỏi mỗi đoạn văn và số câu hỏi mỗi bài viết	72
Hình 6.1: UI Demo - Extractive Q&A	75
Hình 6.2: UI Demo - Closed Generative Q&A	75
Hình 6.3: Pipeline cho mô hình Extractive Q&A	76
Hình 6.4: Ví dụ về context và câu hỏi	77
Hình 6.5: Ví dụ về câu trả lời và chỉ số	77
Hình 6.6: Tổng quan quy trình thực hiện	78
Hình 6.7: Tổng quan quy trình thực hiện - các bước	78

DANH MỤC BẢNG BIỂU

Bảng 1.1 Chú giải về các giai đoạn phát triển của Big Data (Mohammad Shahid Husain et al., 2023)

8

DANH MỤC TỪ VIẾT TẮT

STT	KÝ HIỆU CHỮ VIẾT TẮT	CHỮ VIẾT ĐẦY ĐỦ
1	NLP	Natural Language Processing
2	RDBMS	Relational Database Management System
3	SQL	Structured Query Language
4	IP	Internet Protocol
5	IoT	Internet of Things
6	IIoT	Industrial Internet of Things
7	IoMT	Internet of Medical Things
8	BSN	Body Sensor Network
9	RPA	Robotic Process Automation
10	HDFS	Hadoop Distributed File System
11	BERT	Bidirectional Encoder Representations from Transformers
12	FAISS	Facebook AI Similarity Search
13	UDFs	User-defined Function
14	API	Application Programming Interface
15	DAG	Directed Acyclic Graph
16	HDD	Hard Disk Drive

17	SSD	Solid State Drive
18	RAM	Random-access Memory
19	GUI	Graphical User Interface
20	CPU	Central Processing Unit
21	UDAF	User Defined Aggregate Functions
22	UI	User Interface
23	YARN	Yet Another Resource Negotiator
24	AWS	Amazon Web Services
25	AP+KSP	Access Points + Kafka Syslog Producers
26	KSC	Kafka Syslog Collectors
27	RNN	Recurrent Neural Networks
28	CNN	Convolutional Neural Networks
29	CLS	Classifier
30	SEP	Separator
31	ID	Identification
32	BPE	Byte Pair Encoding
33	GB	Gigabyte
34	NSP	Next Sentence Prediction

35	EM	Exact Match
36	QA/ Q&A	Question & Answering
37	TF-IDF	Term Frequency-Inverse Document Frequency
38	EDA	Exploratory Data Analysis
39	slf4j	Simple Logging Facade for Java

LỜI CẢM ƠN

Trước hết, chúng em xin bày tỏ lòng biết ơn sâu sắc đến Đại học Kinh Tế TP.HCM vì đã đưa môn Dữ liệu lớn và Ứng dụng vào chương trình đào tạo, tạo cơ hội cho chúng em tiếp thu kiến thức chuyên môn quý giá.

Đặc biệt, chúng em xin gửi lời tri ân chân thành đến Thầy Nguyễn Mạnh Tuấn - người đã truyền cảm hứng và dìu dắt chúng em trong suốt hành trình học tập. Nhờ sự tận tâm và tâm huyết của Thầy, chúng em không chỉ nắm vững nền tảng lý thuyết mà còn được trang bị những kinh nghiệm thực tiễn vô cùng giá trị, là hành trang vững chắc cho con đường sự nghiệp tương lai.

Tuy nhiên, do khả năng còn hạn chế và gặp nhiều khó khăn trong việc áp dụng kiến thức vào thực tế, bài báo cáo của nhóm em tuy đã nỗ lực và cố gắng hoàn thành nhưng vẫn có thể còn tồn tại một số thiếu hụt và sai sót, đôi khi là những khái niệm chưa được nắm rõ ràng. Chúng em rất mong nhận được sự góp ý quý báu của Thầy để hoàn thiện bài báo cáo của mình một cách hoàn thiện và quy chuẩn hơn.

Chúng em xin chân thành cảm ơn Thầy!

CHƯƠNG 1: TỔNG QUAN

1.1. Lý do chọn đề tài

1.1.1. Giới thiệu về Big Data và sự bùng nổ của Big Data

Big data là khái niệm để chỉ dữ liệu có kích thước lớn và bao gồm những tập dữ liệu phức tạp. Big data không phải một khái niệm mới xuất hiện gần đây mà đã có hẳn một lịch sử hình thành và phát triển.



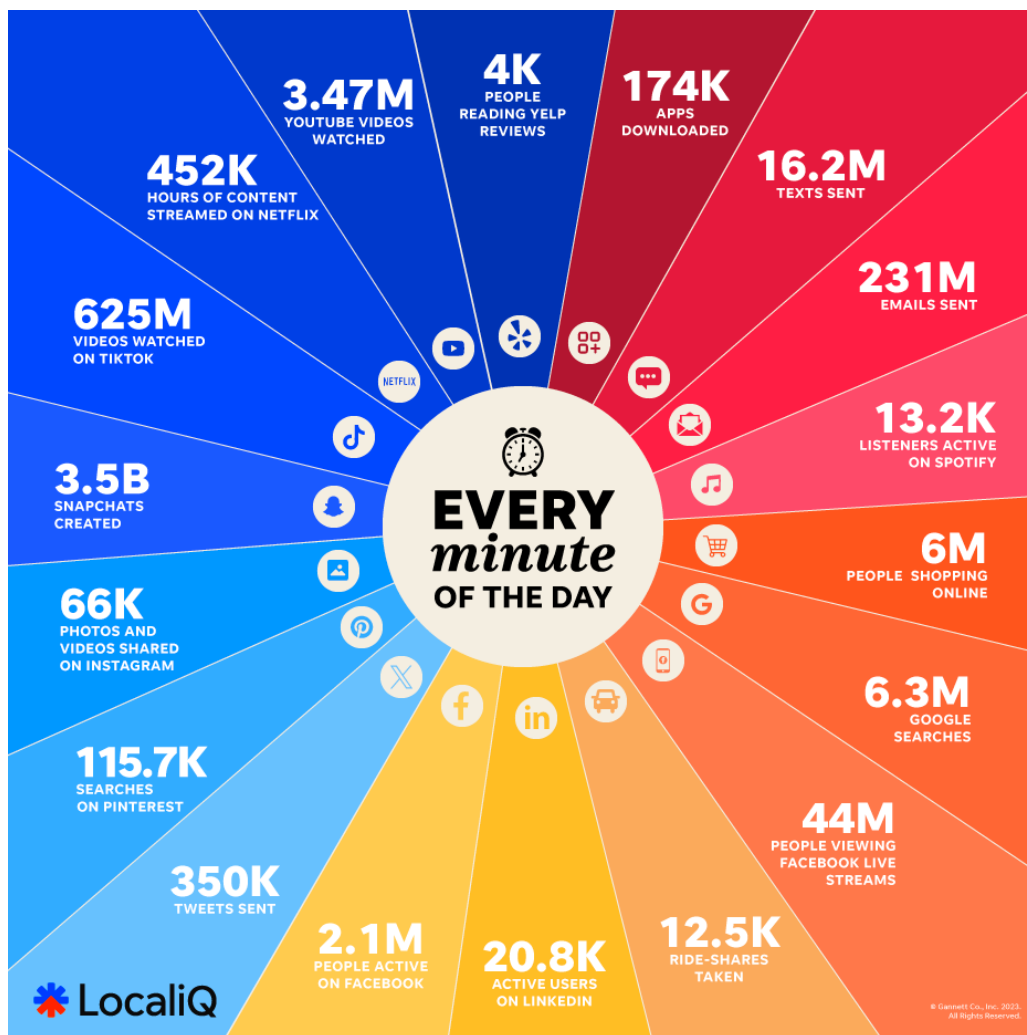
Hình 1.1: Các giai đoạn phát triển của Big Data (Mohammad Shahid Husain et al., 2023)

Giai đoạn	Thời gian	Đặc điểm
Giai đoạn 1	1970 – 2000	Trong giai đoạn này, dữ liệu được lưu trữ, xử lý và khai thác sử dụng hệ quản trị cơ sở dữ liệu như hệ quản trị cơ sở dữ liệu quan hệ (RDBMS). Các doanh nghiệp cũng đã bắt đầu sử dụng data warehouse để lưu trữ những dữ liệu từ quá khứ. Phần lớn dữ liệu được lưu trữ, xử lý và phân tích trong giai đoạn này là dữ liệu có cấu trúc – dữ liệu có định dạng chuẩn hoá, có thể khai thác được bằng các câu lệnh truy vấn như SQL.

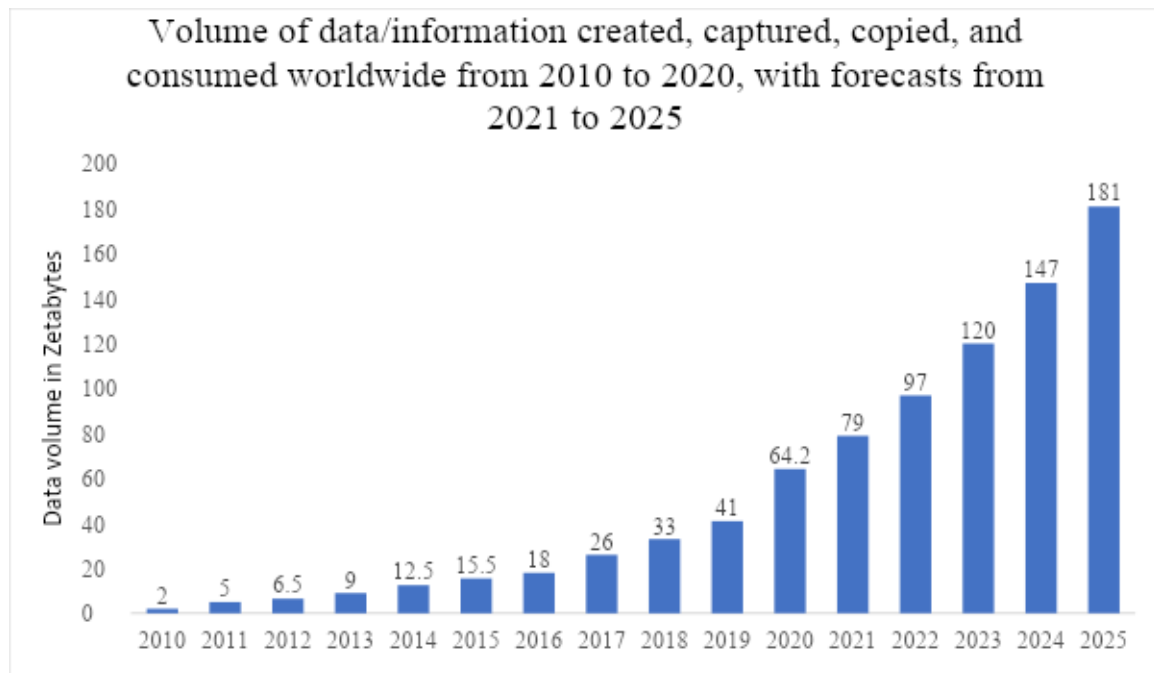
Giai đoạn 2	2000 – 2010	Với sự ra đời của Internet và World Wide Web, dữ liệu không còn bị giới hạn dưới dạng bảng mà được mở rộng ra với rất nhiều hình thức lưu trữ khác nhau: địa chỉ IP, lịch sử duyệt Web, tỷ lệ nhấp chuột, nhật ký tìm kiếm,... Lượng người truy cập web sẽ tỉ lệ thuận với sự gia tăng lưu trữ các loại dữ liệu bán cấu trúc và không cấu trúc kể trên. Vậy nên, để hiểu hơn về khách hàng của mình trong bối cảnh thời đại mới, các doanh nghiệp đã phải tạo ra những cách thức tinh vi hơn để phân tích và trích xuất thông tin hữu ích từ các nguồn dữ liệu phi cấu trúc ngày càng gia tăng.
Giai đoạn 3	2010 đến hiện tại	Bước sang giai đoạn này, tuy ưu tiên hàng đầu của các doanh nghiệp vẫn là tập trung vào xử lý và phân tích dữ liệu phi cấu trúc trên web nhưng sự ra đời của các thiết bị di động đã tạo ra các xu hướng khai thác dữ liệu mới. Ví dụ, các thiết bị di động cho phép lưu trữ và phân tích dữ liệu không gian-thời gian cũng như dữ liệu hành vi của người dùng dựa trên các mô hình điều hướng web của họ. Các xu hướng mới nhất dựa trên công nghệ cảm biến như Internet vạn vật (IoT), Internet vạn vật công nghiệp (IIoT), Internet y tế vạn vật (IoMT), mạng cảm biến cơ thể (BSN), v.v., đang tạo ra khối lượng dữ liệu khổng lồ với tốc độ chưa từng thấy trước đây

Bảng 1.1: Chú giải về các giai đoạn phát triển của Big Data (Mohammad Shahid Husain et al., 2023)

Có thể thấy qua lịch sử phát triển của Big Data, các kiểu dữ liệu mỗi ngày một thêm mới và ngày càng đa dạng. Song song với đó, khối lượng dữ liệu tạo ra sau mỗi năm cũng tăng nhanh, là hệ quả của số lượng dữ liệu đa dạng được tạo ra mỗi ngày. Ước tính đến năm 2025, số lượng dữ liệu sẽ đạt ngưỡng 181 zetabytes.



Hình 1.2: Thống kê dữ liệu được tạo ra trong từng phút của năm 2023 (Heitman, 2022)



Hình 1.3: Thống kê dữ liệu được tạo ra từ năm 2010-2020 và dự đoán cho giai đoạn 2021-2025 (Statista, 2023)

1.1.2. Tầm quan trọng của việc phân tích và xử lý Big Data tại các doanh nghiệp

Với khối lượng dữ liệu khổng lồ được tạo ra mỗi ngày thông qua sự tương tác của người tiêu dùng, doanh nghiệp sẽ thực sự làm chủ cuộc chơi nếu có thể khai thác hết tiềm lực từ lượng thông tin màu mỡ này. Xử lý và khai thác dữ liệu lớn hiệu quả có thể giúp doanh nghiệp:

- **Đưa ra các quyết định kinh doanh chiến lược:** Việc thu thập dữ liệu từ nhiều nguồn khác nhau, với đa dạng chủng loại có thể hỗ trợ doanh nghiệp trong việc đưa ra các quyết định thực tiễn dựa trên số liệu và bằng chứng cụ thể mà không phải chỉ dựa vào phán đoán như trước đây. Ví dụ hệ thống siêu thị Walmart của Mỹ xây dựng một cổng dữ liệu Data Café để phân tích chi tiết hàng trăm luồng dữ liệu nội bộ và bên ngoài từ đó tìm ra được các nguyên nhân đằng sau sự sụt giảm chỉ số bán hàng mà hệ thống siêu thị đang gặp phải.
- **Hiểu rõ nhu cầu khách hàng và cung cấp những sản phẩm, dịch vụ phù hợp:** Dữ liệu lớn thu thập chi tiết thông tin người dùng từ đó cung cấp những sản phẩm và dịch vụ dựa theo sở thích và nhu cầu. Ví dụ Disney đã sáng chế ra vòng đeo

tay thanh toán Magic Band hỗ trợ khách hàng sử dụng tất cả các dịch vụ tại Disney chỉ dùng 1 lần quét: từ khâu vào cổng, trải nghiệm khu vui chơi, đặt phòng khách sạn. Từ đó, Disney thu thập được một lượng lớn dữ liệu về hành vi của người tiêu dùng để vận hành khu nghỉ dưỡng hiệu quả hơn (ví dụ: dễ dàng phát hiện ra tình trạng quá tải) và cung cấp trải nghiệm được cá nhân hóa và chu đáo hơn cho du khách (các nhân vật Disney tự động chào đón du khách bằng tên).

- **Cải thiện hiệu quả hoạt động của doanh nghiệp:** Dữ liệu lớn góp phần tự động hoá nhiều quy trình mang tính chất lặp lại, thủ công của doanh nghiệp từ đó gia tăng năng suất hoạt động. Ví dụ, các chatbot (công cụ trò chuyện tự động) đã được sử dụng để trả lời các câu hỏi thường gặp của khách hàng, nhân viên. công ty phần mềm nhân sự PeopleDoc đã ra mắt một nền tảng Tự động hóa Quy trình bằng Robot (RPA), hoạt động song song với các hệ thống hiện có của công ty và tiếp nhận các quy trình hoặc sự kiện có thể được tự động hóa. Bằng cách này, các tác vụ mang tính chất lặp đi lặp lại có thể được thực hiện bởi máy móc hoặc thuật toán, tiết kiệm thời gian và nguồn lực của nhân viên cho những công việc đòi hỏi chuyên môn và tính sáng tạo.

1.1.3. Lý do chọn đề tài

Nhận thức được Big Data chính là một khía cạnh không thể không nhắc đến khi muốn khai thác đầy đủ thông tin trong bối cảnh bùng nổ dữ liệu trong thời đại ngày nay, cũng như những lợi ích to lớn mà Big Data có thể đem lại cho hoạt động kinh doanh của doanh nghiệp, nhóm đã quyết định chọn hai ông lớn trong lĩnh vực giải trí với số lượng người dùng toàn cầu tính đến năm 2024 là hơn 500 triệu người – Spotify và Facebook – để tìm hiểu về những công nghệ Big Data mà các ông lớn này đã áp dụng trong đế chế tỷ đô của mình.

Cụ thể nhóm sẽ tiến hành tìm hiểu:

- Công nghệ ở Spotify: Hadoop, Hive, Spark, Kafka, Cassandra, HDFS, Storm, Crunch, Scalding.

- Các công nghệ và nền tảng được sử dụng tại Facebook: Scribe, Puma, Stylus, Swift, Laser, Scuba, Hive, Hadoop, HBase.

Và sau đó nhóm sẽ ứng dụng những kiến thức đã học về Big Data để xây dựng hệ thống hỏi đáp sử dụng mô hình BERT và FAISS trên bộ dữ liệu SQuADv2.

1.2. Mục tiêu của đề tài

Nghiên cứu công nghệ Big Data được ứng dụng tại hai doanh nghiệp dẫn đầu trong ngành công nghiệp giải trí là Facebook và Spotify.

Áp dụng các kiến thức Big Data đã học để xây dựng hệ thống hỏi đáp sử dụng mô hình BERT và FAISS trên bộ dữ liệu SQuADv2.

1.3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: Doanh nghiệp Facebook và Spotify, bộ dữ liệu SQuADv2.

Phạm vi nghiên cứu: Công nghệ Big Data sử dụng tại Facebook và Spotify; ứng dụng của mô hình BERT và FAISS trong việc xây dựng hệ thống hỏi đáp

1.4. Phương pháp thực hiện

Nghiên cứu và tổng hợp các tài liệu, bài báo viết về cách các doanh nghiệp lớn như Facebook, Spotify áp dụng công nghệ Big Data trong hoạt động kinh doanh của mình.

Nghiên cứu lý thuyết về mô hình BERT, FAISS từ đó mở rộng lý thuyết để ứng dụng xây dựng hệ thống hỏi đáp.

1.5. Bố cục của đề tài

Bài làm của nhóm được tổ chức thành 7 chương:

Chương 1: Trình bày tổng quan về sự phát triển của Big Data cũng như những lợi ích mà Big Data có thể đem lại cho doanh nghiệp. Từ đó xác định hướng đi của đề tài nghiên cứu cũng như đặt ra những mục tiêu cần đạt được.

Chương 2: Giới thiệu về Facebook và công nghệ Big Data ứng dụng tại doanh nghiệp này.

Chương 3: Giới thiệu về Spotify và công nghệ Big Data ứng dụng tại doanh nghiệp này.

Chương 4: Cơ sở lý thuyết để xây dựng hệ thống hỏi đáp. Tìm hiểu về Tokenizer; cơ chế Attention Mechanism và Transformer; mô hình BERT; mô hình XLM-RoBERTa; mô hình RoBERTa; hệ thống QA System; và độ đo Exact Match và F1.

Chương 5: Giới thiệu về bộ dữ liệu SQuADv2 và phân tích mô tả bộ dữ liệu này.

Chương 6: Tiến hành xây dựng mô hình và xây dựng giao diện cho hệ thống hỏi đáp

Chương 7: Tổng kết về công nghệ Big Data ứng dụng tại Spotify, Facebook cũng như những kết quả đạt được qua việc xây dựng hệ thống hỏi đáp. Từ đó đề ra hướng phát triển cho các doanh nghiệp và hệ thống hỏi đáp đã xây dựng.

1.6. Phân công công việc

STT	Họ và tên	MSSV	Công việc	Đánh giá
1	Phạm Minh Hiền	31211021654	Chương 1: Tổng quan Chương 4: Cơ sở lý thuyết (4.6 và 4.7) Kiểm duyệt thông tin, trình bày word, check đạo văn Làm slide	100%
2	Phan Thị Thanh Hoa	31211027176	Chương 4: Cơ sở lý thuyết (4.1 đến 4.5) Làm slide	100%
3	Nguyễn Phú Hưng	31211023521	Chương 5: Mô tả bộ dữ liệu Làm slide	100%
4	Trương Thị Hồng Mai	31211026726	Chương 6: Xây dựng mô hình & giao diện Làm slide	100%
5	Đào Thị Phương Quỳnh	31211027667	Chương 3: Ứng dụng dữ liệu lớn ở Spotify Demo MapReduce	100%

STT	Họ và tên	MSSV	Công việc	Đánh giá
			Làm slide	
6	Vũ Nguyễn Thảo Vi	31211027686	Chương 2: Ứng dụng dữ liệu lớn ở Facebook Demo HBase Làm slide	100%
7	Phan Dương Hoàng Vũ	31211022533	Xây dựng hệ thống hỏi đáp sử dụng mô hình BERT và FAISS Chương 7: Kết luận Làm slide	100%

Bảng phân công công việc cho các thành viên trong nhóm

CHƯƠNG 2: ỨNG DỤNG DỮ LIỆU LỚN Ở FACEBOOK

2.1. Giới thiệu về Facebook

Facebook được thành lập vào năm 2004 bởi Mark Zuckerberg và các bạn cùng phòng tại Harvard, đã phát triển thành một trong những mạng xã hội lớn nhất và phổ biến nhất trên toàn cầu. Với hơn 3 tỷ người dùng hoạt động hàng tháng tính đến năm 2024 (Meta, 2024), Facebook cung cấp nền tảng cho người dùng kết nối, chia sẻ thông tin, hình ảnh, video, và nhiều nội dung khác. Sự phát triển không ngừng của Facebook đã tạo ra một hệ sinh thái phong phú gồm nhiều dịch vụ và ứng dụng, như Messenger, Instagram và WhatsApp, tất cả đều thuộc sở hữu của Meta Platforms Inc, trước đây là Facebook Inc.

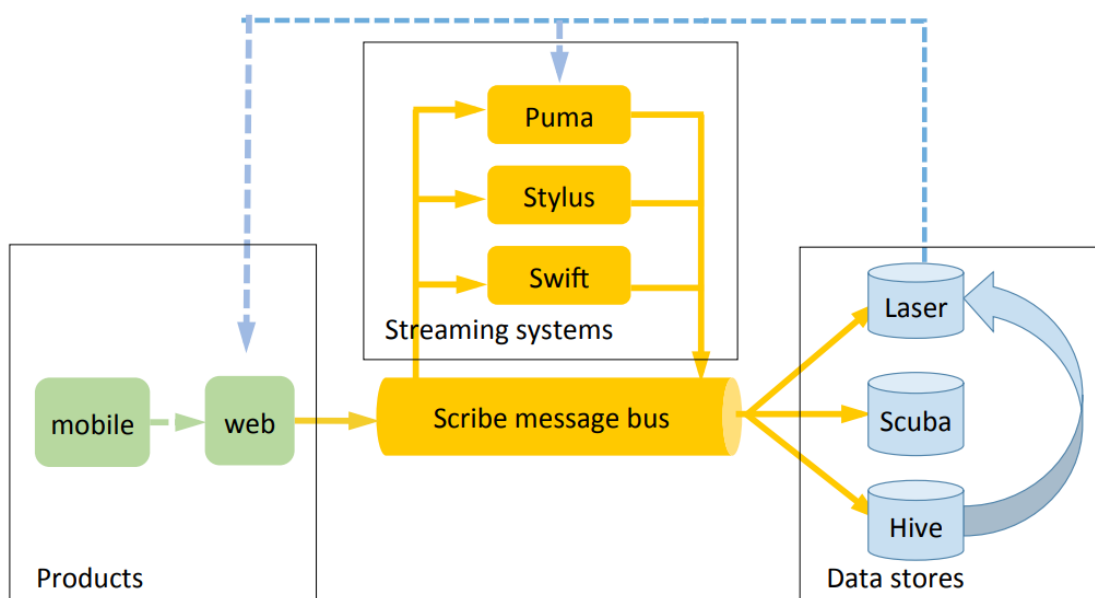
2.2. Dữ liệu lớn từ Facebook

Theo Facebook 2020 statistics, Facebook tạo ra 4 petabyte dữ liệu mỗi ngày. Dữ liệu được tạo ra từ bài viết, hình ảnh, video, tin nhắn, nhật ký hoạt động, hành vi duyệt web của người dùng, bao gồm các trang web mà họ truy cập, các liên kết họ click vào, và thời gian họ dành trên mỗi trang. Vì thế, việc thu thập, lưu trữ và phân tích dữ liệu trở nên cực kỳ phức tạp. Những khó khăn mà Facebook gặp phải trong quá trình quản lý và xử lý lượng dữ liệu lớn là:

- Khó khăn trong việc lưu trữ lượng dữ liệu khổng lồ.
- Dữ liệu trên Facebook không chỉ lớn mà còn thay đổi liên tục theo thời gian. Việc theo dõi và phản ứng nhanh chóng đối với các sự kiện mới là một thách thức lớn, đặc biệt khi cần phải xử lý dữ liệu thời gian thực.
- Việc truy vấn và phân tích dữ liệu phải được thực hiện một cách nhanh chóng.
- Dữ liệu trên Facebook đến từ nhiều nguồn và có nhiều định dạng khác nhau, từ văn bản đến hình ảnh, video và thông tin địa lý. Việc quản lý và phân tích dữ liệu đa dạng này đòi hỏi các công cụ và kỹ thuật phức tạp.

Đối với một công ty truyền thông xã hội như Facebook, việc phân tích dữ liệu theo thời gian thực là vô cùng quan trọng, giúp Facebook tổng hợp xu hướng, phân tích ứng dụng

di động, cung cấp thông tin thời gian thực cho các nền tảng khác. Hình 1 là một hệ sinh thái xử lý dữ liệu thời gian thực của Facebook.



Hình 2.1: Hệ thống xử lý dữ liệu realtime của Facebook. Nguồn: Chen et al. (2016)

Hệ thống thu thập và lưu trữ dữ liệu

Dữ liệu được thu thập từ các sản phẩm của Facebook như web, ứng dụng di động. Dữ liệu này được đẩy vào một hệ thống lưu trữ trung tâm gọi là Scribe. Trong Scribe, dữ liệu được tổ chức thành các "category" - những dòng dữ liệu riêng biệt. Mỗi category lại được chia nhỏ ra thành nhiều "bucket" - những ngăn chứa nhỏ hơn. Điều này giúp các hệ thống khác nhau có thể xử lý từng phần của dữ liệu một cách song song, nâng cao hiệu suất. Scribe cũng cho phép các hệ thống đọc lại dữ liệu đã lưu trong vài ngày gần nhất, hữu ích cho việc khắc phục sự cố hoặc xử lý lại dữ liệu. Scribe lưu trữ dữ liệu vào HDFS, cho phép các hệ thống đọc lại dữ liệu trong vài ngày gần nhất.

Hệ thống trích xuất và xử lý dữ liệu

Sau khi dữ liệu được lưu trữ trong Scribe, các hệ thống xử lý luồng sẽ đọc dữ liệu từ các category/bucket của Scribe và thực hiện các phép xử lý khác nhau. Facebook có ba hệ thống xử lý luồng chính:

- Puma: là hệ thống xử lý dòng dữ liệu với các ứng dụng viết bằng ngôn ngữ tương tự SQL và UDFs viết bằng Java. Puma giúp tạo ra các kết quả truy vấn đã được tính toán trước cho các truy vấn tổng hợp đơn giản và cung cấp tính năng lọc và xử lý dòng dữ liệu Scribe.
- Swift: Cung cấp API đơn giản để đọc và xử lý luồng dữ liệu từ Scribe, hữu ích cho xử lý lưu lượng thấp.
- Stylus: Stylus là framework xử lý dòng dữ liệu cấp thấp viết bằng C++. Stylus có thể xử lý dữ liệu có trạng thái hoặc không có trạng thái và có thể kết hợp các bộ xử lý thành một DAG phức tạp.

Các ứng dụng Puma, Swift, Stylus có thể kết nối với nhau thành một chuỗi dài các bước xử lý phức tạp hình thành nên một mạng lưới phức tạp - DAG (Directed Acyclic Graph).

Cuối cùng, kết quả của quá trình xử lý sẽ được lưu trữ vào các hệ thống khác nhau như Laser, Scuba, Hive để phục vụ các mục đích phân tích và truy vấn khác nhau.

Ngoài ra, Facebook còn áp dụng nhiều công nghệ Big Data khác như Apache Hadoop, Apache Kafka, Apache HBase.

2.3. Các công nghệ và nền tảng xử lý dữ liệu lớn được sử dụng tại Facebook

2.3.1. Scribe

Scribe là một hệ thống tin nhắn phân tán (distributed messaging system) được phát triển bởi Facebook, với mục đích thu thập, tổng hợp và truyền tải khối lượng lớn dữ liệu log với độ trễ vài giây và throughput cao. Được thiết kế để xử lý hàng petabyte log mỗi giờ, Scribe đóng vai trò như cơ chế vận chuyển chính cho cả hệ thống xử lý batch và thời gian thực tại Facebook. Dữ liệu trong Scribe được tổ chức theo các danh mục (categories) và các danh mục này lại được chia thành nhiều bucket để hỗ trợ xử lý song song.

Một số đặc điểm nổi bật của Scribe:

- Độ trễ thấp và throughput cao: Scribe xử lý log với tốc độ nhập dữ liệu có thể vượt quá 2.5 terabyte mỗi giây và xuất dữ liệu có thể vượt quá 7 terabyte mỗi giây.
- Phân loại và tổ chức dữ liệu: Dữ liệu log được tổ chức theo các danh mục và bucket, giúp dễ dàng quản lý và xử lý song song.
- Tính bền vững của dữ liệu: Dữ liệu được lưu trữ bền vững trong HDFS, cho phép các hệ thống tiếp nhận có thể đọc và xử lý lại log trong vài ngày.

Các thành phần trong kiến trúc của Scribe:

- Producer: Thành phần ghi log vào Scribe. Producer có thể là các ứng dụng chạy trong container hoặc các web server.
- Scribed: Daemon cục bộ trên mỗi máy chủ, chịu trách nhiệm gửi log tới backend lưu trữ.
- Write Service: Tầng backend của Scribe, xử lý việc lưu trữ log trực tiếp từ Producer.
- Read Service: Dịch vụ cho phép đọc log theo luồng.

Cách hoạt động của Scribe:

Thu thập log

- Các dịch vụ và ứng dụng tại Facebook ghi log vào Scribe thông qua thư viện Producer.
- Log có thể được gửi đến một daemon cục bộ gọi là Scribed hoặc trực tiếp đến tầng backend của Scribe (Write Service).

Tổ chức và lưu trữ dữ liệu

- Dữ liệu log được tổ chức theo các danh mục (categories), mỗi danh mục đại diện cho một luồng dữ liệu riêng biệt.
- Mỗi danh mục lại chia thành nhiều bucket, đơn vị cơ bản để xử lý luồng dữ liệu. Các bucket này được phân phối tới các tiến trình xử lý khác nhau để tăng cường tính song song.
- Log được lưu trữ bền vững trong HDFS, đảm bảo tính bền vững và khả năng truy xuất lại dữ liệu sau này.

Xử lý và đọc dữ liệu

- Scribe hỗ trợ việc đọc log theo luồng thông qua dịch vụ Read Service, cho phép các ứng dụng phân tích truy cập dữ liệu log theo thời gian thực.
- Dữ liệu log có thể được xử lý theo nhiều cách khác nhau, ví dụ như gửi tới các hệ thống phân tích thời gian thực như Puma, Stylus, và Scuba, hoặc lưu trữ trong kho dữ liệu để phân tích lịch sử.

Quản lý dữ liệu

- Lưu trữ và quản lý log: Log được lưu trữ trong HDFS, cho phép giữ lại dữ liệu trong vài ngày để phục vụ các yêu cầu truy xuất và phân tích sau này.
- Khả năng tái phát (Replay) cho phép dữ liệu log có thể được phát lại bởi cùng hoặc các hệ thống tiếp nhận khác nhau trong thời gian lưu trữ.

2.3.2. Puma

Puma là một hệ thống xử lý luồng (stream processing system) được thiết kế để xử lý và phân tích dữ liệu liên tục theo thời gian thực. Ứng dụng trong Puma được viết bằng một ngôn ngữ tương tự như SQL. Ngoài ra, các hàm do người dùng định nghĩa (UDFs) trong Puma có thể được viết bằng ngôn ngữ lập trình Java. UDFs giúp mở rộng khả năng của Puma, cho phép thực hiện các tác vụ phức tạp hơn mà ngôn ngữ SQL không hỗ trợ. Việc viết, kiểm tra và triển khai một ứng dụng mới trong Puma có thể hoàn thành trong chưa đầy một giờ. Điều này giúp tăng hiệu quả và tốc độ phát triển, đặc biệt là khi cần phân tích dữ liệu nhanh chóng. Puma được sử dụng vào hai mục đích chính:

- Cung cấp kết quả truy vấn được tính toán trước
- Puma cũng cung cấp khả năng lọc và xử lý các luồng dữ liệu Scribe với độ trễ chỉ vài giây. Puma có thể giảm một luồng chứa tất cả các hành động trên Facebook chỉ còn các bài đăng, hoặc chỉ các bài đăng có chứa hashtag cần tìm. Kết quả đầu ra của các ứng dụng này là một luồng Scribe khác, có thể được sử dụng làm đầu vào cho một ứng dụng Puma khác, bất kỳ bộ xử lý luồng thời gian thực nào khác. Các ứng dụng tổng hợp của Puma lưu trữ trạng thái trong một cụm HBase chung.

2.3.3. Stylus

Stylus là một hệ thống xử lý luồng dữ liệu thời gian thực được viết bằng ngôn ngữ C++. Thành phần cơ bản của Stylus là bộ xử lý luồng dữ liệu (stream processor) nhận đầu vào là một luồng Scribe và đầu ra có thể là một luồng Scribe khác hoặc một kho dữ liệu. Bộ xử lý trong Stylus có thể là không trạng thái (stateless) hoặc có trạng thái (stateful), và các bộ xử lý này có thể kết hợp thành một đồ thị xử lý có hướng phức tạp (DAG). Giao diện lập trình ứng dụng (API) của Stylus tương tự như các hệ thống xử lý luồng theo thủ tục khác, cho phép viết ứng dụng xử lý luồng một cách dễ dàng. Stylus xử lý các luồng đầu vào có thứ tự không hoàn hảo bằng cách yêu cầu người viết ứng dụng xác định thời gian xảy ra sự kiện, và cung cấp chức năng ước tính "low watermark" của thời gian sự kiện với một khoảng tin cậy nhất định. Stylus là một hệ thống mạnh mẽ và linh hoạt, đáp ứng nhu cầu của nhiều ứng dụng phân tích dữ liệu hiện đại nhờ khả năng xử lý dữ liệu có trạng thái và không trạng thái, hỗ trợ tạo đồ thị xử lý phức tạp, API thân thiện, và các công cụ xử lý thứ tự không hoàn hảo.

2.3.4. Swift

Swift là một công cụ xử lý luồng dữ liệu được phát triển bởi Facebook, cung cấp các chức năng cơ bản cho việc xử lý dữ liệu ngay lập tức, đặc biệt là được tùy chỉnh để phù hợp với nhu cầu của Scribe, một framework dùng để truyền dữ liệu liên tục. Tính chất cốt lõi của Swift là cung cấp một cơ chế checkpointing mạnh mẽ, cho phép khôi phục dữ liệu một cách trơn tru và duy trì tính liên tục trong quá trình xử lý dữ liệu ngay cả khi ứng dụng gặp sự cố. Kiến trúc cơ bản của Swift được đặc trưng bởi một API đơn giản, giúp dễ dàng tích hợp và sử dụng.

Điểm then chốt trong thiết kế của Swift là sự phụ thuộc vào ống hệ thống (system-level pipes) để giao tiếp với các ứng dụng khách. Lựa chọn thiết kế này không chỉ nâng cao hiệu suất mà còn tăng khả năng chịu lỗi.

2.3.5. Laser

Laser là một dịch vụ lưu trữ key-value có khả năng xử lý lượng truy vấn cao, độ trễ thấp (trong hàng mili giây), được xây dựng trên nền tảng của RocksDB. Laser có khả năng

đọc dữ liệu từ bất kỳ danh mục Scribe nào trong thời gian thực hoặc từ bất kỳ bảng Hive nào một lần trong một ngày. Cặp key và value trong Laser có thể là bất kỳ kết hợp nào của các cột trong luồng đầu vào (đã được serialize). Dữ liệu được lưu trữ trong Laser sau đó có thể truy cập được bởi mã sản phẩm của Facebook và các ứng dụng Puma và Stylus.

Laser có hai trường hợp sử dụng phổ biến. Thứ nhất, Laser có thể làm cho luồng dữ liệu đầu ra của một ứng dụng Puma hoặc Stylus có sẵn cho các sản phẩm của Facebook. Thứ hai, Laser cũng có thể làm cho kết quả của một truy vấn Hive phức tạp hoặc một luồng Scribe có sẵn cho một ứng dụng Puma hoặc Stylus, thường được sử dụng cho việc kết hợp tra cứu, chẳng hạn như xác định chủ đề cho một hashtag cụ thể.

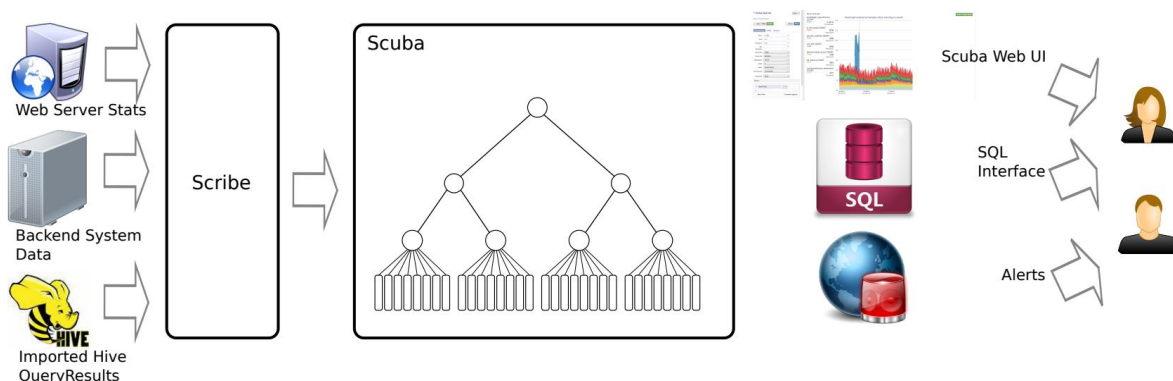
2.3.6. Scuba

Scuba là một hệ thống cơ sở dữ liệu trên bộ nhớ (in-memory database system) được phát triển bởi Facebook. Đây là một công nghệ được thiết kế để thu thập, lưu trữ và truy vấn dữ liệu thời gian thực với độ trễ rất thấp. Scuba hỗ trợ phân tích và giám sát hiệu suất của hệ thống với tốc độ cao và tính linh hoạt cao, giúp Facebook nhanh chóng chẩn đoán và xử lý các vấn đề hiệu suất cũng như đánh giá tác động của các thay đổi hạ tầng.

Đặc điểm của Scuba là dữ liệu được lưu trữ trực tiếp trong RAM, thay vì trên các thiết bị lưu trữ như HDD hoặc SSD. Việc lưu trữ dữ liệu trong RAM cho phép truy cập và xử lý dữ liệu với tốc độ nhanh hơn nhiều so với lưu trữ trên đĩa cứng, vì RAM có tốc độ truy cập dữ liệu cao hơn đáng kể. Vì vậy, độ trễ giữa sự kiện xảy ra và dữ liệu xuất hiện trong các truy vấn thường dưới một phút. Scuba cho phép chạy trên hàng trăm máy chủ với tổng dung lượng RAM lớn, hỗ trợ hàng triệu hàng dữ liệu mỗi giây. Scuba còn cung cấp giao diện truy vấn SQL và GUI, hỗ trợ các phân tích và truy vấn dữ liệu thời gian thực. Ngoài ra, các bảng dữ liệu được phân tán ngẫu nhiên trên tất cả các máy chủ trong cụm, giúp tăng tính sẵn sàng và khả năng chịu lỗi.

Cách hoạt động của Scuba:

Scuba bao gồm nhiều máy chủ kết nối với nhau, mỗi máy chủ được chia thành các đơn vị lưu trữ logic gọi là "Leaf Nodes" (lá). Mỗi máy chủ chứa 8 lá, tương ứng với số lõi CPU. Dữ liệu trong các bảng được phân chia thành từng phần, lưu trên các lá khác nhau và mỗi truy vấn sẽ truy cập đến tất cả các lá này.



Hình 2.2: Kiến trúc hệ thống Scuba. Nguồn: (Abraham et al., 2013)
Quá trình thu nhập dữ liệu (Data Ingestion):

- Dữ liệu từ các nguồn khác nhau được đưa vào Scuba thông qua hệ thống Scribe (phần mềm nguồn mở thu thập log).
- Tiến trình tailer sẽ đọc dữ liệu từ Scribe rồi gửi tới Scuba qua Thrift API.

Lưu trữ dữ liệu (Data Storage):

- Dữ liệu được phân phối một cách ngẫu nhiên và lưu trữ hoàn toàn trong bộ nhớ trên các lá.
- Khi dữ liệu mới được gửi vào Scuba, chúng được nhóm thành từng "batch" nhỏ. Mỗi batch chứa các hàng dữ liệu mới, và tất cả các hàng trong cùng batch đều có timestamp trong cùng một khoảng thời gian ngắn. Ví dụ, tất cả hàng trong batch 1 có thời gian từ 10:00-10:05, batch 2 từ 10:05-10:10, batch 3 từ 10:10-10:15, v.v.
- Mỗi bảng dữ liệu sẽ có các phần dữ liệu được phân tán trên nhiều lá khác nhau. Vì vậy, khi cần truy xuất hay phân tích dữ liệu, Scuba sẽ phải truy cập song song lên tất cả các lá có chứa phần dữ liệu của bảng đó.

Xử lý truy vấn (Query Processing):

- Giao diện web, dòng lệnh và Thrift API cho phép người dùng tạo truy vấn.
- Truy vấn được gửi tới Root Aggregator, sau đó lan truyền qua các Intermediate Aggregator. Các Intermediate Aggregator tiếp tục chuyển truy vấn xuống các Leaf Aggregator gần nhất với dữ liệu cần tìm. Mỗi máy chủ (server) của Scuba sẽ có một Leaf Aggregator riêng.
- Leaf Aggregator gửi truy vấn song song tới các Leaf Server trên cùng máy để xử lý. Các Leaf Server này chính là nơi dữ liệu được lưu trữ.
- Các Leaf Server thực hiện tìm kiếm và trả về kết quả cho Leaf Aggregator của máy chủ đó. Leaf Aggregator sẽ tổng hợp kết quả và chuyển ngược lên các Intermediate Aggregator, rồi đến Root Aggregator.
- Cuối cùng, Root Aggregator sẽ tổng hợp tất cả kết quả, tính toán thêm nếu cần, rồi trả lại kết quả cuối cùng cho người dùng thông qua giao diện web, dòng lệnh hoặc ứng dụng.

Quản lý dữ liệu (Data Management):

- Dữ liệu cũ có thể được xóa sau một khoảng thời gian nhất định để giải phóng dung lượng lưu trữ và tăng hiệu suất hệ thống.
- Thay vì xóa hoàn toàn dữ liệu cũ, một phần mẫu dữ liệu có thể được lưu trữ để phục vụ mục đích phân tích, nghiên cứu sau này.

Tóm lại, Scuba có kiến trúc phân tán với các thành phần chính gồm: Ingestion, Storage, Query Processing và Data Management layer. Hệ thống được thiết kế để xử lý dữ liệu lớn với tốc độ cao bằng cách phân tán xử lý trên nhiều máy chủ.

2.3.7. Hive

Hive là một hệ thống kho dữ liệu được sử dụng để truy vấn và phân tích các tập dữ liệu lớn được lưu trữ trong HDFS. Hive sử dụng ngôn ngữ truy vấn có tên HiveQL, tương

tự như SQL. Nó được xây dựng dựa trên các công nghệ nguồn mở như Apache Hadoop, Apache Hive và Presto.

Các thành phần trong kiến trúc của Hive:

- Metastore: Lưu trữ metadata về bảng, phân vùng và các cột trong các bảng.
- Query compiler và execution engine: Chuyển đổi các truy vấn SQL thành các job MapReduce để thực hiện trên Hadoop.
- SerDe và ObjectInspectors: Cung cấp các giao diện lập trình để xử lý các định dạng dữ liệu và kiểu dữ liệu phổ biến.
- UDF và UDAF: Cung cấp các giao diện lập trình cho các hàm do người dùng định nghĩa (hàm scalar và hàm tổng hợp).
- Clients: Cung cấp giao diện dòng lệnh tương tự như MySQL và một giao diện web.

Cách hoạt động của Hive:

Người dùng gửi truy vấn từ giao diện người dùng (UI) đến Driver. Driver sẽ nhận truy vấn và gửi yêu cầu tới Compiler để lập kế hoạch thực thi. Kế hoạch này bao gồm các bước chi tiết về cách thực thi truy vấn. Sau đó, Compiler gửi yêu cầu tới Metastore để lấy metadata. Metadata chứa thông tin về cấu trúc của bảng, các cột, phân vùng và các thông tin khác liên quan đến dữ liệu. Metastore sẽ gửi metadata trở lại cho Compiler và Compiler sử dụng metadata để biên dịch truy vấn thành kế hoạch thực thi (execution plan) và gửi kế hoạch này trở lại cho Driver. Driver gửi kế hoạch thực thi tới Execution Engine. Execution Engine là thành phần chịu trách nhiệm thực thi kế hoạch này. Trong quá trình thực thi, Execution Engine có thể cần thực hiện các thao tác liên quan đến metadata trong Metastore, chẳng hạn như đọc hoặc ghi thông tin metadata. Sau khi hoàn thành việc thực thi, Execution Engine gửi kết quả truy vấn trở lại cho Driver. Cuối cùng, Driver gửi kết quả cuối cùng của truy vấn về giao diện người dùng để hiển thị cho người dùng.

2.3.8. Apache Hadoop

Hadoop là một nền tảng phần mềm nguồn mở được sử dụng để lưu trữ và xử lý các tập dữ liệu lớn có kích thước từ gigabyte đến petabyte. Thay vì sử dụng một máy tính lớn để lưu trữ và xử lý dữ liệu, Hadoop cho phép kết hợp nhiều máy tính thành một cụm để phân tích các tập dữ liệu khổng lồ song song nhanh hơn.

Các thành phần chính của Hadoop:

HDFS (Hadoop Distributed File System)

HDFS là hệ thống file phân tán được sử dụng để lưu trữ dữ liệu. Nó phân chia dữ liệu thành các khối và phân tán chúng trên nhiều máy tính.

MapReduce

MapReduce là khung mô hình lập trình để xử lý và tính toán trên số lượng dữ liệu lớn trong môi trường phân tán. Nó hoạt động dựa trên 2 pha chính là Map và Reduce. Pha Map sẽ chia nhỏ dữ liệu thành các cặp key-value, sau đó pha Reduce sẽ nhóm các cặp key-value lại với nhau để tính toán kết quả cuối cùng.

YARN (Yet Another Resource Negotiator)

YARN là một khung phần mềm quản lý và phân bổ tài nguyên để vận hành các ứng dụng trên Hadoop. Nó cung cấp các dịch vụ như quản lý tài nguyên, lên lịch và giám sát cho các ứng dụng chạy trên Hadoop. YARN phân bổ RAM, bộ nhớ và các tài nguyên khác cho các ứng dụng khác nhau.

Cách hoạt động của Hadoop:

Hadoop hoạt động theo mô hình master-worker phân tán.

Lưu trữ dữ liệu với HDFS (Hadoop Distributed File System):

- Dữ liệu được chia thành các khối (block) và lưu trữ trên nhiều máy chủ (DataNode) trong cụm (cluster).

- Có một máy chủ quản lý trung tâm (NameNode) quản lý vị trí của các khối dữ liệu.
- Dữ liệu được sao chép trên nhiều máy chủ để đảm bảo sự dự phòng khi có lỗi xảy ra.

Xử lý dữ liệu với MapReduce:

- JobTracker (máy chủ) phân chia công việc thành nhiều công đoạn Map và Reduce nhỏ hơn.
- TaskTracker (nhân viên) trên mỗi máy chủ nhận và thực hiện các công đoạn Map/Reduce được giao.
- Giai đoạn Map: Dữ liệu đầu vào được lập thành các cặp khóa-giá trị và xử lý song song trên nhiều máy chủ.
- Giai đoạn Shuffle and Sort: Kết quả của Map được trộn và sắp xếp lại.
- Giai đoạn Reduce: Các giá trị được gộp lại theo khóa và xử lý tiếp để tạo ra kết quả cuối cùng.

Điều phối tài nguyên với YARN:

- ResourceManager quản lý tài nguyên trong cụm như CPU, RAM, băng thông mạng.
- NodeManager trên mỗi máy chủ theo dõi và cung cấp tài nguyên cho các công việc được giao.
- YARN phân bổ tài nguyên hiệu quả cho các ứng dụng chạy trên Hadoop.

2.3.9. HDFS

HDFS (Hadoop Distributed File System) là một hệ thống file phân tán được thiết kế để lưu trữ dữ liệu. HDFS được phát triển như một phần của dự án Hadoop, và là một thành phần chính trong hệ sinh thái Big Data của Facebook. HDFS được thiết kế để có khả

năng chịu lỗi cao bằng cách chia nhỏ các tệp lớn thành các khối nhỏ và lưu trữ các khối này trên các máy chủ khác nhau. Mỗi khối dữ liệu có kích thước mặc định là 128MB, được sao chép và lưu trữ ở nhiều vị trí để đảm bảo tính sẵn sàng và khả năng phục hồi. Điều này giúp đảm bảo dữ liệu không bị mất mát ngay cả khi một số máy chủ gặp sự cố.

HDFS sử dụng kiến trúc Master/Slave (Nguyễn Mạnh, 2024), gồm một NameNode (nút tên) và nhiều DataNode (nút dữ liệu):

- NameNode là một trung tâm điều khiển của HDFS, lưu trữ thông tin về vị trí và trạng thái các khối dữ liệu.
- DataNode là các nút lưu trữ dữ liệu thực sự và phân tán dữ liệu trên các nút khác nhau, theo sự quản lý của NameNode.

Cách hoạt động của HDFS:

Khi ghi dữ liệu vào HDFS:

- Client gửi yêu cầu ghi tệp tới NameNode.
- NameNode ghi lại metadata của tệp, phân chia dữ liệu thành các khối và gán vị trí lưu trữ cho các khối trên DataNodes.
- Client ghi dữ liệu trực tiếp vào các DataNodes theo chỉ định của NameNode.

Khi đọc dữ liệu từ HDFS:

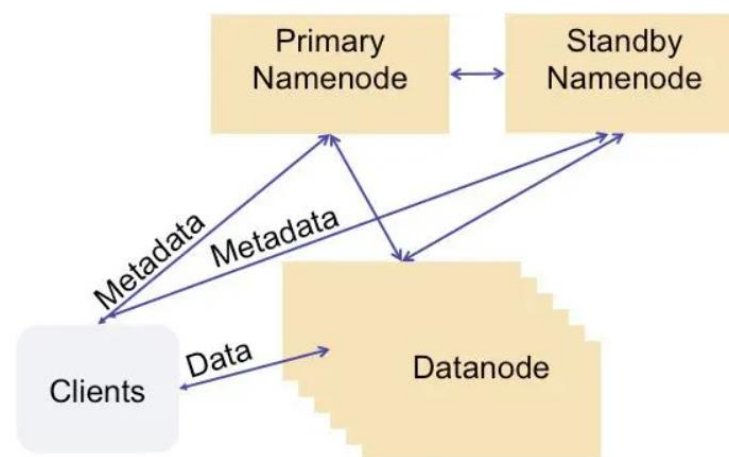
- Client gửi yêu cầu đọc tệp tới NameNode.
- NameNode trả về vị trí lưu trữ các khối dữ liệu của tệp đó trên DataNodes.
- Client đọc song song dữ liệu từ các DataNodes lưu trữ các khối tương ứng.

Ứng dụng của HDFS tại Facebook:

HDFS được Facebook sử dụng để lưu trữ dữ liệu trong Data Warehouse. HDFS giúp Facebook chia nhỏ và lưu trữ dữ liệu trên nhiều máy chủ khác nhau thay vì chỉ một máy

chủ duy nhất. Điều này giúp Facebook có thể lưu trữ một lượng dữ liệu cực kỳ lớn và phân tán việc truy cập dữ liệu, tránh quá tải cho một máy chủ. Tuy nhiên, nếu NameNode (Master) gặp sự cố, toàn bộ hệ thống sẽ bị ảnh hưởng và không thể truy cập dữ liệu. Facebook nhận thấy rằng NameNode là một "điểm yếu" của HDFS, gây ra nhiều sự cố và thời gian ngừng hoạt động không mong muốn trong Data Warehouse. Vì vậy, họ đang nghiên cứu cách để có nhiều hơn một NameNode, giúp hệ thống HDFS trở nên mạnh mẽ và ổn định hơn. Để giải quyết nhược điểm kiến trúc của NameNode đơn lẻ, Facebook bắt đầu làm việc trên AvatarNode. Cụ thể, AvatarNode có hai máy chủ NameNode - một NameNode chính và một NameNode dự phòng. Cả hai NameNode này đều lưu giữ thông tin về toàn bộ dữ liệu trong hệ thống.

Trong trường hợp bình thường, NameNode chính sẽ đóng vai trò quản lý dữ liệu. Nhưng nếu NameNode chính gặp sự cố, NameNode dự phòng sẽ tự động nhận vai trò quản lý để thay thế, đảm bảo dữ liệu vẫn có thể được truy cập và xử lý.



**Simplified HDFS Architecture:
Highly Available Namenode**

Hình 2.3: Kiến trúc HDFS. Nguồn: (Ryan, 2012)

2.3.10. HBase

HBase là một cơ sở dữ liệu NoSQL nguồn mở dùng để lưu trữ và xử lý dữ liệu lớn. HBase được xây dựng trên nền tảng phần mềm nguồn mở Hadoop, sử dụng mô hình lưu trữ dữ liệu kiểu cặp khóa-giá trị (key-value store). Mục tiêu của HBase là lưu trữ các bảng lớn với hàng tỷ hàng và hàng triệu cột. Nó có thể lưu trữ lượng dữ liệu khổng lồ ở định dạng bảng để đọc và ghi cực nhanh.

HBase có khả năng lưu trữ và truy xuất nhanh chóng các dữ liệu có cấu trúc hoặc bán cấu trúc, như dữ liệu web, dữ liệu cảm biến, dữ liệu đa phương tiện và dữ liệu phân tích. Điều này giúp HBase trở thành một công cụ quan trọng trong việc xây dựng các ứng dụng đám mây lớn, phân tán và ứng dụng phân tích dữ liệu lớn.

Cách thức hoạt động của HBase:

HBase hoạt động dựa trên mô hình phân cụm phân tán (distributed cluster model). Mỗi cụm HBase bao gồm các thành phần chính sau:

HMaster

- Là trạm quản lý chính của cụm HBase.
- Quản lý vòng đời của các RegionServer trong cụm.
- Phân bổ các vùng dữ liệu (regions) cho các RegionServer.
- Khôi phục dữ liệu khi có sự cố xảy ra với RegionServer.
- Điều phối quá trình tải dữ liệu vào HBase.

RegionServer

- Chịu trách nhiệm lưu trữ và xử lý các yêu cầu đọc/ghi dữ liệu từ client.
- Mỗi RegionServer quản lý một hoặc nhiều vùng dữ liệu (regions) của các bảng HBase.
- Khi nhận yêu cầu từ client, RegionServer sẽ tìm kiếm và thực hiện các thao tác đọc/ghi trên vùng dữ liệu tương ứng.
- Các RegionServer giao tiếp với HDFS để lưu trữ dữ liệu vào các tệp trên HDFS.

Zookeeper

- Là một dịch vụ phối hợp và quản lý cấu hình cho cụm HBase.
- Lưu trữ thông tin về vị trí của các vùng dữ liệu (regions) và các RegionServer quản lý chúng.
- Cung cấp dịch vụ đăng ký và phát hiện cho các thành phần khác trong cụm HBase.
- Đồng bộ hóa các cấu hình và thông tin trạng thái giữa các nút trong cụm.

Client

- Là ứng dụng hoặc người dùng gửi yêu cầu đọc/ghi dữ liệu tới HBase.
- Client sẽ giao tiếp với Zookeeper để lấy thông tin về vị trí của các vùng dữ liệu và RegionServer tương ứng.
- Sau đó, client gửi trực tiếp yêu cầu đến RegionServer quản lý vùng dữ liệu đó.

Về cách lưu trữ dữ liệu, HBase sử dụng mô hình dữ liệu kiểu cặp khóa-giá trị (key-value store). Dữ liệu được tổ chức thành các bảng, mỗi bảng gồm nhiều hàng (rows) và cột (columns). Mỗi hàng được xác định bởi một khóa duy nhất (row key), và mỗi cột được định danh bởi một cặp column family và column qualifier. Giá trị của mỗi ô dữ liệu (cell) được lưu trữ cùng với một thẻ thời gian (timestamp), cho phép lưu trữ nhiều phiên bản của cùng một giá trị theo thời gian.

HBase sử dụng HDFS (Hadoop Distributed File System) làm hệ thống lưu trữ phân tán. Dữ liệu trong HBase được chia thành các vùng dữ liệu (regions) và lưu trữ trên nhiều nút (nodes) trong cụm HDFS, giúp đảm bảo khả năng mở rộng, tính sẵn sàng cao và khả năng chịu lỗi.

2.4. Cài đặt và sử dụng HBase

2.4.1. Mô tả

Phần này sẽ minh họa cách sử dụng HBase, một cơ sở dữ liệu NoSQL phân tán và mở rộng theo chiều ngang, để lưu trữ và truy xuất thông tin người dùng, bao gồm tên, email và danh sách bạn bè. Ví dụ này mô phỏng một tính năng cơ bản của Facebook, có thể được mở rộng để bao gồm các tính năng khác như đăng bài viết, bình luận, thích, v.v.

2.4.2. Cài đặt

Cài đặt Hadoop

Đầu tiên, ta cần cập nhật các gói phần mềm và cài đặt OpenJDK 8 (Java Development Kit 8), đây là yêu cầu để chạy Hadoop và HBase.

```
# Cập nhật các gói phần mềm và cài đặt OpenJDK 8
!apt-get update > /dev/null
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

Sau đó, tải về phiên bản mới nhất của Hadoop (3.3.5), giải nén và di chuyển thư mục Hadoop vào /usr/local/.

```
!wget -q
https://downloads.apache.org/hadoop/common/hadoop-
3.3.5/hadoop-3.3.5.tar.gz
!tar -xzf hadoop-3.3.5.tar.gz
!mv hadoop-3.3.5/ /usr/local/
```

Thiết lập các biến môi trường JAVA_HOME, HADOOP_HOME và thêm đường dẫn đến thư mục bin của Hadoop vào biến PATH.

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-
amd64"
os.environ["HADOOP_HOME"] = "/usr/local/hadoop-3.3.5/"
current_path = os.getenv('PATH')
new_path = current_path+':/usr/local/hadoop-3.3.5/bin/'
os.environ["PATH"] = new_path
```

Cài đặt HBase

Tải về phiên bản mới nhất của HBase (2.5.8) và giải nén.

```
!wget -q https://downloads.apache.org/hbase/2.5.8/hbase-
2.5.8-bin.tar.gz
!tar xzf hbase-2.5.8-bin.tar.gz
```

Thiết lập biến môi trường "HBASE_HOME" và thêm đường dẫn đến thư mục bin của HBase vào biến "PATH".

```
os.environ["HBASE_HOME"] = "/content/hbase-2.5.8/"
!echo $HBASE_HOME
current_path = os.getenv("PATH")
```

```
new_path = current_path+':/content/hbase-2.5.8/bin'

os.environ["PATH"] = new_path
```

In ra các biến môi trường để kiểm tra cấu hình.

```
!echo $PATH

!echo $JAVA_HOME

!echo $HADOOP_HOME

!echo $HBASE_HOME
```

```
/content/hbase-2.5.8/
/opt/bin:/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/tools/node/bin:/tools/google-cloud-sdk/bin:/usr/local/hadoop-3.3.5/
/usr/lib/jvm/java-8-openjdk-amd64
/usr/local/hadoop-3.3.5/
/content/hbase-2.5.8/
```

Kiểm tra xem thư viện slf4j (Simple Logging Facade for Java) đã được bao gồm trong Hadoop và HBase hay chưa.

```
!find /content/hbase-2.5.8/lib -name slf4j*

!ls /content/hbase-2.5.8/lib/client-facing-
thirdparty/slf4j-api-1.7.33.jar

!ls $HADOOP_HOME/share/hadoop/common/lib/*slf4j*
```

Di chuyển các file slf4j ra khỏi thư mục lib của Hadoop để tránh xung đột logging.

```
!mv /usr/local/hadoop-
3.3.5//share/hadoop/common/lib/slf4j-api-
1.7.36.jar.disable /usr/local/hadoop-
3.3.5//share/hadoop/common/lib/slf4j-api-1.7.36.jar

!mv /usr/local/hadoop-
3.3.5//share/hadoop/common/lib/slf4j-reload4j-
1.7.36.jar.disable /usr/local/hadoop-
3.3.5//share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar
```

Cài đặt thư viện Python happybase để tương tác với HBase.

```
!pip -qq install happybase  
  
import happybase
```

Khởi động HBase và Thrift Server (giao thức giao tiếp giữa Python và HBase), kiểm tra các tiến trình Java đang chạy bằng lệnh jps.

```
!hbase-daemon.sh start thrift  
  
!start-hbase.sh  
  
!jps
```

Sử dụng HBase

Khởi tạo kết nối với HBase chạy trên localhost và cổng 9090 (cổng mặc định của Thrift Server).

```
connection =  
happybase.Connection('localhost', 9090, autoconnect=True)
```

Tạo kết nối với HBase chạy trên localhost và mở kết nối.

```
connection.open()  
  
connection.tables()
```

Tạo bảng users với hai cột: personal (chứa thông tin cá nhân người dùng) và friends (chứa danh sách bạn bè).

```
connection.open()  
  
# Tạo bảng "users"  
  
connection.create_table(  
    'users',  
    {'personal': dict(),
```

```

        'friends': dict()

    }

)

connection.close()

```

Lấy tham chiếu đến bảng users để thực hiện các thao tác.

```

connection.open()

table = connection.table('users')

connection.close()

```

Chèn thông tin người dùng Vi và Mike vào bảng, bao gồm tên và email (lưu trong cột personal).

```

connection.open()

table.put(b'user1', {b'personal:name': b'Vi',
b'personal:email': b'vi@example.com'})

table.put(b'user2', {b'personal:name': b'Mike',
b'personal:email': b'mike@example.com'})

```

Chèn thông tin về bạn bè của Vi (bạn với Mike) và bạn bè của Mike (bạn với Vi) vào cột friends.

```

table.put(b'user1', {b'friends:user2': b'Mike'})

table.put(b'user2', {b'friends:user1': b'Vi'})

connection.close()

```

Truy xuất thông tin người dùng Vi và Mike từ bảng, in ra tên, email và danh sách bạn bè của họ.

```

connection.open()

```

```
user1 = table.row(b'user1')

print(f"User: {user1[b'personal:name']} - Email: {user1[b'personal:email']}")

print(f"Friends: {user1[b'friends:user2']}")

connection.close()
```

User: b'Vi' - Email: b'vi@example.com'
Friends: b'Mike'

```
connection.open()

user2 = table.row(b'user2')

print(f"User: {user2[b'personal:name']} - Email: {user2[b'personal:email']}")

print(f"Friends: {user2[b'friends:user1']}")

connection.close()
```

User: b'Mike' - Email: b'mike@example.com'
Friends: b'Vi'

CHƯƠNG 3: ỨNG DỤNG DỮ LIỆU LỚN Ở SPOTIFY

3.1. Giới thiệu doanh nghiệp Spotify

Spotify được thành lập vào năm 2006, là một nền tảng nghe nhạc trực tuyến lớn nhất thế giới với hơn 406 triệu người dùng đăng ký, trong đó có 180 triệu người dùng premium trả phí. Với lượng người dùng khổng lồ như vậy, Spotify tạo ra một khối lượng dữ liệu lớn vô cùng đáng kinh ngạc mỗi ngày.

Nền tảng này cung cấp một thư viện âm nhạc rộng lớn trải dài trên nhiều thể loại, ngôn ngữ và văn hóa. Người dùng có thể nghe bài hát yêu thích của mình mọi lúc, mọi nơi, cho dù đó là trên máy tính hay điện thoại thông minh.

Một trong những tính năng chính của Spotify là khả năng cá nhân hóa trải nghiệm nghe nhạc cho từng người dùng. Spotify phải lưu trữ và xử lý hàng tỷ lượt phát nhạc, podcast và album từ hàng trăm triệu người dùng trên khắp thế giới. Dữ liệu này bao gồm thông tin chi tiết về thời gian, vị trí, thiết bị nghe và hành vi lắng nghe của người dùng. Bằng cách tận dụng dữ liệu lớn, Spotify hiểu được sở thích cá nhân, sở thích âm nhạc và thói quen nghe nhạc. Điều này cho phép nền tảng cung cấp danh sách phát và đề xuất được chọn lọc kỹ lưỡng, phù hợp với sở thích âm nhạc riêng của từng người dùng.

Ngoài ra, Spotify còn thu thập và phân tích dữ liệu về nội dung âm nhạc và podcast từ hàng triệu nghệ sĩ và nhà sản xuất trên nền tảng. Các thuộc tính như thể loại, ca sĩ, tốc độ, cảm xúc,... được mã hóa và khai thác để giúp tối ưu trải nghiệm nghe cho người dùng. Dữ liệu lớn cũng giúp Spotify phát hiện các mẫu hình và xu hướng âm nhạc mới, xác định các nghệ sĩ tiềm năng và tạo ra các playlist phù hợp. Ngoài ra, phân tích dữ liệu cũng được sử dụng để đề xuất nhạc mới, âm nhạc độc quyền và cải thiện dịch vụ của Spotify.

Để xử lý khối lượng dữ liệu khổng lồ này, Spotify đã xây dựng một nền tảng dữ liệu lớn đặc biệt sử dụng công nghệ điện toán đám mây hiện đại và các kỹ thuật máy học, trí tuệ nhân tạo. Điều này cho phép Spotify phân tích và khai thác dữ liệu một cách hiệu quả để cải thiện trải nghiệm người dùng và dịch vụ của mình.

Cơ sở hạ tầng dữ liệu của Spotify:

- 1300 node Hadoop
- 42 PB dung lượng lưu trữ
- 30 TB dữ liệu đi vào qua Kafka mỗi ngày
- 400 TB dữ liệu được xử lý bằng Hadoop mỗi ngày

Spotify sử dụng một cơ sở hạ tầng dữ liệu lớn tiên tiến và đa dạng để đáp ứng nhu cầu xử lý khối lượng dữ liệu khổng lồ từ hàng trăm triệu người dùng. Một số thành phần chính trong hệ thống này bao gồm:

- Hệ thống lưu trữ dữ liệu phân tán:

Spotify sử dụng hệ thống lưu trữ dữ liệu phân tán dựa trên Hadoop và công nghệ Dremio để lưu trữ và xử lý dữ liệu tại chỗ. Hệ thống này cho phép lưu trữ và truy xuất dữ liệu một cách liền mạch và hiệu quả từ nhiều nguồn khác nhau.

- Hệ thống xử lý dữ liệu luồng (Stream Processing):

Kafka được sử dụng để xử lý luồng dữ liệu phát trực tuyến từ hàng trăm triệu lượt phát nhạc mỗi ngày. Các công nghệ như Apache Samza và Druid giúp phân tích dữ liệu luồng theo thời gian thực.

- Hệ thống phân tích dữ liệu batch:

Apache Spark được sử dụng để xử lý và phân tích hàng chục petabyte dữ liệu batch mỗi ngày. Các phân tích này bao gồm khuyến nghị nhạc, phát hiện xu hướng âm nhạc và báo cáo cho nghệ sĩ.

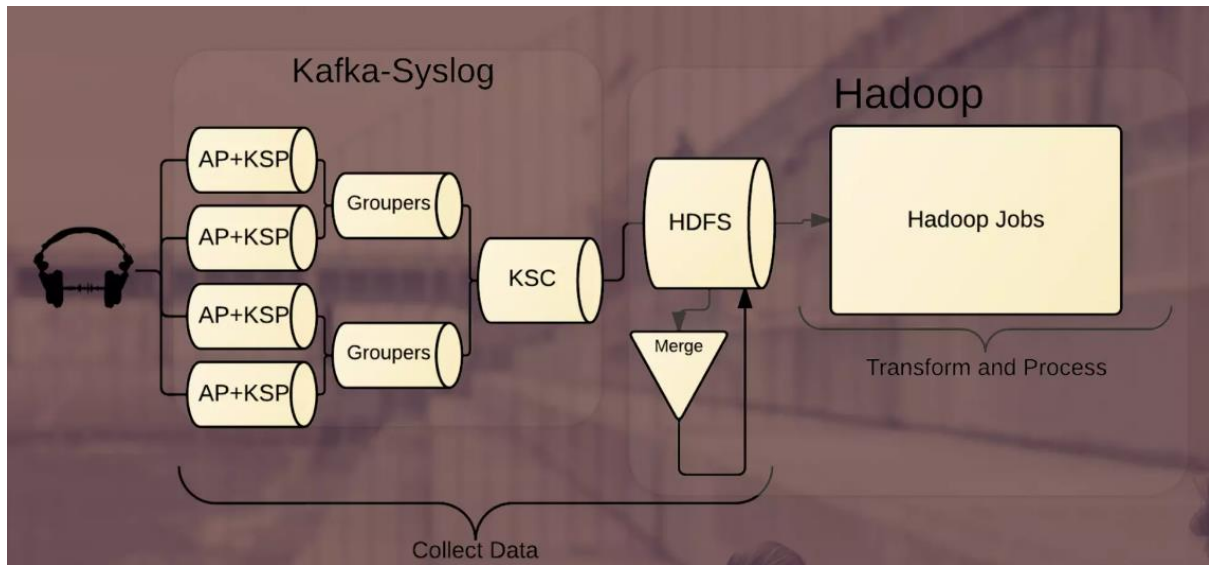
- Cơ sở dữ liệu quan hệ và NoSQL:

PostgreSQL và Cassandra lưu trữ dữ liệu người dùng, đăng nhập và phân quyền. Redis được sử dụng làm bộ nhớ đệm và hàng đợi bất đồng bộ.

- Dịch vụ người dùng:

Các dịch vụ người dùng cuối được triển khai trên nền tảng đám mây AWS và Google Cloud để đảm bảo khả năng mở rộng và đáp ứng nhu cầu toàn cầu.

3.2. Quy trình thu thập dữ liệu ở Spotify



Hình 3.1: Quy trình thu thập dữ liệu của Spotify (Baer, 2015)

- Dữ liệu từ các AP+KSP (Access Points + Kafka Syslog Producers) được thu thập và nhóm lại bởi các Groupers.
- Dữ liệu sau đó được chuyển đến KSC (Kafka Syslog Collectors) để lưu trữ tạm thời.
- Dữ liệu từ KSC được lưu trữ trong HDFS.
- Dữ liệu trong HDFS được hợp nhất định kỳ để đảm bảo tính nhất quán.
- Các công việc Hadoop xử lý và biến đổi dữ liệu để trích xuất thông tin cần thiết.

Toàn bộ quy trình này giúp Spotify thu thập, lưu trữ khối lượng dữ liệu lớn một cách hiệu quả và chính xác, giúp họ cải thiện dịch vụ và trải nghiệm người dùng.

3.3. Quy trình xử lý và lưu trữ dữ liệu ở Spotify

Spotify sử dụng Hadoop như một phần của cơ sở hạ tầng dữ liệu mở rộng để quản lý và xử lý khối lượng dữ liệu lớn do người dùng của họ tạo ra. Spotify có một cụm Apache Hadoop lớn nằm trong cơ sở hạ tầng nội bộ, được coi là một trong những cụm lớn nhất ở Châu Âu. Cụm này chạy hơn 20.000 luồng việc mỗi ngày, xử lý các nhiệm vụ như báo cáo kinh doanh, đề xuất âm nhạc, phân phối quảng cáo, và cung cấp thông tin chi tiết cho nghệ sĩ.

Để đáp ứng lượng dữ liệu lớn, cụm Hadoop của Spotify bao gồm khoảng 2.500 node. Cụm này xử lý dữ liệu cho hàng tỷ lượt phát nhạc trên nhiều thị trường khác nhau và thêm hàng ngàn bài hát mới vào danh mục mỗi ngày.

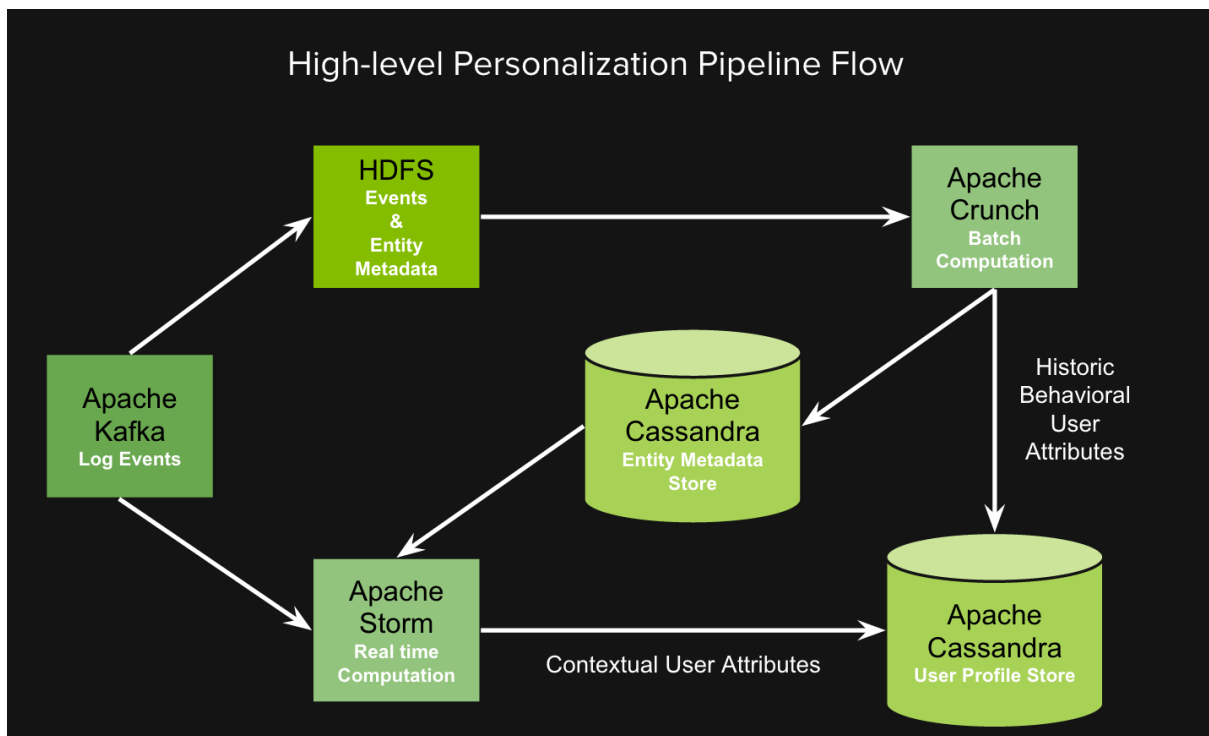
Spotify sử dụng data lake trên Hadoop để thực hiện các phân tích phức tạp, chẳng hạn như tính toán tiền bản quyền, đề xuất bài hát cho người dùng, và đo lường phản hồi của khán giả đối với các tính năng và chức năng mới.

Quá trình xử lý dữ liệu ở Spotify sử dụng:

- Kafka để thu thập log (các lượt nghe nhạc)
- Storm để xử lý sự kiện theo thời gian thực
- Crunch để chạy các công việc MapReduce theo batch trên Hadoop
- Cassandra để lưu trữ các thuộc tính hồ sơ người dùng và siêu dữ liệu về các thực thể như playlist, nghệ sĩ...

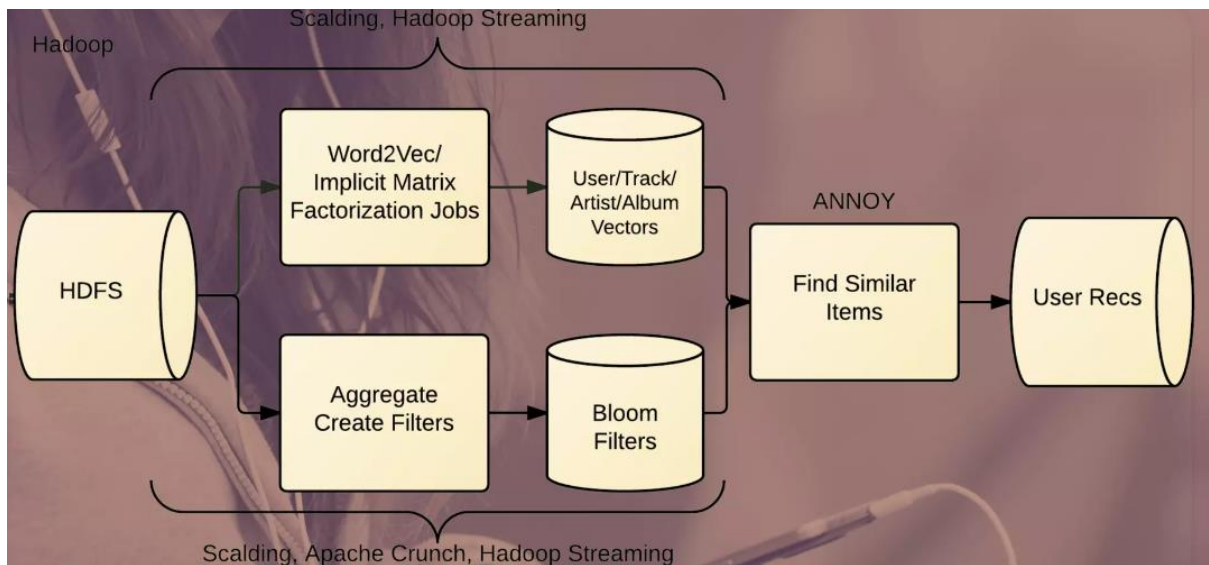
Quá trình này diễn ra như sau:

Dữ liệu tĩnh được lưu vào HDFS (là hệ thống tệp được phân tán và dùng để lưu trữ dữ liệu lớn của Hadoop), dữ liệu luồng (log) được gửi tới Kafka. Crunch đọc dữ liệu từ HDFS, thực hiện các xử lý như loại bỏ dữ liệu trùng lặp, lọc các trường không mong muốn và lưu kết quả vào cơ sở dữ liệu Cassandra. Storm nhận luồng tin từ Kafka, thực hiện xử lý theo thời gian thực và cũng lưu kết quả vào Cassandra. Cassandra trở thành nguồn dữ liệu tập trung để các hệ thống khác của Spotify xử lý.



Hình 3.2: Quy trình xử lý dữ liệu ở Spotify (Mishra & Brown, 2015)

Ngoài việc xử lý dữ liệu theo từng lô và theo luồng, Spotify còn thực hiện nhiều phân tích ad-hoc sử dụng Apache Hive. Hive cho phép các nhà phân tích kinh doanh và quản lý sản phẩm dễ dàng phân tích lượng dữ liệu khổng lồ bằng các truy vấn tương tự SQL. Tuy nhiên, các truy vấn Hive sau đó được chuyển thành MapReduce, dẫn đến gây ra nhiều chi phí (như kéo dài thời gian xử lý dữ liệu, hạn chế băng thông, tốn kém nhiều bộ nhớ và CPU...). Thêm vào đó, Spotify lưu trữ hầu hết dữ liệu của mình trong các tệp Avro (hệ thống tuần tự hóa dữ liệu) theo từng hàng, nghĩa là bất kỳ truy vấn nào, dù chọn cột nào, cũng yêu cầu quét toàn bộ các tệp đầu vào. Do đó, việc sử dụng Hive để xử lý có tốc độ còn hạn chế. Sau này, Spotify đã chuyển sang dùng Google BigQuery cho hầu hết các trường hợp sử dụng truy vấn ad-hoc và đã trải nghiệm sự cải thiện đáng kể về hiệu suất.



Hình 3.3: Sơ đồ minh họa hệ thống đề xuất cho người dùng của Spotify (Baer, 2015)

Dữ liệu của người dùng, ví dụ như bài hát/nghe sĩ/album đã nghe sẽ được lưu trữ ở HDFS. Sau đó những dữ liệu này được xử lý bằng Hadoop Streaming, Scalding, Crunch để tính toán biểu diễn vector. Tiếp đến, Spotify sử dụng thư viện ANNOY để tìm các đề xuất tương tự (bài hát/nghe sĩ/album) dựa trên các biểu diễn vector đã tính toán ở trước, từ đó tạo ra danh sách phát hoặc những bài hát/nghe sĩ yêu thích và đề xuất cho người dùng như ta thấy hiện nay.

3.4. Cài đặt và sử dụng MapReduce

3.4.1. Mô tả

Ví dụ dưới đây được sử dụng để phân tích số lần phát các bài hát mô phỏng trên dữ liệu của Spotify bằng mô hình lập trình MapReduce thông qua Hadoop Streaming.

3.4.2. Cài đặt

Sau khi cài đặt Hadoop, ta lần lượt tạo file mapper.py (bước Map) và file reducer.py (bước Reduce) như dưới đây:

Ví dụ dữ liệu đầu vào để thực hiện MapReduce gồm 3 cột như sau:

user_id, song_id, timestamp

u1, s1, 2024-05-01 12:00:00

u2, s2, 2024-05-01 12:01:00

u1, s2, 2024-05-01 12:02:00

u3, s1, 2024-05-01 12:03:00

Giai đoạn Map: File mapper.py xử lý từng dòng dữ liệu đầu vào để phát ra các cặp key-value, trong đó key là song_id và value là 1. Điều này giúp đếm số lượt phát cho mỗi bài hát.

Input: Mỗi dòng của một tệp CSV đại diện cho số lần phát nhạc.

Output: <song_id, 1> cho mỗi lần phát nhạc.

```
#!/usr/bin/env python
import sys
# Đầu vào đến từ STDIN (standard input)
for line in sys.stdin:
    # Loại bỏ khoảng trắng ở đầu và cuối dòng
    line = line.strip()

    # Tách dòng thành các trường
    fields = line.split(',')
    if len(fields) == 3:
        # Lấy song_id
        song_id = fields[1].strip()

        # Ghi kết quả ra STDOUT (standard output);
        # Những gì chúng ta xuất ra ở đây sẽ là đầu vào cho
        Reducer

        print(f'{song_id}\t1')
```

Đây là kết quả sau khi chạy xong bước Map:

s1 1

s2 1

s2 1

s1 1

Giai đoạn Reduce: File reducer.py tổng hợp đầu ra từ mapper bằng cách cộng các giá trị cho mỗi song_id, kết quả là tổng số lượt phát cho mỗi bài hát.

Input: Các cặp key-value trong đó key là song_id và value là số lượt (từ mapper).

Output: <song_id, total_count> cho mỗi bài hát.

```
current_song_id = None
current_count = 0
song_id = None
# Đầu vào đến từ STDIN
for line in sys.stdin:
    # Loại bỏ khoảng trắng ở đầu và cuối dòng
    line = line.strip()

    # Phân tích đầu vào mà chúng ta nhận được từ mapper.py
    song_id, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        # Nếu count không phải là một số, thì bỏ qua dòng này
        continue

    # Kiểm tra này chỉ hoạt động vì Hadoop sắp xếp đầu ra
    của mapper

    # theo key (ở đây là song_id) trước khi truyền nó đến
    reducer

    if current_song_id == song_id:
        current_count += count
    else:
```



```
if current_song_id:
    # Ghi kết quả ra STDOUT
    print(f'{current_song_id}\t{current_count}')
    current_count = count
    current_song_id = song_id

# Xuất ra số lần phát của bài hát
if current_song_id == song_id:
    print(f'{current_song_id}\t{current_count}')
```

Đây là kết quả sau khi chạy xong bước Reduce:

s1 2

s2 2

Kết quả sau khi thực hiện MapReduce cho dữ liệu ví dụ ở trên là số lần mỗi bài hát được phát. Ở đây cả bài hát s1 và s2 đều được nghe 2 lần.

CHƯƠNG 4: CƠ SỞ LÝ THUYẾT VÀ Q&A

4.1. Tokenization

Tokenization là một bước tiền xử lý văn bản quan trọng trong NLP, được sử dụng để phân chia văn bản thành các đơn vị nhỏ hơn gọi là tokens. Tokens có thể là từ (word), từ phụ (subword) hoặc ký tự (character).

Mục đích:

- Chuẩn bị dữ liệu cho các mô hình NLP: Tokenization giúp chia nhỏ văn bản thành các đơn vị cơ bản mà các mô hình NLP có thể dễ dàng xử lý và phân tích.
- Tạo từ vựng: Từ các tokens, ta có thể xây dựng từ vựng cho kho ngữ liệu, giúp mô hình học cách biểu diễn và phân biệt các từ trong ngôn ngữ.
- Tiền xử lý văn bản: Tokenization có thể được sử dụng để loại bỏ các ký tự đặc biệt, chuẩn hóa văn bản, v.v., giúp cải thiện hiệu quả của các mô hình NLP.

Các loại tokenization phổ biến:

- Tokenization dựa trên từ (Word-based tokenization): Đây là phương pháp đơn giản nhất, chia văn bản thành các từ dựa trên khoảng trắng. Tuy nhiên, phương pháp này có thể gặp hạn chế với các ngôn ngữ không có khoảng trắng rõ ràng hoặc các từ ghép phức tạp.
- Tokenization dựa trên từ phụ (Subword-based tokenization): Phương pháp này chia nhỏ từ thành các đơn vị nhỏ hơn gọi là từ phụ, giúp mô hình học tốt hơn các từ hiếm gặp hoặc các từ ghép phức tạp. Một số thuật toán tokenization dựa trên từ phụ phổ biến bao gồm BPE (Byte Pair Encoding) và WordPiece.
- Tokenization dựa trên ký tự (Character-based tokenization): Phương pháp này chia văn bản thành từng ký tự riêng lẻ. Phương pháp này có thể hữu ích cho các ngôn ngữ có cấu trúc phức tạp hoặc các nhiệm vụ NLP đòi hỏi độ chính xác cao.

Ví dụ:

Câu "Hãy cùng học tokenization." có thể được tokenized theo các cách sau:

- Tokenization dựa trên từ: ["Hãy", "cùng", "học", "tokenization."]
- Tokenization dựa trên từ phụ: ["Hãy", "cùng", "học", "token", "ization."]
- Tokenization dựa trên ký tự: ['H', 'a', 'y', ' ', 'c', 'ù', 'n', 'g', ' ', 'h', 'ạ', 'c', ' ', 't', 'o', 'k', 'e', 'n', 'i', 'z', 'a', 't', 'i', 'o', 'n', '.']

Lựa chọn phương pháp tokenization:

Phương pháp tokenization phù hợp sẽ phụ thuộc vào từng nhiệm vụ NLP cụ thể và loại mô hình được sử dụng. Nên cân nhắc các yếu tố như độ phức tạp của ngôn ngữ, kích thước kho ngữ liệu, và yêu cầu về độ chính xác của mô hình.

Lưu ý:

- Tokenization là một bước quan trọng trong NLP nhưng nó không phải là bước duy nhất. Sau khi tokenization, văn bản cần được xử lý thêm bằng các kỹ thuật khác như stemming, lemmatization, v.v. trước khi đưa vào mô hình NLP.
- Có nhiều thư viện NLP cung cấp các hàm tokenization cho các ngôn ngữ khác nhau.

4.2. Cơ chế Attention Mechanism và Transformers

Transformer là một dòng mạng nơ ron nhân tạo đang ngày càng trở nên phổ biến. Transformer được sử dụng bởi GPT-2 của OpenAI hay trong AlphaStar của DeepMind - một chương trình có khả năng đánh bại những người chơi đỉnh cao của Starcraft.

Transformers được phát triển để xử lý bài toán chuyển đổi chuỗi và dịch máy. Điều này có nghĩa là chúng có khả năng làm việc với mọi nhiệm vụ liên quan đến chuyển đổi một chuỗi đầu vào thành một chuỗi đầu ra. Các ứng dụng điển hình bao gồm nhận dạng giọng nói, chuyển đổi văn bản thành giọng nói và nhiều ứng dụng trí tuệ nhân tạo khác.

Để mô hình có thể thực hiện chuyển đổi chuỗi, nó cần có một dạng bộ nhớ nào đó. Ví dụ như mô hình cần dịch câu nói “Minh là người Việt Nam, hiện tại anh ấy đang sống

tại Hà Nội “. Để thu nhận và chuyển đổi được chuỗi văn bản này, nó cần biết được Minh và anh ấy có mối quan hệ đồng nhất với nhau.

Để có thể dịch được câu như vậy, mô hình cần tìm ra được mối liên hệ ràng buộc và phụ thuộc giữa các thực thể trong câu. Mạng nơ ron hồi quy (RNN) và mạng nơ ron tích chập (CNN) có những đặc điểm phù hợp để giải quyết vấn đề này. Tuy nhiên những mô hình đó vẫn tồn tại những hạn chế như chúng chỉ xử lý các từ một cách tuần tự. Đó là một rào cản nếu như muốn tính toán song song.

Để khắc phục tình trạng trên, các nhà nghiên cứu đã phát triển một phương pháp giúp mô hình tập trung vào từng từ cụ thể, tương tự như cách con người làm. Phương pháp này được gọi là "Attention mechanism"

Attention là thành phần không thể thiếu trong kiến trúc của Transformer. Nó cho phép mô hình tập trung vào các phần quan trọng của chuỗi đầu vào và tạo ra kết quả dựa trên sự tương quan giữa các yếu tố. Điều này đóng vai trò quan trọng trong việc cải thiện khả năng hiểu và sản sinh ngôn ngữ tự nhiên.

Trong kiến trúc Transformer, phương pháp Attention có tên gọi là "Scaled Dot-Product Attention", được sử dụng rộng rãi trong các phần encoder, decoder và layer-to-layer của mô hình. Nó ước lượng mức độ tương quan giữa các từ trong chuỗi dựa vào độ quan trọng của chúng. Quá trình Attention này giúp mô hình tập trung vào các thành phần quan trọng và tạo ra các vector biểu diễn cho kết quả cuối cùng.

4.3. Mô hình BERT

4.3.1 Giới thiệu về mô hình BERT

4.3.1.1. Định nghĩa

BERT là một mô hình xử lý ngôn ngữ tự nhiên (NLP) đột phá được phát triển bởi Google. BERT đã thay đổi hoàn toàn việc máy móc hiểu ngôn ngữ. Giờ đây, nhờ có BERT, máy móc đã có thể nắm bắt ngữ cảnh và các sắc thái của ngôn ngữ.

4.3.1.2. Hướng xử lý ngôn ngữ hai chiều của BERT (*bidirectional approach*)

BERT là viết tắt của for Bidirectional Encoder Representations from Transformers. BERT được định nghĩa là mô hình ngôn ngữ hai chiều (*bidirectional*). Điều này có nghĩa là khi phân tích ngữ nghĩa, BERT xem xét cả các từ đứng trước và đứng sau nó để hiểu hơn về mối quan hệ của các từ trong ngữ cảnh. Đây là điểm khiến BERT vượt trội hơn hẳn các mô hình ngôn ngữ truyền thống.

Cụ thể, các mô hình ngôn ngữ truyền thống xử lý văn bản một cách tuần tự, từ trái sang phải hoặc phải sang trái. Phương pháp này giới hạn khả năng nhận thức của mô hình chỉ trong bối cảnh ngay trước từ mục tiêu. BERT sử dụng cách tiếp cận hai chiều xem xét cả ngữ cảnh bên trái và bên phải của các từ trong câu, thay vì chỉ đơn thuần phân tích văn bản một cách tuần tự, BERT xem xét đồng thời tất cả các từ trong câu.

Ví dụ:

“The bank is situated on the _____ of the river.”

(Tạm dịch: “Ngân hàng nằm ở _____ của con sông.”)

Trong cách xử lý văn bản một chiều (từ trái sang phải hoặc ngược lại), việc hiểu khoảng trống sẽ phụ thuộc rất nhiều vào các từ trước đó và mô hình có thể gặp khó khăn trong việc phân biệt liệu “bank” ám chỉ một tổ chức tài chính hay bên bờ sông.

Ngược lại, với cách xử lý văn bản hai chiều của BERT, cả bối cảnh ở bên trái (“The bank is situated on the” – Tạm dịch: “Ngân hàng nằm trên”) và bối cảnh bên phải (“of the river” – Tạm dịch: “của dòng sông”) đều được xét đến. Chiến lược này giúp mô hình hiểu sắc thái của văn bản rõ hơn, dẫn đến có thể suy luận được rằng từ còn thiếu có khả năng liên quan đến vị trí địa lý của ngân hàng, thể hiện sự phong phú về ngữ cảnh mà cách tiếp cận hai chiều mang lại.

4.3.1.3. Kiến trúc của BERT

Kiến trúc của BERT dựa trên mô hình Transformer. Mô hình Transformer sử dụng các cơ chế self-attention để nắm bắt mối quan hệ giữa các từ trong câu mà không cần dựa vào quá trình xử lý tuần tự, khiến mô hình này hoạt động với hiệu quả cao.

Kiến trúc Transformer của mô hình BERT bao gồm nhiều lớp encoder. Mỗi lớp encoder của mô hình lại sử dụng các cơ chế self – attention kết hợp với các mạng neuron chuyển tiếp (feed-forward neural network) để tạo ra các biểu diễn (dưới dạng vector) theo ngữ cảnh của từ.

Bằng cách kết hợp nhiều lớp encoder, BERT liên tục tinh chỉnh các vector ngữ cảnh từ vựng từ đó có thể nắm bắt cấu trúc ngôn ngữ ngày càng phức tạp. Các lớp encoder kết hợp cùng mạng neuron chuyển tiếp còn cho phép BERT học được các lớp nghĩa khác nhau của câu. Các lớp đầu tiên của BERT sẽ tập trung vào những điều cơ bản như mối liên hệ giữa các từ (ví dụ: "con" đi với "mèo"). Càng lên các lớp sâu hơn, BERT sẽ xử lý những khía cạnh trừu tượng hơn của ngôn ngữ, liên quan đến ý nghĩa của cả câu (ví dụ: xác định câu hỏi, câu cảm thán).

4.3.2. Cách hoạt động của mô hình BERT

Mô hình BERT bao gồm hai bước cốt lõi là: pre-training (tiền huấn luyện) và fine-tuning (tinh chỉnh).

4.3.2.1. Pre-training

4.3.2.1.1. Text - preprocessing

Các dữ liệu cần cho quá trình tiền huấn luyện này là các dữ liệu văn bản không gán nhãn như sách, báo, các trang web. Các dữ liệu này cần trải qua bước tiền xử lý trước khi tiến vào bước tiền huấn luyện

Tokenize dữ liệu

Tại bước tokenize dữ liệu, các văn bản sẽ được chia thành các đơn vị nhỏ hơn gọi là token. Mô hình BERT sử dụng WordPiece làm phương pháp chia nhỏ dữ liệu. WordPiece sẽ tách từ thành các phần nhỏ hơn giúp mô hình BERT tránh khỏi rắc rối khi gặp những từ lạ (Ví dụ từ “running” sẽ được WordPiece tách thành “run” và “ing”).

Với văn bản gốc là “ChatGPT is fascinating.”, WordPiece sẽ tách thành các token: [“Chat”, “##G”, “##PT”, “is”, “fascinating”, “.”].

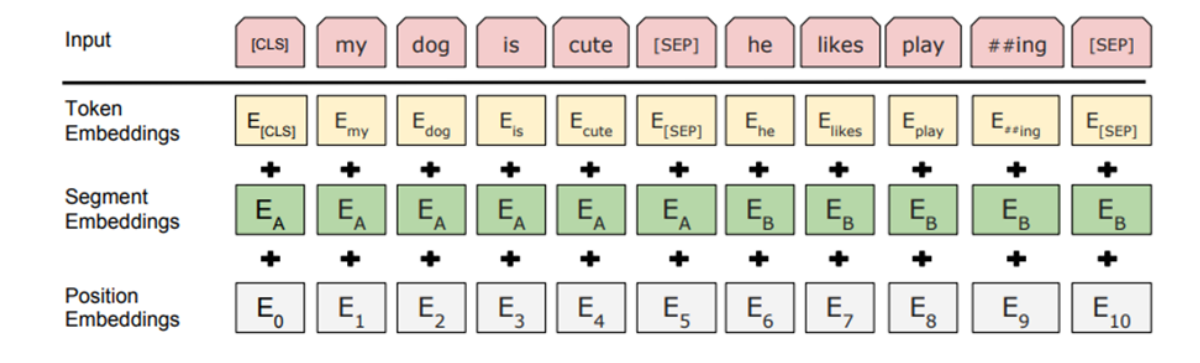
Tinh chỉnh input

Mục tiêu của BERT là nắm rõ ngữ cảnh vậy nên ta cần cung cấp ngữ cảnh văn bản theo cách BERT có thể hiểu. Để thực hiện điều đó, ta sẽ thêm vào các token xác định ngữ cảnh của văn bản. Các token đặc biệt sẽ được thêm vào văn bản để làm rõ ngữ cảnh như: token [CLS] mang tính chất phân loại, đứng ở đầu câu, token [SEP] sử dụng để phân cách câu, thường xuất hiện ở giữa các câu.

Ví dụ với đoạn văn bản “ChatGPT is fascinating.”, input được tinh chỉnh sẽ trở thành “[CLS]”, “Chat”, “##G”, “##PT”, “is”, “fascinating”, “.”, “[SEP]”.

Chuyển input thành các vector biểu diễn ngữ cảnh từ vựng đưa vào encoder

Sau khi đã phân tách và định dạng input cho phù hợp. Các input này sẽ được chuyển hoá thành các vector biểu diễn ngữ cảnh từ vựng. Các vector biểu diễn ngữ cảnh từ vựng là sự kết hợp của các vector sau:



Hình 4.1: Minh hoạ cách tạo thành của các vector biểu diễn ngữ cảnh

- Token Embeddings: Chuyển đổi input đã phân tách và tinh chỉnh định dạng thành dạng vector bằng cách gán cho mỗi token một ID tương ứng với tần suất xuất hiện của nó trong đoạn văn bản.
- Segment Embeddings: Gán thêm một nhãn cho mỗi token để phân biệt chúng thuộc về câu A hay câu B. Nhờ đó, BERT có thể hiểu được ranh giới giữa các câu.
- Positional Embedding: Gán thêm một giá trị cho mỗi token để thể hiện vị trí của nó trong câu. Ví dụ, từ đầu tiên sẽ có giá trị vị trí khác với từ thứ ba.

Sau khi hoàn tất tạo thành các vector ngữ cảnh từ vựng, các vector này sẽ được đưa vào các lớp mạng neuron để BERT có thể học các ngữ cảnh của từng token trong câu.

4.3.2.1.2. Pre-training

Giai đoạn tiền huấn luyện của BERT sẽ tập trung vào việc huấn luyện lượng lớn dữ liệu văn bản không gán. Trong giai đoạn này, mô hình học cách dự đoán các từ còn thiếu trong câu bằng cách xem xét các từ xung quanh. Quá trình này cho phép BERT có được sự hiểu biết sâu sắc về cấu trúc cú pháp và ngữ nghĩa của ngôn ngữ.

Ở giai đoạn này, BERT được huấn luyện trên hai nhiệm vụ quan trọng:

- Masked Language Modeling
- Next Sentence Prediction

Masked Language Modeling

Nhiệm vụ thông thường của mô hình ngôn ngữ là dự đoán từ tiếp theo dựa trên chuỗi các từ trước đó. Tuy nhiên, trong Masked Language Modeling, thay vì dự đoán tất cả các từ tiếp theo, một tỷ lệ phần trăm các từ đầu vào sẽ được che khuất ngẫu nhiên và chỉ những từ bị che khuất này mới được dự đoán.

Khi một số từ trong câu bị che khuất, BERT buộc phải dựa vào ngữ cảnh xung quanh để dự đoán các từ bị thiếu. Đồng thời, việc dự đoán các từ bị che khuất giúp BERT học được các biểu diễn từ phong phú và sâu sắc hơn.

Việc huấn luyện BERT trong giai đoạn tiền huấn luyện bằng Masked Language Modeling sẽ giúp mô hình tăng cường khả năng học ngữ cảnh và cải thiện khả năng biểu diễn của từ.

Next Sentence Prediction

Nhiệm vụ dự đoán câu tiếp theo là một bài toán phân loại nhị phân. Cho một cặp câu, mô hình cần dự đoán xem câu thứ hai có phải là câu tiếp theo thực sự của câu thứ nhất hay không.

Việc huấn luyện mô hình BERT để nhận diện câu tiếp theo xuất hiện trong văn bản là rất hữu ích bởi vì nhiều bài toán Xử lý Ngôn ngữ Tự nhiên sau này, như hệ thống hỏi

đáp (Question Answering System) và Suy luận ngôn ngữ tự nhiên (Natural Language Inference),... yêu cầu sự hiểu biết về mối quan hệ giữa các câu với nhau.

Tóm lại, trong giai đoạn tiền huấn luyện, BERT đã được huấn luyện đồng thời với cả Masked Language Modeling và Next Sentence Prediction để học hỏi và hiểu được ý nghĩa của ngôn ngữ một cách sâu sắc.

```
Input = [CLS] the man went to [MASK] store [SEP]
        he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
        penguin [MASK] are flight ##less birds [SEP]
Label = NotNext
```

Hình 4.2: Minh hoạ cách nhận diện Next Sentence Prediction

4.3.2.2 *Fine-tuning*

Sau quá trình tiền huấn luyện, BERT được tinh chỉnh cho các nhiệm vụ cụ thể, như phân loại văn bản, trả lời câu hỏi và phân tích cảm xúc.

Trong giai đoạn tinh chỉnh này, BERT được huấn luyện trên dữ liệu được dán nhãn cho nhiệm vụ mục tiêu. Điều này cho phép BERT điều chỉnh các vector biểu diễn ngôn ngữ đã học của nó để phù hợp với các yêu cầu cụ thể của nhiệm vụ.

Bằng cách chuyển giao kiến thức thu được trong quá trình tiền huấn luyện cho các nhiệm vụ cụ thể này, BERT đã đạt được hiệu suất đáng kể và trả về những chỉ số tốt khi được ứng dụng để giải quyết các bài toán NLP khác nhau.

4.3.3. Đánh giá và ứng dụng của mô hình BERT

4.3.3.1. Ưu điểm

- BERT đem lại hiệu quả cao cho các nhiệm vụ NLP cụ thể: BERT được huấn luyện trên một lượng lớn dữ liệu chung, giúp nó học được nền tảng vững chắc về ngôn ngữ. Khi được tinh chỉnh cho các nhiệm vụ cụ thể, BERT có thể tận dụng kiến thức nền này để đạt được độ chính xác cao.
- Dễ dàng tinh chỉnh: Các thông số của BERT có thể được tinh chỉnh nhanh chóng với dữ liệu được dán nhãn cho từng nhiệm vụ, giúp tiết kiệm thời gian và nguồn lực so với việc xây dựng mô hình mới từ đầu.
- Kết quả cập nhật: BERT được cập nhật thường xuyên, đảm bảo hiệu suất cao và kết quả tiên tiến.
- Đa ngôn ngữ: BERT có sẵn các phiên bản được huấn luyện trước cho hơn 100 ngôn ngữ, cực kì hữu dụng cho các dự án đa ngôn ngữ.

4.3.3.2. Nhược điểm

- Tốn thời gian tính toán: Việc huấn luyện và sử dụng BERT đòi hỏi nhiều tài nguyên tính toán.
- Phụ thuộc vào tinh chỉnh: BERT được thiết kế để làm nền tảng cho các hệ thống khác, vì vậy nó cần được tinh chỉnh cho từng nhiệm vụ cụ thể, có thể tốn thời gian và công sức.

4.3.3.3. Ứng dụng

BERT được ứng dụng mạnh mẽ trong các lĩnh vực sau:

- Tìm kiếm thông tin: BERT có thể giúp cải thiện kết quả tìm kiếm bằng cách hiểu ngữ cảnh của truy vấn thay vì chỉ khớp các từ khóa.
- Tóm tắt văn bản: BERT có thể tự động tóm tắt các văn bản dài thành các đoạn ngắn hơn, dễ hiểu hơn.
- Phân tích cảm xúc: BERT có thể xác định xem một đoạn văn có ngữ điệu tích cực, tiêu cực hay trung lập.

- Trả lời câu hỏi: BERT có thể được sử dụng để xây dựng các hệ thống trả lời câu hỏi tự động, hiểu được ngữ cảnh của câu hỏi và tìm ra câu trả lời chính xác trong văn bản.

Ngoài ra, BERT còn có nhiều ứng dụng khác trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên.

4.4. XLM-RoBERTa

XLM-RoBERTa (Mô hình Ngôn ngữ Đa Ngôn ngữ - RoBERTa) là một biến thể của mô hình RoBERTa, vốn dựa trên mô hình BERT (Bi-directional Encoder Representations from Transformers - Biểu diễn Mã hóa Hướng hai chiều từ Transformers). XLM-RoBERTa được thiết kế đặc biệt để hiểu đa ngôn ngữ và có thể xử lý nhiều ngôn ngữ khác nhau.

Kiến trúc của XLM-RoBERTa: XLM-RoBERTa bao gồm các thành phần chính sau:

- Phân tách từ (Tokenization): Giống như các mô hình ngôn ngữ khác, XLM-RoBERTa phân tách văn bản đầu vào thành các đơn vị phụ từ (subword) bằng kỹ thuật BPE (Byte Pair Encoding). Điều này cho phép mô hình xử lý cả biểu diễn ở cấp từ và cấp phụ từ.
- Mã hóa Encoder Transformer: XLM-RoBERTa sử dụng một chồng các bộ mã hóa encoder Transformer. Mỗi bộ mã hóa bao gồm cơ chế tự chú ý đa đầu (multi-head self-attention) và mạng nơ-ron feed-forward. Các bộ mã hóa này nắm bắt các mối quan hệ theo ngữ cảnh trong chuỗi đầu vào.
- Mục tiêu Huấn luyện Ngầm (Pre-training Objective): XLM-RoBERTa được huấn luyện ngầm bằng các tác vụ không giám sát khác nhau, chẳng hạn như mô hình hóa ngôn ngữ che khuất (masked language modeling - tương tự như BERT) và mô hình hóa ngôn ngữ dịch thuật. Điều này giúp mô hình học được các biểu diễn ngôn ngữ tổng hợp và thông tin đa ngôn ngữ.
- Chia sẻ Kiến thức Đa Ngôn ngữ: Một tính năng chính của XLM-RoBERTa là khả năng chia sẻ kiến thức giữa nhiều ngôn ngữ trong quá

trình huấn luyện ngầm. Điều này cho phép mô hình tận dụng dữ liệu từ các ngôn ngữ khác nhau để cải thiện khả năng hiểu và hiệu suất trên các tác vụ liên quan đến ngôn ngữ.

- Mã Ngôn ngữ (Language Embeddings): XLM-RoBERTa sử dụng mã ngôn ngữ để mã hóa thông tin về ngôn ngữ đầu vào. Các mã này giúp mô hình phân biệt giữa các ngôn ngữ khác nhau và xử lý hiệu quả các đầu vào đa ngôn ngữ.
- Fine-tuning (Tinh chỉnh): Sau khi huấn luyện ngầm, XLM-RoBERTa có thể được tinh chỉnh cho các tác vụ hạ tầng cụ thể, chẳng hạn như phân loại văn bản, nhận dạng thực thể tên riêng hoặc dịch máy, bằng cách thêm các lớp dành riêng cho tác vụ trên đầu của mô hình đã được huấn luyện ngầm.

XLM-RoBERTa đã đạt được hiệu suất vượt trội trên nhiều tác vụ đa ngôn ngữ và đặc biệt hữu ích trong các tình huống cần hiểu đa ngôn ngữ và học chuyển giao.

4.5. RoBERTa

RoBERTa, do Facebook phát triển, là mô hình ngôn ngữ được huấn luyện lại từ BERT với phương pháp tiên tiến hơn và sử dụng gấp 10 lần dữ liệu.

Điểm khác biệt chính giữa RoBERTa và BERT nằm ở phương pháp huấn luyện:

- Kỹ thuật mặt nạ động (dynamic masking): Thay vì dự đoán câu kế tiếp (NSP) như BERT, RoBERTa sử dụng kỹ thuật mặt nạ động, trong đó các từ khóa (token) sẽ được thay đổi ngẫu nhiên trong quá trình huấn luyện.
- Kích thước batch lớn: Sử dụng kích thước batch lớn hơn trong quá trình huấn luyện RoBERTa cho thấy hiệu quả cao hơn.

Ngoài ra, RoBERTa được huấn luyện trên tập dữ liệu khổng lồ 160GB, bao gồm:

- 16GB sách và Wikipedia tiếng Anh (đã được sử dụng trong huấn luyện BERT)
- 63 triệu bản tin và 76GB dữ liệu từ CommonCrawl News dataset

- 38GB ngữ liệu văn bản Web
- 31GB dữ liệu từ Common Crawl Stories

Với sự hỗ trợ của 1024 GPU Tesla V100, RoBERTa chỉ mất 1 ngày để hoàn tất quá trình huấn luyện. Nhờ những cải tiến này, RoBERTa đạt hiệu quả vượt trội hơn cả BERT và XLNet trên tập đánh giá GLUE.

4.6 Độ đo Exact Match và F1

Trong nhiều bộ dữ liệu về vấn đề tự động, bao gồm cả SQuAD, có hai số liệu đánh giá chính là Đánh giá Chính xác tuyệt đối (Exact Match - EM) và Điểm F1. Các điểm số này được tính toán trên từng cặp câu hỏi-đáp án riêng lẻ.

Nếu một câu hỏi có nhiều đáp án đúng, điểm EM và F1 sẽ được tính dựa trên kết quả khớp nhất với một trong các đáp án đúng. Cuối cùng, điểm EM và F1 tổng thể của một mô hình được tính toán bằng cách lấy trung bình cộng điểm của từng ví dụ riêng lẻ.

Đánh giá Chính xác tuyệt đối (Exact Match - EM):

Đây là một cách đánh giá đơn giản. Đối với mỗi cặp câu hỏi-đáp án, nếu các ký tự trong dự đoán của mô hình hoàn toàn khớp với các ký tự của một trong các đáp án đúng, thì EM bằng 1, ngược lại EM bằng 0. Đây là một phép đo khắt khe theo kiểu tất cả-hoặc-không-gì; nếu sai lệch dù chỉ một ký tự thì điểm sẽ là 0. Khi đánh giá với một ví dụ phủ định (không có đáp án), nếu mô hình dự đoán bất kỳ văn bản nào, nó sẽ tự động nhận điểm 0 cho ví dụ đó.

Điểm F1:

Điểm F1 là một phép đo phổ biến cho các vấn đề phân loại, và được sử dụng rộng rãi trong lĩnh vực vấn đề tự động. Thích hợp khi chúng ta quan tâm ngang nhau đến độ chính xác (precision) và tính đầy đủ (recall). Trong trường hợp này, điểm F1 được tính toán trên từng từ riêng lẻ trong dự đoán so với các từ trong câu trả lời đúng. Số lượng từ trùng khớp giữa dự đoán và câu trả lời chính xác là cơ sở để tính điểm F1:

- Độ chính xác (Precision): Tỷ lệ giữa số từ trùng khớp với tổng số từ trong dự đoán của mô hình.
- Tính đầy đủ (Recall): Tỷ lệ giữa số từ trùng khớp với tổng số từ trong câu trả lời đúng.

4.7. Mô hình Question & Answering (QA Model)

4.7.1 Mô hình hỏi đáp - Question & Answering Model

4.7.1.1. Định nghĩa

Mô hình hỏi đáp (Question-Answering – QA) là một loại mô hình Xử lý Ngôn ngữ Tự nhiên, có khả năng trả lời câu hỏi được đặt ra bằng ngôn ngữ thông thường. Mô hình này được huấn luyện trên một kho dữ liệu văn bản khổng lồ. Từ đó, chúng sử dụng số lượng dữ liệu huấn luyện để học cách hiểu ý nghĩa câu hỏi và tìm ra câu trả lời tương ứng trong văn bản.

Tính chất của câu trả lời tạo ra từ các mô hình hỏi đáp phụ thuộc vào loại mô hình hỏi đáp được sử dụng: câu trả lời được lấy trực tiếp từ đoạn văn bản ngữ cảnh cung cấp hay câu trả lời được tạo mới hoàn toàn dựa trên tri thức sẵn có,...

4.7.1.2. Ứng dụng

Mô hình hỏi đáp được ứng dụng rộng khắp trong đời sống thông qua

- Các công cụ tìm kiếm
- Chatbot
- Trợ lý ảo

Nhờ có những ứng dụng này mà khả năng nắm bắt, tiếp thụ và tương tác với thông tin của người dùng được cải thiện hơn rất nhiều.

4.7.1.3. Phân loại

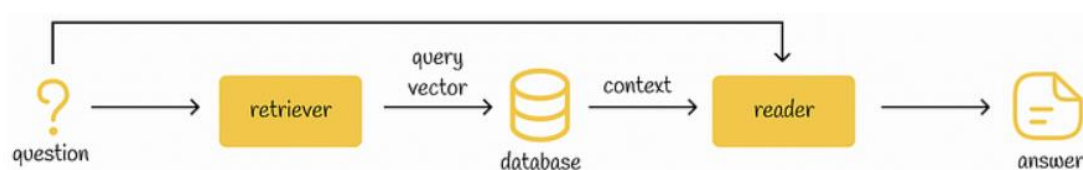
Tuỳ thuộc vào cách câu trả lời được tạo ra mà những mô hình hỏi đáp được phân loại thành:

- Mô hình hỏi đáp trích xuất (Extractive Question-Answering Model): Mô hình trích xuất câu trả lời từ ngữ cảnh đầu vào và cung cấp trực tiếp cho người dùng. Ví dụ điển hình của loại mô hình hỏi đáp này là mô hình BERT.
- Mô hình hỏi đáp tạo sinh (Generative Question-Answering Model): Mô hình sinh ra câu trả lời tự do dựa trên ngữ cảnh cung cấp (tri thức đã có).

Không những thế, các mô hình hỏi đáp còn khác nhau ở nguồn gốc câu trả lời:

- Mô hình hỏi đáp mở (Open Question-Answering Model): Câu trả lời được trích xuất trực tiếp từ ngữ cảnh cung cấp.
- Mô hình hỏi đáp đóng (Closed Question-Answering Model): Toàn bộ câu trả lời được tạo ra bởi mô hình mà không có ngữ cảnh cung cấp.

4.7.2. Extractive Question & Answering



Hình 4.3: Cơ chế hoạt động của mô hình hỏi đáp trích xuất

4.7.2.1 Tìm hiểu mô hình

Cấu trúc

Mô hình hỏi đáp trích xuất gồm ba thành phần chính

- Bộ truy xuất (Retriever)
- Cơ sở dữ liệu (Database)
- Bộ đọc (Reader)

Phân tích cơ chế hoạt động

Đầu tiên, câu hỏi được đưa vào bộ truy xuất. Mục tiêu của bộ truy xuất là trả về một vector nhúng (embedding vector) tương ứng với câu hỏi. Có nhiều cách triển khai bộ truy xuất, từ các phương pháp vector hóa đơn giản như TF-IDF đến các mô hình phức tạp hơn. Trong phần lớp trường hợp, các mô hình phức tạp như BERT được tích hợp vào bộ truy xuất. Khác với các phương pháp hình thành vector nhúng đơn giản chỉ dựa

vào tần suất, các phương pháp tạo vector nhúng trong bộ truy xuất của mô hình ngôn ngữ giúp tạo ra các vector có khả năng nắm bắt được ngữ nghĩa văn bản.

Sau khi có được vector nhúng tương ứng từ câu hỏi, vector này lại được sử dụng để tìm kiếm các vector có độ tương đồng cao nhất trong một tập hợp các tài liệu bên ngoài. Thông thường, tập hợp dữ liệu này được xử lý trong giai đoạn huấn luyện bằng cách được đưa qua bộ truy xuất, bộ truy xuất sẽ xuất ra các vector nhúng tương ứng cho các tài liệu nhận vào. Các vector nhúng này sau đó thường được lưu trữ trong cơ sở dữ liệu để có thể tìm kiếm hiệu quả trong giai đoạn so sánh với vector nhúng của câu hỏi.

Bằng cách truy xuất k vector từ cơ sở dữ liệu có độ tương đồng nhất với vector truy vấn, các tài liệu liên quan nhất đến vector truy vấn được lấy từ cơ sở dữ liệu chuyển vào bộ đọc. Bộ đọc xem xét từng tài liệu để tìm đoạn văn bản có chứa câu trả lời. Nó sử dụng các mô hình ngôn ngữ để xác định đoạn văn bản nào có khả năng cao chứa câu trả lời chính xác. Sau đó, bộ đọc gán một xác suất cho mỗi câu trả lời được trích xuất, dựa trên độ tin cậy của nó. Câu trả lời với xác suất cao nhất sẽ được bộ đọc trả về cho người dùng.

4.7.2.2 Đánh giá mô hình

Ưu điểm:

- Vì tính chất trích xuất của chúng, các mô hình này không cần phải lưu trữ thông tin thực tế, do đó không nhất thiết phải có nhiều tham số hơn. So với các mô hình tạo sinh, các mô hình trích xuất thường nhỏ hơn nhiều về kích thước và yêu cầu ít dữ liệu huấn luyện hơn.
- Kích thước tương đối nhỏ của chúng cũng làm cho việc chia sẻ các mô hình này trở nên dễ dàng hơn.
- Ứng dụng điển hình nhất của mô hình này là trong các trường hợp mà người dùng muốn nhận được câu trả lời nguyên văn từ một bộ sưu tập tài liệu lớn. Ví dụ, nó có thể được sử dụng để trích xuất câu trả lời từ các tài liệu kỹ thuật trong một khoảng thời gian ngắn.

Nhược điểm:

- Giới hạn ngữ cảnh: Hệ thống QA trích xuất chỉ có thể trả lời dựa trên ngữ cảnh có sẵn và không thể tạo ra câu trả lời nếu thông tin không có trong tài liệu.
- Không linh hoạt: Do chỉ trích xuất thông tin từ văn bản có sẵn, hệ thống này không thể cung cấp các câu trả lời sáng tạo hoặc diễn giải khác đi từ nội dung đã được cung cấp.
- Phụ thuộc vào chất lượng tài liệu: Nếu các tài liệu nguồn không chứa thông tin chính xác hoặc không rõ ràng, hệ thống sẽ không thể cung cấp câu trả lời đúng hoặc đủ chi tiết.

4.7.3. Closed Generative Question & Answering



Hình 4.4: Cơ chế hoạt động của mô hình hỏi đáp tạo sinh đóng
4.7.3.1 Tìm hiểu mô hình

Các mô hình QA tạo sinh đóng không có quyền truy cập vào bất kỳ thông tin bên ngoài nào và tạo ra câu trả lời chỉ bằng cách sử dụng thông tin từ câu hỏi.

Phần lớn thời gian, mô hình tạo sinh đóng được sử dụng trong các ứng dụng với các câu hỏi mang nét nghĩa chung. Đối với các lĩnh vực rất cụ thể, hiệu suất của các mô hình tạo sinh đóng có xu hướng giảm.

4.7.3.2 Đánh giá mô hình

Ưu điểm:

- Thời gian xử lý nhanh hơn: Hệ thống QA tạo sinh đóng không cần phải tìm kiếm qua một bộ sưu tập tài liệu bên ngoài lớn, do đó giảm thời gian xử lý.

Nhược điểm:

- Chi phí huấn luyện cao và độ chính xác phụ thuộc vào kiến thức huấn luyện: Bộ tạo sinh phải rất mạnh mẽ và có lượng kiến thức huấn luyện lớn để tạo ra các câu trả lời chính xác. Việc huấn luyện các mô hình này tốn nhiều tài nguyên.

- Không cập nhật thông tin mới: Bộ tạo sinh không biết thông tin xuất hiện sau thời điểm nó được huấn luyện. Để cập nhật thông tin mới, mô hình phải được huấn luyện lại trên dữ liệu mới, điều này rất tốn tài nguyên vì các mô hình này thường có hàng triệu hoặc hàng tỷ tham số.
- Khó khăn trong việc cập nhật thông tin so với các hệ thống khác: So với hệ thống QA trích xuất và QA tạo sinh mở, việc cập nhật thông tin mới trong hệ thống QA tạo sinh đóng phức tạp hơn nhiều.

CHƯƠNG 5: GIỚI THIỆU BỘ DỮ LIỆU

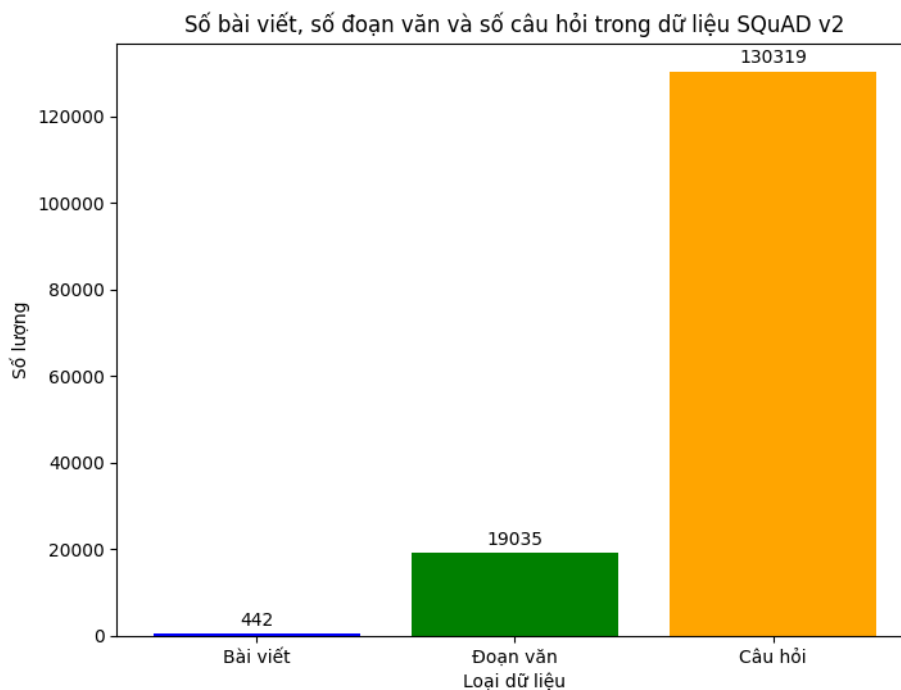
5.1. Giới thiệu bộ dữ liệu

Để đào tạo và xây dựng hệ thống trả lời câu hỏi của mình, chúng tôi đã sử dụng bộ dữ liệu “Squad v2” được công nhận rộng rãi, do Đại học Stanford phát triển đặc biệt cho các nhiệm vụ xử lý ngôn ngữ tự nhiên (NLP). Đóng vai trò là phiên bản nâng cao của "Squad v1", tập dữ liệu này bao gồm hơn 100.000 phiên bản, mỗi phiên bản bao gồm một đoạn văn và một câu hỏi tương ứng. Tập dữ liệu "Squad v2" được phân chia thành hai tập con: training, bao gồm 130.319 cặp câu hỏi-câu trả lời và validation, với 11.873 cặp. Thuộc tính quan trọng của tập dữ liệu là sự phân bố công bằng các câu hỏi có thể trả lời và không thể trả lời trong cả hai bộ, đảm bảo phương pháp đánh giá cân bằng. Được lưu trữ ở định dạng JSON, mỗi tệp tập dữ liệu bao gồm một mảng các mục dữ liệu. Các mục này bao gồm tiêu đề và một tập hợp các đoạn văn, trong đó mỗi đoạn chứa văn bản và danh sách các câu hỏi liên quan.

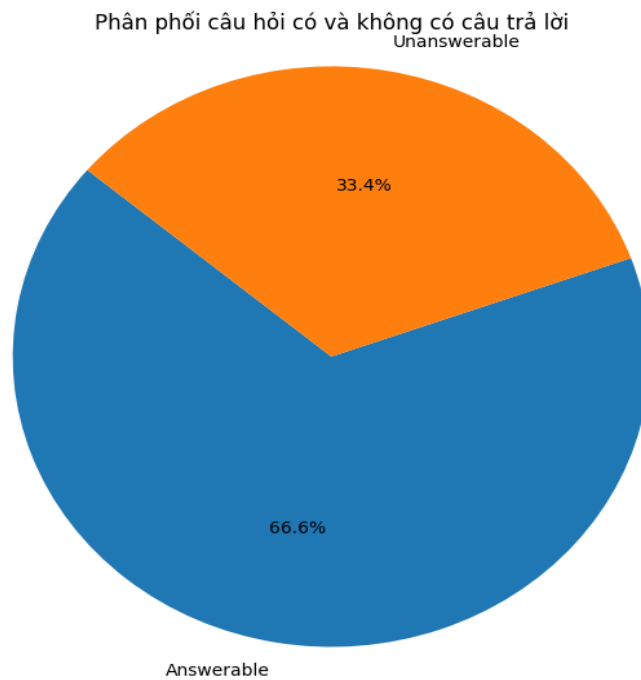
Mỗi câu hỏi được xác định duy nhất bằng ID, văn bản câu hỏi, nhãn khả năng trả lời và một hoặc nhiều câu trả lời, mỗi câu trả lời được chỉ định theo vị trí văn bản và ký tự. Để định lượng độ phức tạp về ngôn ngữ, văn bản trong tập dữ liệu "Squad v2" trải qua quá trình encoding, tạo ra tổng cộng 1.535.809 token. Tập training bao gồm 1.321.104 token, trong khi tập validation bao gồm 214.705 token. Đáng chú ý, phương pháp encoding phân đoạn văn bản dựa trên ranh giới từ và dấu chấm câu, loại bỏ khoảng trắng. Một tính năng đặc biệt của “Squad v2” là sự kết hợp của các trường hợp “không thể”, được thiết kế một cách chiến lược để nâng cao khả năng đo lường độ chính xác bằng cách đưa ra các tình huống đầy thách thức. Những trường hợp này bao gồm các tình huống trong đó câu trả lời không có trong văn bản hoặc khi câu hỏi chứa thông tin sai lệch hoặc không đầy đủ. Phạm vi bao phủ toàn diện của tập dữ liệu về các chủ đề đa dạng và ý nghĩa trong thế giới thực góp phần cải thiện khả năng khái quát hóa mô hình.

5.2. Phân tích dữ liệu khám phá (EDA)

Để hiểu rõ hơn về bộ dữ liệu SQuAD v2 và chuẩn bị cho việc xây dựng hệ thống hỏi đáp sử dụng BERT và FAISS, nhóm đã tiến hành phân tích dữ liệu khám phá (Exploratory Data Analysis - EDA).

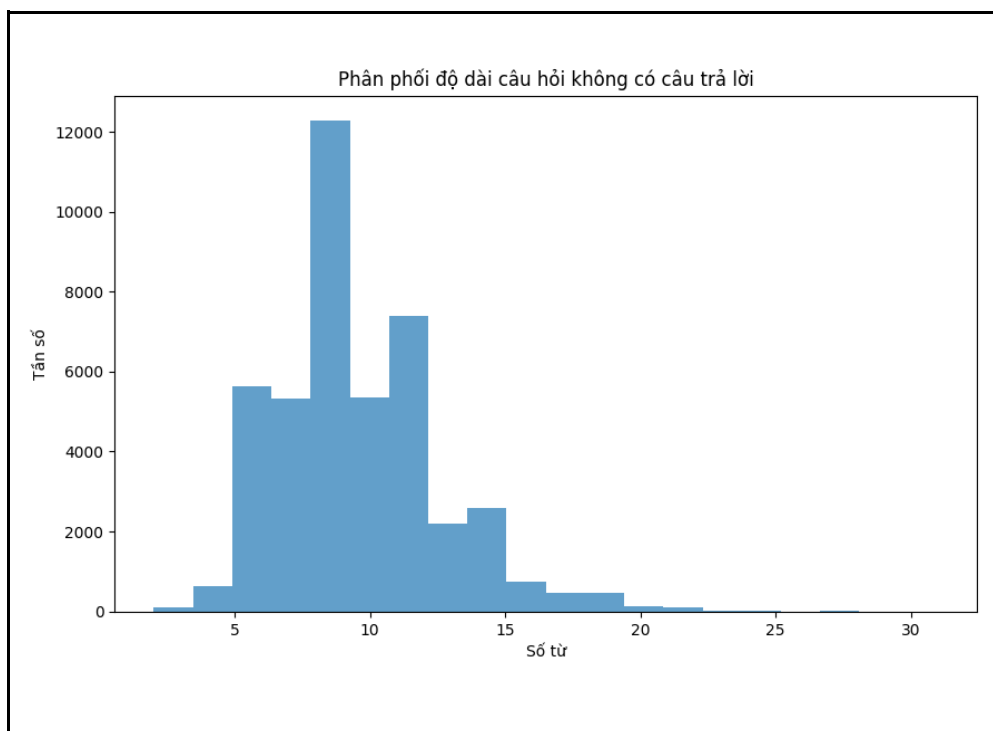


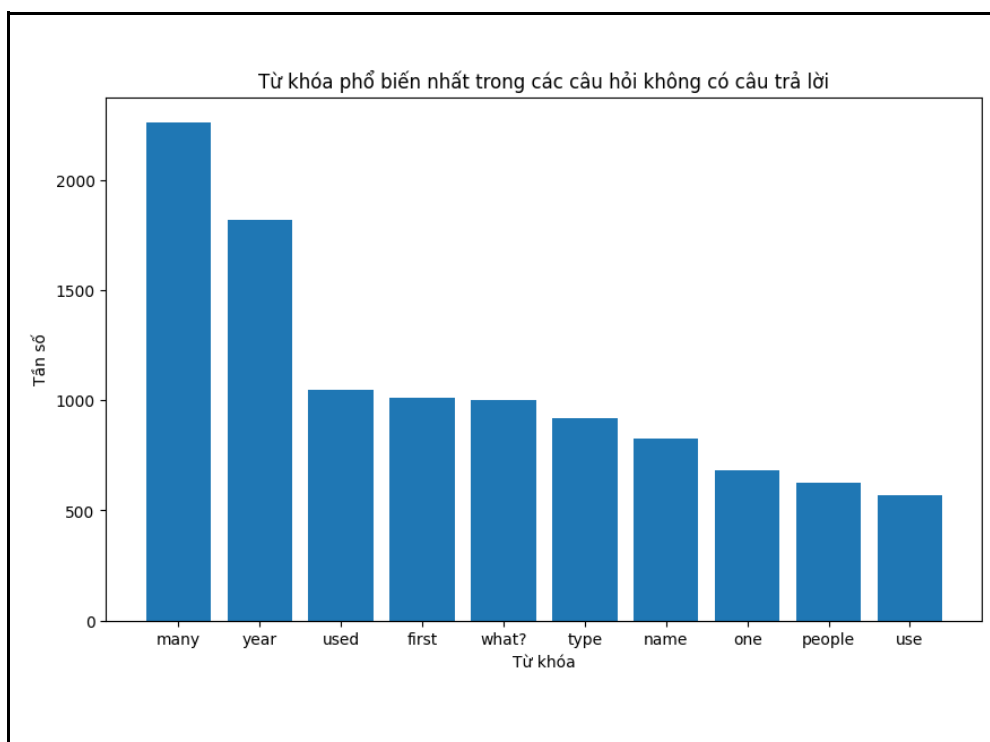
Hình 5.1: Số lượng bài viết, đoạn văn và câu hỏi trong bộ dữ liệu SQuADv2



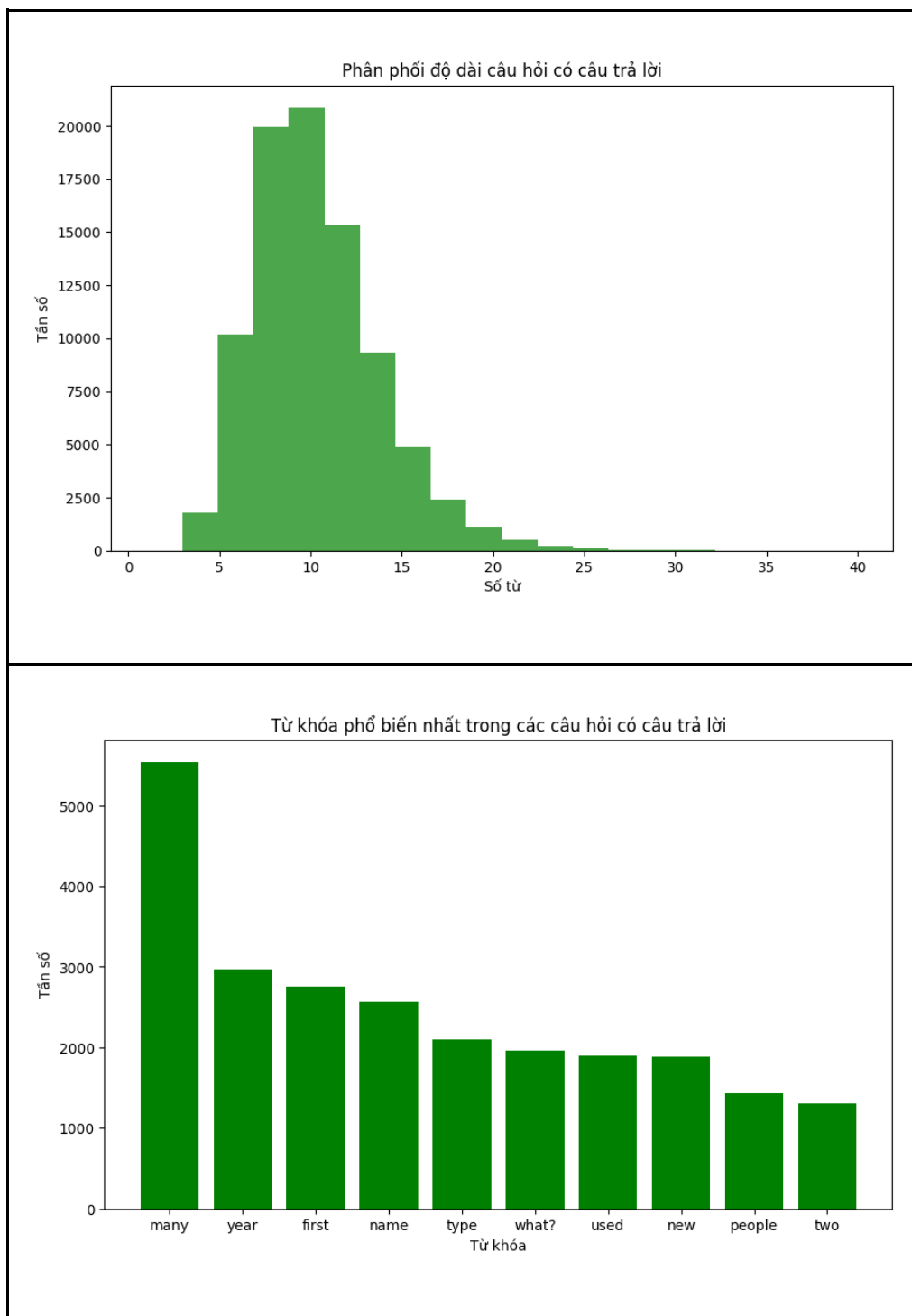
Hình 5.2: Phân phối câu hỏi có câu trả lời và câu hỏi không có câu trả lời

Bộ dữ liệu SQuAD v2 bao gồm tổng cộng 442 bài viết, với tổng số đoạn văn là 19,035 và tổng số câu hỏi là 130,319. Trong số đó, có 86,821 câu hỏi có câu trả lời và 43,498 câu hỏi không có câu trả lời.





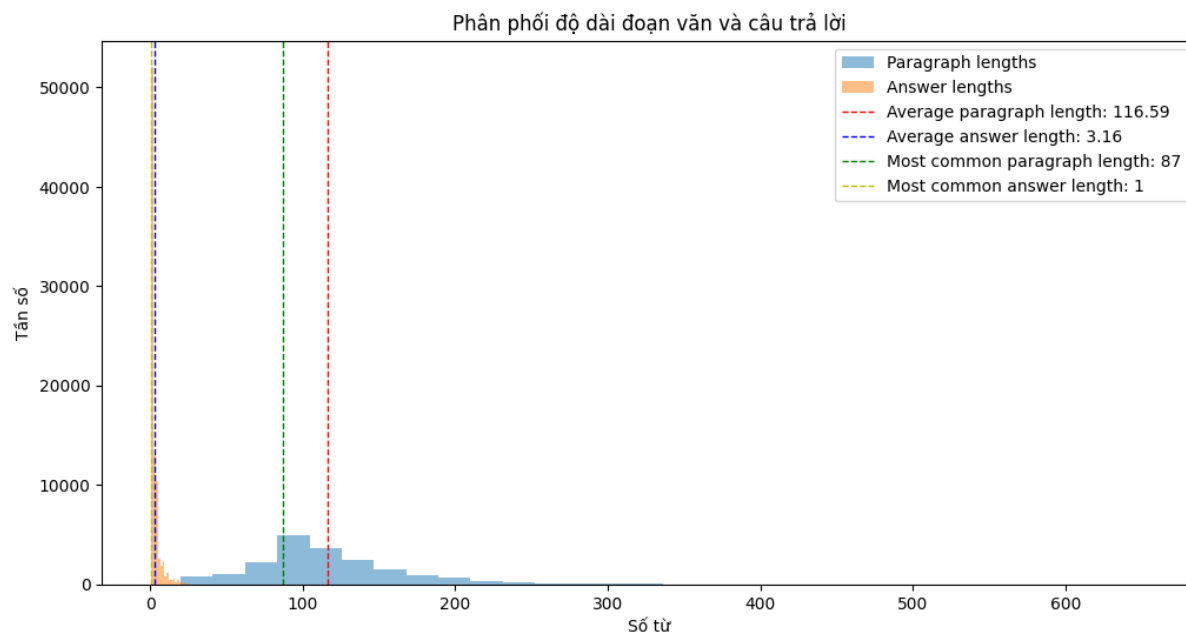
Hình 5.3: Phân tích câu hỏi không có câu trả lời



Hình 5.4: Phân tích câu hỏi có câu trả lời

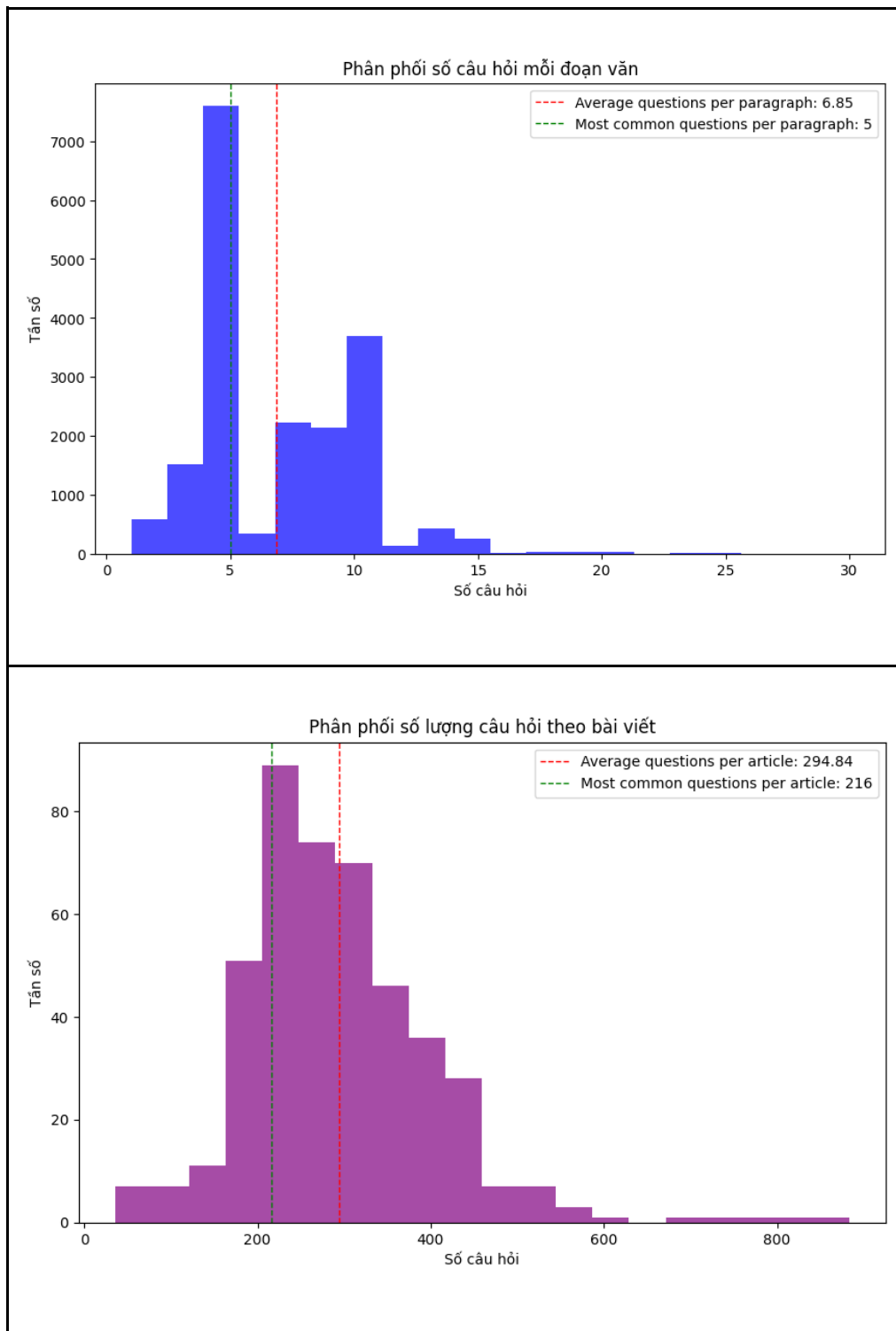
Phân tích chi tiết về độ dài câu hỏi trên Hình 5.3 và Hình 5.4 cho thấy rằng độ dài trung bình của các câu hỏi không có câu trả lời là khoảng 9 từ, trong khi độ dài trung bình của các câu hỏi có câu trả lời là khoảng 10 từ. Các từ khóa phổ biến nhất trong các câu hỏi không có câu trả lời bao gồm 'many', 'year', 'used', 'first', và 'what?', trong khi trong các câu hỏi có câu trả lời, các từ khóa phổ biến nhất bao gồm 'many', 'year', 'first', 'name',

và 'type'. Sau khi phân tích thì nhóm cảm thấy không có sự khác biệt nhiều ở hai nhóm câu hỏi, có lẽ sự khác biệt nằm ở cách câu hỏi được hỏi.



Hình 5.5: Phân phối độ dài đoạn văn và câu trả lời

Khi phân tích độ dài của đoạn văn và câu trả lời, ta thấy rằng độ dài của câu trả lời thường nằm trong khoảng từ 1 đến 20 từ, với độ dài trung bình là 3 từ. Đối với đoạn văn, độ dài thường nằm trong khoảng từ 10 đến 350 từ, với độ dài trung bình là 116 từ.



Hình 5.6: Phân phối số câu hỏi mỗi đoạn văn và số câu hỏi mỗi bài viết

Phân tích về số câu hỏi của mỗi đoạn văn ở Hình 5.6 cho thấy rằng số câu hỏi thường nằm trong khoảng từ 1 đến 26 câu, với số câu hỏi trung bình là khoảng 7 câu và số câu hỏi phổ biến nhất của mỗi đoạn văn là 5 câu. Tương tự, phân tích về số câu hỏi theo bài

viết cũng cho kết quả tương tự với số câu hỏi trung bình là khoảng 295 câu và số câu hỏi phổ biến nhất của mỗi bài viết là 216 câu.

Các phân tích này giúp nhóm hiểu rõ hơn về cấu trúc và tính chất của bộ dữ liệu SQuAD v2, từ đó có thể tối ưu hóa quá trình xây dựng hệ thống hỏi đáp sử dụng các mô hình như BERT và FAISS.

CHƯƠNG 6: XÂY DỰNG MÔ HÌNH & GIAO DIỆN

Trước khi sử dụng mô hình trong ứng dụng, nhóm cần phải lựa chọn mô hình có hiệu quả cao nhất dựa trên các tiêu chí đã chọn.

6.1. Thiết lập thực nghiệm

Fine-tuning là một quy trình trong học máy, trong đó ta cần tinh chỉnh một mô hình đã được huấn luyện sẵn (pre-trained) trên một bộ dữ liệu cụ thể. Mục tiêu của quá trình này là để điều chỉnh để đảm bảo mô hình cuối cùng có thể đáp ứng được nhu cầu của người dùng tốt hơn, trong khi các chức năng chính vẫn không thay đổi.

Trong dự án này, nhóm sẽ thực hiện huấn luyện 3 mô hình khác nhau đã đề cập từ phía trên bao gồm DistilBERT - base, RoBERTa - base, và XLM- RoBERTa - base và thực hiện so sánh kết quả của cả ba mô hình này trên bộ dữ liệu SQuADv2. Chúng ta sẽ tập trung chủ yếu vào 2 độ đo là Exact Match và F1.

Bộ dữ liệu SQuADv2 sẽ được chia làm ba tập training, validation và test với kích thước lần lượt là 130,319, 11,873 và 8,862.

Toàn bộ cả ba mô hình sẽ được thiết lập với cùng các tham số như nhau và huấn luyện trên cùng một bộ dữ liệu training và validation được sắp xếp giống nhau. Các tham số mà nhóm chúng tôi cài đặt để huấn luyện cho cả ba mô hình bao gồm gia tốc học (learning rate) là $2e-5$, kích thước batch cho phần huấn luyện (training batch size) là 16, kích thước cho việc đánh giá qua mỗi epoch (validation batch size) là 8, hệ số điều chỉnh giảm trọng số (weight decay) là 0.01, và cuối cùng là số lượng epoch huấn luyện là 2.

6.2. Đánh giá kết quả mô hình.

Models	Total Parameters	Metrics			
		EM(%)	F1(%)	HasAns Exact(%)	HasAns F1(%)
DistilBERT	66M	43.59	47.72	74.19	82.46
RoBERTa	124M	64.99	68.19	84.16	90.59
XLM-RoBERa	277M	65.78	69.13	80.99	87.70

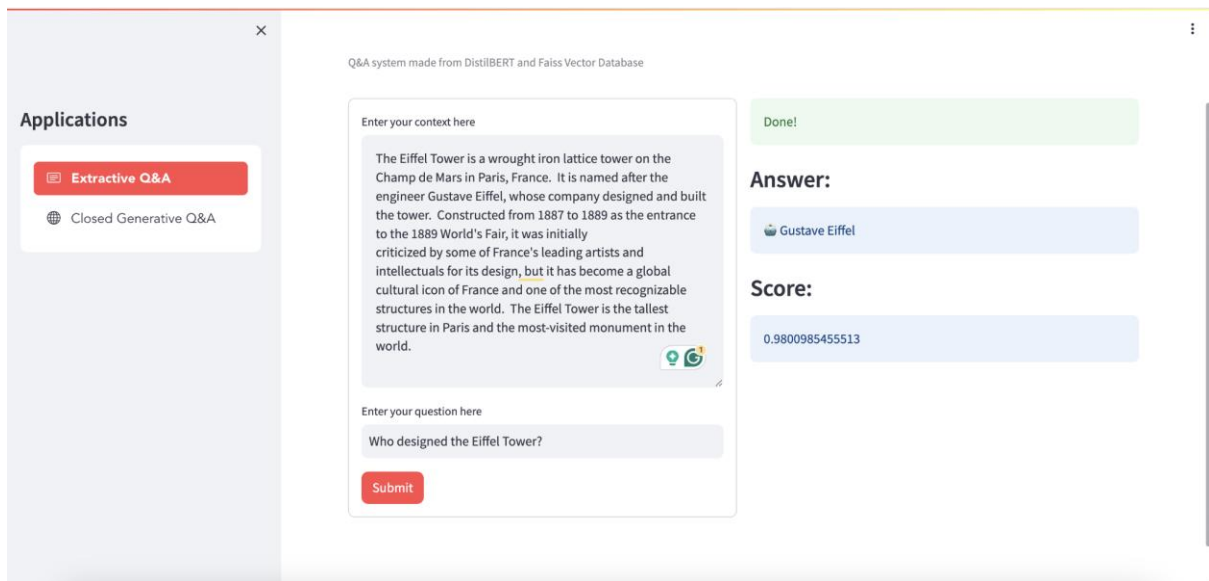
Dựa trên các kết quả thu được, nhóm nhận thấy rằng mô hình XLM-RoBERTa đạt hiệu suất cao nhất trong việc trả lời câu hỏi trên bộ dữ liệu SQuADv2, với chỉ số Exact Match (EM) là 65.78% và chỉ số F1 là 69.13%. Điều này cho thấy XLM-RoBERTa không chỉ chính xác trong việc xác định câu trả lời đúng mà còn thể hiện khả năng vượt trội trong việc nhận diện và trả lời các câu hỏi có câu trả lời (HasAns). Mặc dù vậy, mô hình này có số lượng tham số nhiều nhất (277 triệu tham số), yêu cầu tài nguyên tính toán lớn. RoBERTa, với 124 triệu tham số, đạt hiệu suất rất gần với XLM-RoBERTa, với EM là 64.99% và F1 là 68.19%, và có thể là lựa chọn tốt nếu cân cân bằng giữa hiệu suất và tài nguyên tính toán. Trong khi đó, DistilBERT, với số lượng tham số ít nhất (66 triệu tham số), cho thấy hiệu suất thấp hơn đáng kể với EM là 43.59% và F1 là 47.72%, nhưng lại có ưu điểm về tốc độ và yêu cầu tài nguyên thấp, phù hợp cho các ứng dụng yêu cầu thời gian phản hồi nhanh và hạn chế về tài nguyên. Tổng kết lại, XLM-RoBERTa là mô hình có hiệu suất tốt nhất, tiếp theo là RoBERTa và DistilBERT, nên nhóm đi đến kết luận sẽ áp dụng mô hình này cho việc xây dựng mô hình Q&A System cuối cùng.

6.3. Xây dựng hệ thống QA System

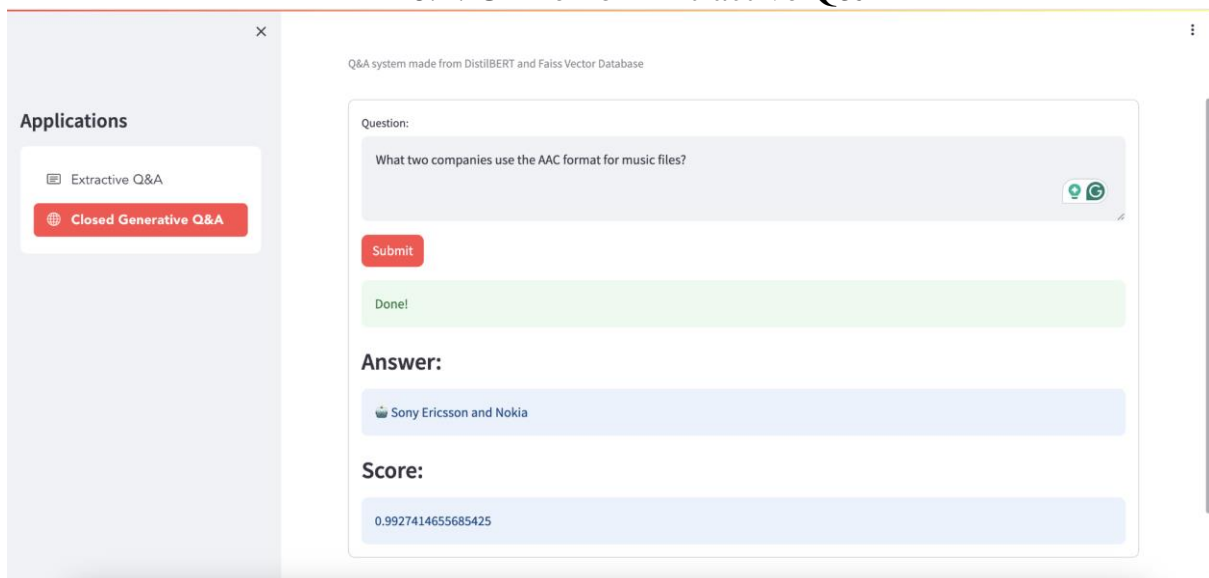
6.3.1. Giao diện

Giao diện gồm có hai phần chính: Extractive Q&A và Closed Generative Q&A. Sau khi nhập nội dung cần thiết, người dùng có thể gửi và sẽ nhận được câu trả lời

(answer) cùng chỉ số đánh giá (score).



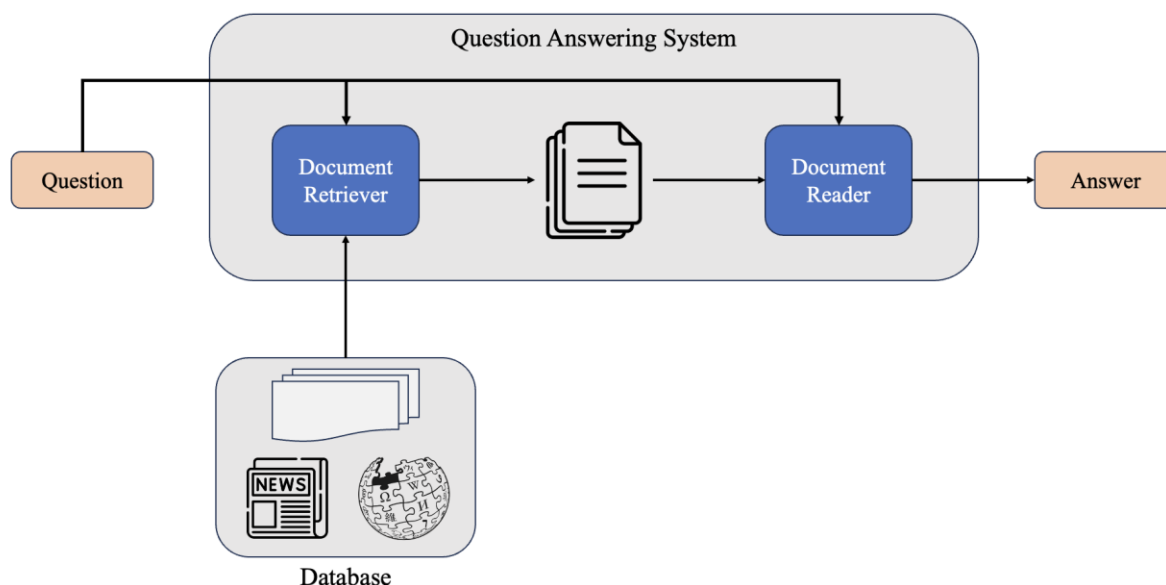
Hình 6.1: UI Demo - Extractive Q&A



Hình 6.2: UI Demo - Closed Generative Q&A

6.3.2. Extractive Q&A

Đây là dạng hệ thống Q&A mà trong đó người dùng phải nhập vào nội dung bối cảnh (context) cho câu hỏi. Đây là các nội dung liên quan tới câu hỏi và có chứa câu trả lời. Với hệ thống này, mô hình XLM-RoBERTa mà nhóm đã huấn luyện từ trước sẽ được tái sử dụng cho việc dự đoán câu trả lời dựa trên context đó. Quan sát hình ảnh cho pipeline mô hình Extractive Q&A bên dưới.



Hình 6.3: Pipeline cho mô hình Extractive Q&A

Hệ thống Extractive QA cần hai module chính là Retriever và Reader. Với các câu hỏi được nhập vào module Retriever sẽ dựa vào đó mà xác định các nội dung mà nó đã từng học. Từ đó, module Reader sẽ có thể hiểu được và nó sẽ rút trích (extract) được câu trả lời từ context cho sẵn.

Enter your context here

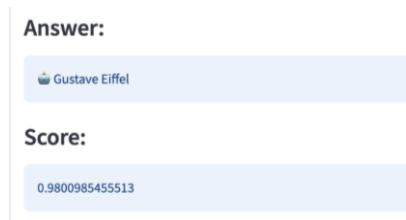
The Eiffel Tower is a wrought iron lattice tower on the Champ de Mars in Paris, France. It is named after the engineer Gustave Eiffel, whose company designed and built the tower. Constructed from 1887 to 1889 as the entrance to the 1889 World's Fair, it was initially criticized by some of France's leading artists and intellectuals for its design, but it has become a global cultural icon of France and one of the most recognizable structures in the world. The Eiffel Tower is the tallest structure in Paris and the most-visited monument in the world.

Enter your question here

Who designed the Eiffel Tower?

Submit

Hình 6.4: Ví dụ về context và câu hỏi



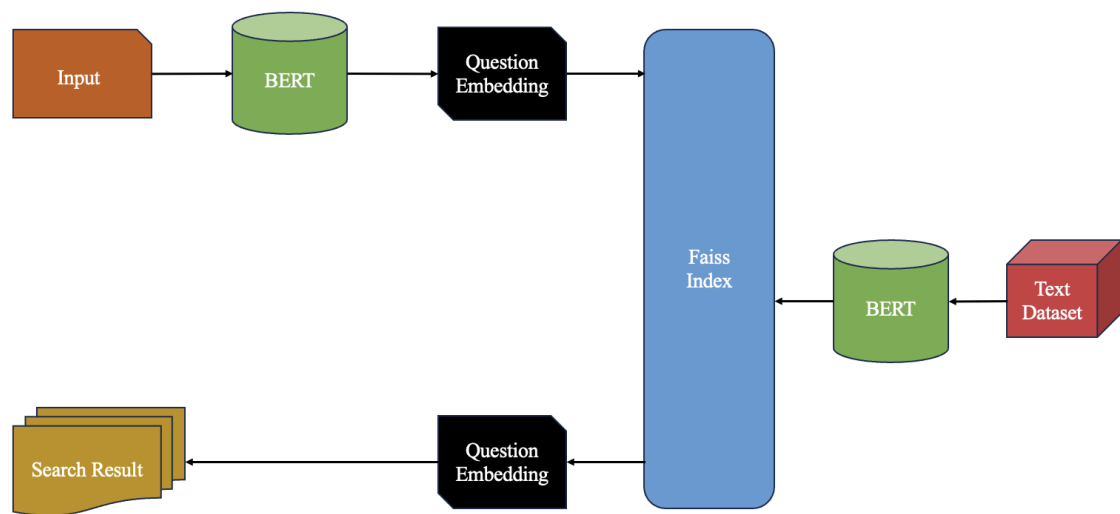
Hình 6.5: Ví dụ về câu trả lời và chỉ số

Có thể thấy, với ví dụ ở trên, hệ thống đã trả về câu trả lời chính xác với score rất cao (0.98).

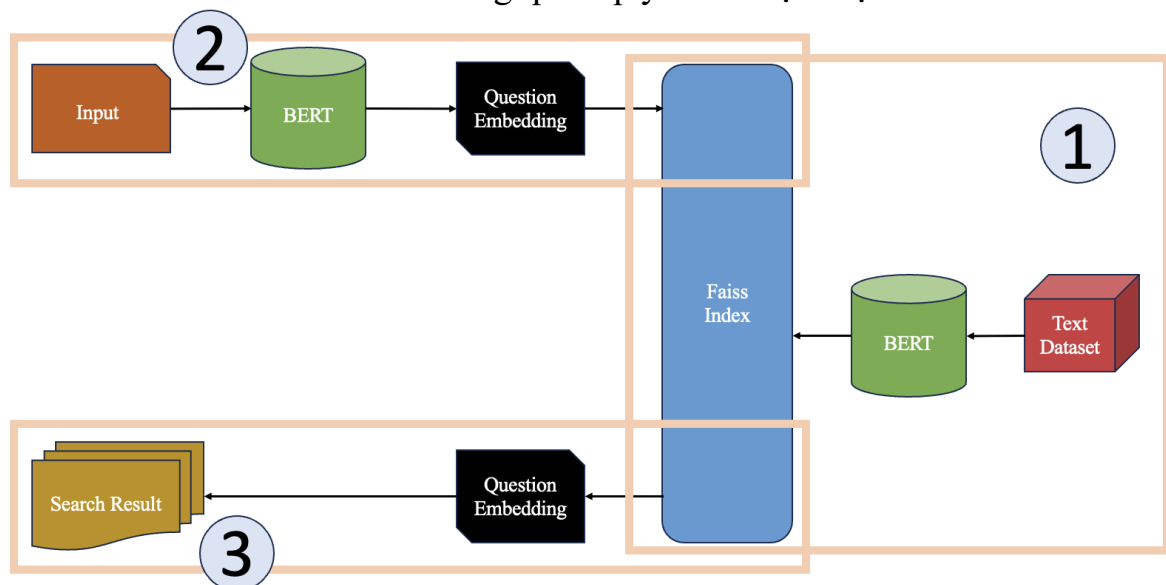
6.3.3. Closed Generative Q&A

Khác với Extractive Q&A, hệ thống Closed Generative Q&A không yêu cầu người dùng nhập vào context, chỉ cần nhập vào câu hỏi. Nhưng trong đề án này, nhóm sẽ cải biên lại hệ thống Closed Generative Q&A bằng việc áp dụng hệ thống cơ sở dữ liệu vector Faiss. Đầu tiên ta sẽ sử dụng mô hình XLM-RoBERTa để tạo ra các embedding của câu hỏi có sẵn trong bộ dữ liệu SQuADv2, kèm theo đó là context có sẵn của chính câu hỏi đó. Sau đó lưu nó vào một hệ cơ sở dữ liệu vector Faiss. Mỗi khi người dùng đặt một câu hỏi, XLM-RoBERTa sẽ encoding lại câu hỏi đó và dựa vào Faiss để tìm được các câu hỏi có nghĩa tương ứng với câu hỏi của người dùng. Khi đã tìm được, bước cuối cùng ta chỉ việc tái sử dụng mô hình XLM-RoBERTa đã qua huấn luyện phía trên và tạo ra câu trả lời giống với hệ thống Extractive Q&A.

Sau đây là sơ đồ mô tả các bước thực hiện của mô hình Closed Generative QA:



Hình 6.6: Tổng quan quy trình thực hiện




Hình 6.7: Tổng quan quy trình thực hiện - các bước

- Bước 1: Là giai đoạn lưu trữ các index cần thiết cho bộ dữ liệu có sẵn. Từ bộ dữ liệu có sẵn (SQUADv2), mô hình XLM-RoBERTa sẽ khởi tạo các embedding cần thiết, sau đó các embedding này được đánh chỉ mục (index) để tìm kiếm tương đồng hiệu quả hơn.

- Bước 2: Tại ứng dụng, người dùng sẽ nhập vào nội dung cần trả lời. Mô hình XLM-RoBERTa sẽ trả về các embedding cho dữ liệu đầu vào này và tìm chỉ mục Faiss gần nhất.
- Bước 3: Sau khi có được chỉ mục, tìm câu hỏi và trả về câu trả lời tương ứng cho câu hỏi trên.

Question:


What two companies use the AAC format for music files?



Submit

Done!

Answer:

 Sony Ericsson and Nokia

Score:

0.9927414655685425

Hình 6.8: Ví dụ về câu hỏi, câu trả lời và chỉ số

Dựa trên ví dụ, có thể thấy, kể cả khi không được cung cấp context mô hình vẫn có thể trả về được kết quả chính xác, với chỉ số cao.

CHƯƠNG 7: KẾT LUẬN

Qua bài nghiên cứu, nhóm đã thành công tìm hiểu về khái niệm của dữ liệu lớn và ứng dụng cụ thể tại hai doanh nghiệp đã chọn là Facebook và Spotify. Các công nghệ sử dụng tại các doanh nghiệp này được nhóm mô tả chi tiết cũng như minh họa lại để cho thấy hiệu quả của chúng.

Thêm vào đó, nhóm đã thành công xây dựng một hệ thống hỏi đáp bằng một mô hình có kết quả tốt. Bằng các so sánh kết quả của các biến thể của mô hình BERT, bao gồm DistilBERT - base, RoBERTa - base, và XLM- RoBERTa - base trên bộ dữ liệu SQuADv2, nhóm đã chọn ra mô hình tốt nhất để xây dựng hệ thống cuối cùng.

Dựa trên các số liệu hiệu suất của mô hình XLM-RoBERTa, ta thấy tỷ lệ EM (Exact Match) đạt khoảng 65.78%, có nghĩa là mô hình đưa ra câu trả lời chính xác cho khoảng hơn một nửa số câu hỏi được đặt ra. Đối với chỉ số F1, mô hình đạt 69.13%, biểu thị mức độ kết hợp giữa độ chính xác và độ phủ của câu trả lời so với câu trả lời thực tế. Điều này cho thấy mô hình có khả năng hiểu và phản hồi với mức độ chính xác khá tốt.

Khi tập trung vào các câu hỏi có câu trả lời có sẵn, tỷ lệ chính xác tăng lên đáng kể. Với chỉ số Exact Match đạt 80.99% và F1 score đạt 87.70%, mô hình XLM-RoBERTa có khả năng tìm ra câu trả lời đúng cho hầu hết các câu hỏi có sẵn.

Bằng cách kết hợp với hệ thống Faiss, hệ thống QA trở nên mạnh mẽ hơn. Faiss, với khả năng tối ưu hóa việc truy xuất thông tin từ cơ sở dữ liệu lớn, giúp tăng cường hiệu suất của hệ thống. Nó cung cấp cơ chế tìm kiếm nhanh chóng và hiệu quả, giúp hệ thống trả lời câu hỏi một cách linh hoạt và nhanh chóng hơn. Sử dụng Faiss trong quá trình này giúp tăng tốc độ và hiệu suất của hệ thống, đặc biệt là khi xử lý các cơ sở dữ liệu lớn. Nó giúp hệ thống có thể tìm kiếm và đưa ra câu trả lời một cách nhanh chóng và chính xác, tạo ra một trải nghiệm người dùng tốt hơn.

Kết hợp mô hình ngôn ngữ như XLM-RoBERTa với công cụ như Faiss không chỉ cải thiện hiệu suất của hệ thống QA mà còn tạo ra một hệ thống mạnh mẽ, linh hoạt và đáng tin cậy, cung cấp thông tin chính xác và đáng tin cậy cho người dùng.

Có thể nói, bài nghiên cứu của nhóm đã hoàn thành các mục tiêu đề ra ban đầu.

Tiếp đến, nhóm định hướng sẽ mở rộng chủ đề nghiên cứu bằng cách so sánh thêm mô hình, lựa chọn bộ dữ liệu có kích thước lớn hơn và ứng dụng các kỹ thuật xử lý phân tán cho hệ thống này.

TÀI LIỆU THAM KHẢO

1. Acceldata Product Team. (2022, May 6). Data Engineering: How Spotify Upgraded its Data Orchestration Platform. Acceldata.
<https://www.acceldata.io/blog/data-engineering-best-practices-how-spotify>
2. Efimov, V. (2024, February 28). Question-Answering Systems: Overview of Main Architectures. Medium. <https://towardsdatascience.com/question-answering-systems-overview-of-main-architectures-46b94d58bae6>
3. Heitman, S. (2022, May 5). What Happens in an Internet Minute? [2022 Statistics]. LOCALiQ. <https://localiq.com/blog/what-happens-in-an-internet-minute/>
4. Jain, H. (2021). BERT for “Everyone” (Tutorial + Implementation). Kaggle. <https://www.kaggle.com/code/harshjain123/bert-for-everyone-tutorial-implementation>
5. Li, N. (2017, October 16). Big Data Processing at Spotify: The Road to Scio (Part 1). Spotify Engineering. <https://engineering.atspotify.com/2017/10/big-data-processing-at-spotify-the-road-to-scio-part-1/>
6. Marr, B. (2021, July 2). How Does Big Data Help Companies? Bernard Marr. <https://bernardmarr.com/how-does-big-data-help-companies/>
7. Mishra, K., & Brown, M. (2015, January 9). Personalization at Spotify Using Cassandra. Spotify Engineering. <https://engineering.atspotify.com/2015/01/personalization-at-spotify-using-cassandra/>
8. pawangfg. (2020, April 30). Explanation of BERT Model - NLP. GeeksforGeeks. <https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/>
9. ProjectPro. (2024, April 9). BERT NLP Model Explained for Complete Beginners. ProjectPro. <https://www.projectpro.io/article/bert-nlp-model-explained/558>
10. Rathore, P. (2023, July 25). Q&A Model. Medium. <https://iprathore71.medium.com/q-a-model-76957da40e07>

11. Remotebase. (2024). BERT Explained: Revolutionizing Natural Language Processing. Remotebase.com. <https://remotebase.com/website/glossary/BERT>
12. Shaikh, R. (2023, August 29). Mastering BERT: A Comprehensive Guide from Beginner to Advanced in Natural Language Processing. Medium. <https://medium.com/@shaikhrayyan123/a-comprehensive-guide-to-understanding-bert-from-beginners-to-advanced-2379699e2b51>
13. Statista. (2023, August 22). Data Created Worldwide 2010-2025 . Statista; Statista. <https://www.statista.com/statistics/871513/worldwide-data-created/>
14. trituenhantao.io. (2020, April 29). BERT, RoBERTa, DistilBERT, XLNet - Chọn cái nào? Trí Tuệ Nhân Tạo. <https://trituenhantao.io/kien-thuc/bert-roberta-distilbert-xlnet-chon-cai-nao/>
15. Twentyman, J. (2015, April 16). Hadoop is the right track for Spotify. Diginomica. <https://diginomica.com/hadoop-is-the-right-track-for-spotify>
16. Yadav, S. (2023, July 13). What is BERT? How it is trained ? A High Level Overview. Medium. https://medium.com/@Suraj_Yadav/what-is-bert-how-it-is-trained-a-high-level-
17. Yucel, B. (2023, May 22). Generative vs. Extractive Language Models. Haystack. <https://haystack.deepset.ai/blog/generative-vs-extractive-models>
18. Alexis Conneau et al. (2020, 8 April). Unsupervised Cross-lingual Representation Learning at Scale.
19. Jacob Devlin et al. (2019, 24 May). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
20. Pranav Rajpurkar et al. (2016, 11 Oct). SQuAD: 100,000+ Questions for Machine Comprehension of Text.
21. Yinhan Liu et al. (2019, 26 July). RoBERTa: A Robustly Optimized BERT Pretraining Approach.
22. Victor SANH et al. (2020, 1 Mar). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter
23. Chen, G. J., Wiener, J., Iyer, S., Jaiswal, A., Lei, R., Simha, N., Wang, W., Wilfong, K., Williamson, T., & Yilmaz, S. (2016, June 25). Realtime Data

- Processing at Facebook. <https://research.facebook.com/publications/realtime-data-processing-at-facebook/>
24. Meta. (2024). Company Info | About Facebook. Meta.
<https://about.fb.com/company-info/>
25. Ryan, A. (2012, June 13). Under the Hood: Hadoop Distributed Filesystem reliability with Namenode and Avatarnode. Engineering at Meta.
<https://engineering.fb.com/2012/06/13/core-infra/under-the-hood-hadoop-distributed-filesystem-reliability-with-namenode-and-avatarnode/>
26. Nguyễn Mạnh, T. (2024). Slide bài giảng học phần Dữ liệu lớn và Ứng dụng - Chương 2: Hadoop: Distributed Architecture, HDFS, and MapReduce .
27. Mohammad Shahid Husain, Mohammad Zunnun Khan, & Siddiqui, T. (2023). Big Data Concepts, Technologies, and Applications (1st ed.). CRC Press.
28. Abraham, L., Allen, J., Barykin, O., Borkar, V., Chopra, B., Gere, C., Merl, D., Metzler, J., Reiss, D., Subramanian, S., Wiener, J., & Zed, O. (2013, August 27). Scuba: Diving into Data at Facebook. International Conference on Very Large Data Bases (VLDB).
<https://research.facebook.com/publications/scuba-diving-into-data-at-facebook/>
29. Baer, J. (2015, June 4). The Evolution of Big Data at Spotify. SlideShare.
<https://www.slideshare.net/slideshow/the-evolution-of-big-data-at-spotify/48996682>

PHỤ LỤC

[Đường dẫn đến slides; demo công nghệ tại chương 2,3; EDA chương 5; và demo hệ thống hỏi đáp](#)