HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

ĐỒ ÁN
TỐT NGHIỆP ĐẠI HỌC

ĐỀ TÀI:

TÌM HIỂU VÀ ỨNG DỤNG HỌC MÁY TRONG PHÂN TÍCH TRẠNG THÁI ĐƠN HÀNG CỦA CHUỖI LOGISTICS QUỐC TẾ

Giảng viên hướng dẫn : **ThS. NGUYỄN VĂN TIẾN**

Sinh viên thực hiện : **VŨ CHIẾN THẮNG**

Lớp : **E17CN01**

Khóa : **2017 - 2022**

Hệ : **ĐẠI HỌC CHÍNH QUY**

Hà Nội, tháng 01 năm 2022

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20…

Giảng viên hướng dẫn

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20…

GIẢNG VIÊN PHẢN BIỆN

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới thầy ThS. Nguyễn Văn Tiến – người luôn tận tình hướng dẫn em dù cho em có rất nhiều thiếu sót trong quá trình học tập nói chung cũng như quá trình thực hiện đồ án này nói riêng. Việc được thầy hướng dẫn chỉ bảo đã giúp em vượt qua khó khăn trong quá trình thực hiện đề tài và hoàn thiện với kết quả như mong muốn, ngoài ra em cũng học hỏi được rất nhiều lời khuyên, kinh nghiệm và tác phong làm việc từ thầy. Em xin kính chúc thầy luôn mạnh khỏe, đạt được thật nhiều thành công trong công tác giảng dạy và nghiên cứu cũng như trong cuộc sống.

Em xin gửi lời cảm ơn tới anh chị, bạn bè đã và đang theo học tại Học viện Công nghệ Bưu chính viễn thông vì đã giúp đỡ em trong quá trình thực hiện nghiên cứu đồ án này.

Em xin chân thành cảm ơn tới các thầy, cô, cán bộ tại Học viện Công nghệ Bưu chính Viễn thông nói chung và các thầy cô giảng dạy tại khoa CNTT1 nói riêng, những người đã luôn quan tâm, ủng hộ và tạo cho sinh viên chúng em một môi trường lý tưởng trong suốt trình em học tập tại Học viện.

Cuối cùng, em xin cảm ơn gia đình em, những người luôn là nguồn động viên, khích lệ và chỗ dựa vững chắc cho em để có thể hoàn thành được đồ án này.

Hà Nội, tháng 1 năm 2021

Vũ Chiến Thắng

TÓM TẮT NỘI DUNG ĐỒ ÁN

Trong thời đại phát triển của thương mại điện tử như hiện nay, số lượng người sử dụng hình thức thanh toán trực tuyến ngày một tăng, một trong những vấn đề mà cả nhà phát triển hệ thống và người dùng quan tâm là theo dõi tình hình của đơn hàng. Người dùng có thể cập nhật thông tin mà đơn hàng của mình đang đặt, và người quản lý phải nắm bắt được thông tin của đơn hàng để có thể phản hồi với khách hàng. Vì vậy, việc trích xuất trạng thái đơn hàng từ các log mà nhà vận chuyển cung cấp là một việc cần thiết. Vận dụng học máy để giải quyết vấn đề này đem lại nhiều ưu điểm so với các giải pháp trước đây như là xử lý được dữ liệu đầu vào (các log) ngẫu nhiên, xử lý các log có nhiều ngôn ngữ (do vận chuyển qua nhiều quốc gia khác nhau) hay đơn hàng sắp xếp sai thứ tự thời gian …

Cấu trúc Đồ án Tốt nghiệp được chia thành 5 chương chính, bao gồm:

Chương 1. Giới thiệu đề tài

Chương 2. Cơ sở lý thuyết

Chương 3. Phân tích và thiết kế hệ thống

Chương 4. Xây dựng mô hình học máy

Chương 5. Đánh giá kết quả và tổng kết

Trong mỗi chương, nội dung sẽ được trình bày thành các đề mục nhỏ nhằm phân tích rõ vấn đề và các bước cần thiết để triển khai từng nhiệm vụ.

MỤC LỤC

DANH SÁCH CÁC TỪ VIẾT TẮT VÀ THUẬT NGỮ

API	application programming interface
API	application programming interface
BERT	Bidirectional encoder representations from transformers
BERT	Bidirectional encoder representations from transformers
COMPLETED	khách hảng nhận hàng thành công
COMPLETED	khách hảng nhận hàng thành công
CSDL	Cơ sở dữ liệu
CSDL	Cơ sở dữ liệu
ĐATN	Đồ án Tốt nghiệp
ĐATN	Đồ án Tốt nghiệp
IN_US	đơn hàng tới Mỹ và được vận chuyển trong Mỹ
IN_US	đơn hàng tới Mỹ và được vận chuyển trong Mỹ
log	thông tin sự kiện của đơn hàng
log	thông tin sự kiện của đơn hàng
LSTM	Long-short term memory
LSTM	Long-short term memory
message	thông điệp
message	thông điệp
model	mô hình
model	mô hình
NLP	natural languague processing
NLP	natural languague processing
paper	Bài báo nghiên cứu khoa học được công bố

paper	Bài báo nghiên cứu khoa học được công bố
pre-trained model	Mô hình đã được huấn luyện trước
pre-trained model	Mô hình đã được huấn luyện trước
RETURN_TO_SENDER	khách hàng trả hàng
RETURN_TO_SENDER	khách hàng trả hàng
scm	supply chain management (quản lý chuỗi cung ứng)
scm	supply chain management (quản lý chuỗi cung ứng)
Tập dữ liệu test	tập dữ liệu đánh giá
Tập dữ liệu test	tập dữ liệu đánh giá
Tập dữ liệu training	tập dữ liệu huấn luyện
Tập dữ liệu training	tập dữ liệu huấn luyện
tracking number	mã đơn hàng
tracking number	mã đơn hàng
TRACKING_AVAILABLE	đơn hàng được tạo
TRACKING_AVAILABLE	đơn hàng được tạo
TRACKING_ONLINE	đơn hàng được vận chuyển trong Trung Quốc
TRACKING_ONLINE	đơn hàng được vận chuyển trong Trung Quốc
USPS	Dịch vụ bưu chính Hoa Kỳ
USPS	Dịch vụ bưu chính Hoa Kỳ

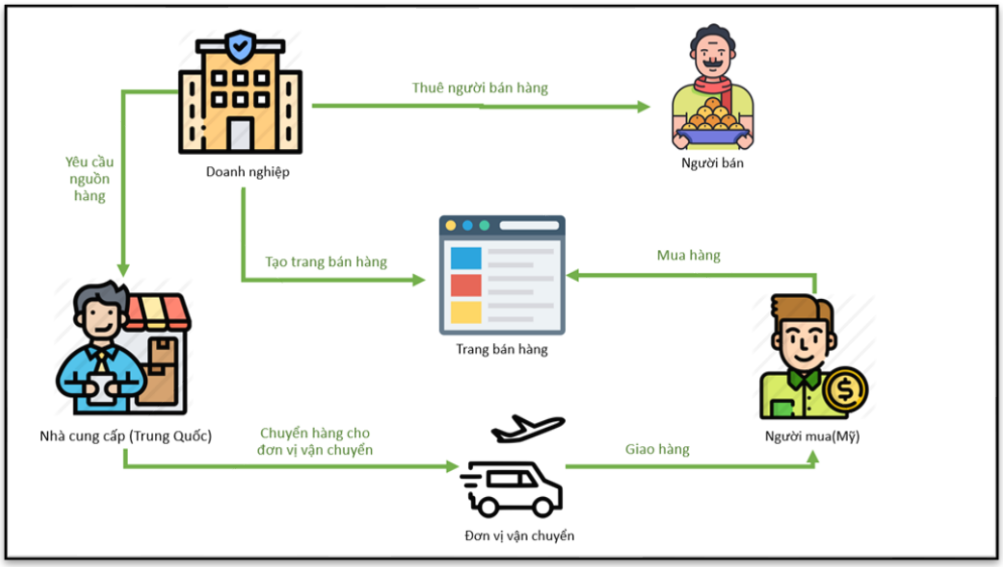
DANH MỤC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Giới thiệu mô hình kinh doanh

Đề tài xuất phát từ một ý tưởng nhằm cải thiện chu trình quản lý hậu cần logistic cho một công ty thương mại nhằm giải quyết tình trạng phản hồi trả hàng được nhanh chóng và hiệu quả.

Sau đây là hình ảnh mô tả nguyên tắc vận hành của chuỗi hậu cần.



Hình 1.1 Mô hình kinh doanh

Mô tả:

- Doanh nghiệp: tạo các trang Website bán sản phẩm của mình trên Internet (doanh nghiệp đầu tư chủ yếu vào thị trường Mỹ), sau đó thuê những người bán hàng để bán sản phẩm của trình trên trang Web. Khi có đơn hàng được đặt, doanh nghiệp sẽ lấy nguồn hàng từ nhà cung cấp ở Trung Quốc để giao cho khách hàng.
- Người bán: có thể chỉnh sửa nội dung của trang bán hàng theo ý của mình, có trách nghiệm quảng cáo trang bán hàng trên các trang mạng xã hội như Facebook, Instagram, … để thu hút người mua. Người bán thu lợi nhuận từ số lượng sản phẩm bán được cho khách hàng.
- Nhà cung cấp (Trung Quốc): được doanh nghiệp ký hợp đồng, khi có đơn hàng họ có trách nghiệm cung cấp hàng cho đơn vị vận chuyển.
- Đơn vị vận chuyển: có nhiệm vụ vận chuyển đơn hàng từ Trung Quốc sang Mỹ và giao sản phẩm đến tận tay người mua, đơn vị vận chuyển có thể kể đến là USPS.
- Người mua: là khách hàng bên Mỹ, họ có nhu cầu tìm kiếm và mua những sản phẩm may mặc như: quần, áo, tất, … mà doanh nghiệp cung cấp.

1.2 Đặt vấn đề

Trong hệ thống thương mại điện tử, việc theo dõi trạng thái của đơn hàng là cần

thiết, giúp cho doanh nghiệp giám sát được đơn hàng cũng như có thể giải quyết những vấn đề phát sinh với đơn hàng và với khách hàng. Những vấn đề phát sinh ở đây có thể là:

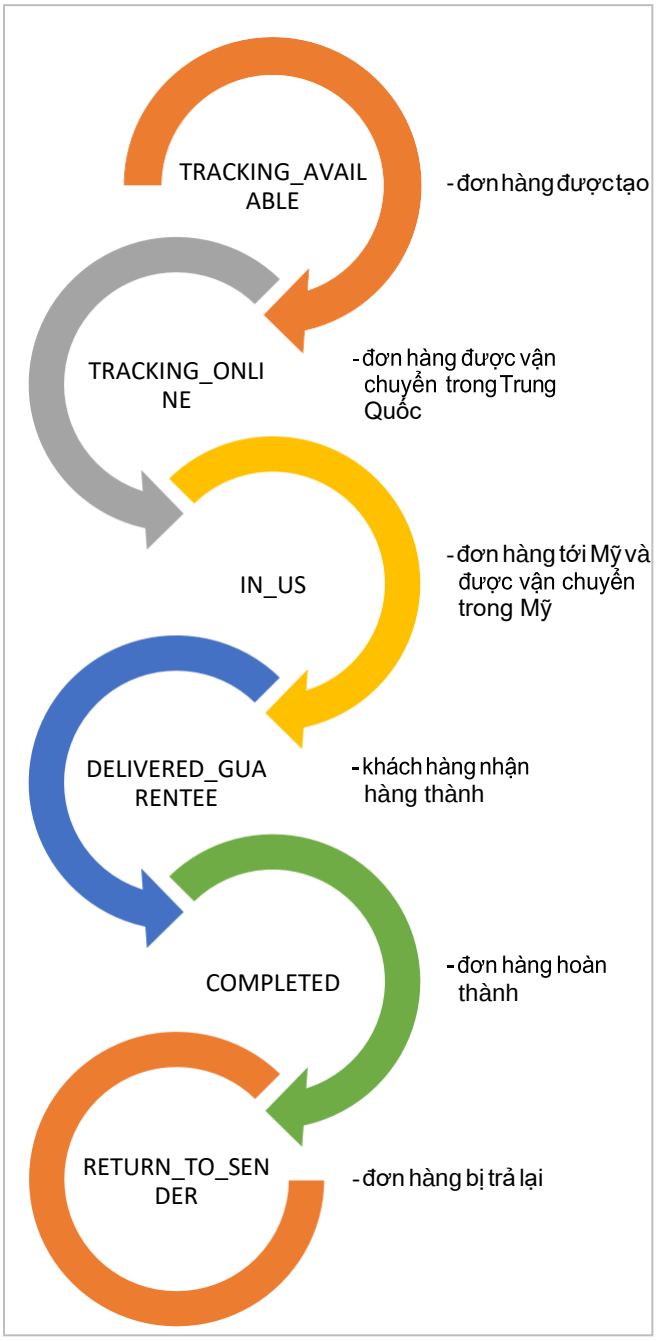
- Hàng bị thất lạc trong quá trình vận chuyển
- Hàng bị giao chậm
- Khách hàng muốn trả hàng vì không muốn mua nữa, mua nhầm, hàng không đúng yêu cầu, …

Việc nắm rõ, chính xác trạng thái của đơn hàng giúp cho doanh nghiệp có thể phản hồi một cách kịp thời tới khách hàng, điều này giúp thể hiện sự quan tâm và tạo sự tin tưởng cho khách hàng.

Một đơn hàng có thể có 5 trạng thái như hình bên:

Thông tin của đơn hàng (log) đi cùng với mã đơn hàng (tracking number). Khi có một sự kiện mới phát sinh của đơn hàng sẽ được cập nhật vào log và thông báo cho doanh nghiệp. Tuy nhiên thông tin này không phải là các trạng thái trên mà là đoạn văn bản do **người vận chuyển nhập tay**. Ví dụ như: Shipment information received (đã nhận thông tin vận chuyển), đôi khi thông tin này không phải là tiếng Anh mà là **tiếng Trung** vì do bên vận chuyển Trung Quốc nhập.

Giải pháp trước đây để trích xuất trạng thái từ các log này là lưu trữ các mẫu câu của các đơn hàng đã có trước đây (string pattern). Sau đó khi có một sự kiện mới đến, sẽ so sánh với các mẫu câu đã lưu trữ với sự kiện mới để suy ra trạng thái mới của đơn hàng. Tuy nhiên, giải pháp này có một số nhược điểm như:



Hình 1.2 Trạng thái của đơn hàng

- **Loại dữ liệu mới:** Mỗi khi xuất hiện một sự kiện có nội dung mới mà hệ thống chưa lưu trữ, mỗi lần như vậy phải lưu trữ nội dung mới vào.
- **Chuỗi sự kiện:** Trích xuất trạng thái của đơn hàng từ sự kiện xảy ra mới nhất: một đơn hàng có quá trình từ lúc bắt đầu cho tới lúc kết thúc, vì thế phải lấy tất cả các sự kiện của đơn hàng để trích xuất ra trạng thái, nếu chỉ lấy sự kiện mới nhất sẽ không đem lại độ chính xác cao.
- **Trình tự thời gian:** Các sự kiện của đơn hàng có thể bị sắp xếp sai theo thứ tự thời gian, ví dụ như khi đơn hàng tới Mỹ, bên Mỹ nhập thông tin trước bên Trung Quốc khiến cho thứ tự sự kiện bị sai.
- **Ngôn ngữ:** Các log chứa tiếng Trung, giải pháp trước không thể xử lý được

nếu chỉ dùng việc so sánh mẫu câu.

Hình 1.3 Bốn vấn đề của hệ thống hiện tại

1.3 Hướng giải quyết

Từ việc tìm hiểu và nghiên cứu, giải pháp đề xuất **áp dụng mô hình học máy** để giải quyết các vấn đề trên. Giải pháp có một số ưu điểm mà mô hình học máy đem lại so với giải pháp cũ như sau:

- **Hiểu ngữ nghĩa:** Mô hình học máy có khả năng hiểu được ngữ nghĩa của câu nên khi cho một đoạn log vào (có thể là log mà chưa từng cho mô hình học) mô hình có thể đưa ra kết quả có độ chính xác tương đối.
- **Xâu chuỗi sự kiện:** Mô hình được học là dữ liệu chứa tất cả các sự kiện của đơn hàng nên khi đưa ra dự đoán sẽ đạt được độ chính xác cao.
- **Phát hiện lỗi trình tự:** Có thể phát hiện được đơn hàng có các log sai thứ tự.
- **Dịch ngôn ngữ:** Ngoài ra, để giải quyết các log tiếng Trung, đầu tiên sử dụng Google Translate để dịch log ra tiếng Anh sau đó sẽ đưa vào mô hình để dự đoán.

Hình 1.4 Hướng giải quyết

1.3.1 Google Colab

Hình 1.5 Google Colab logo

Google Colab (Colaboratory) là một dịch vụ đám mây miễn phí của Google, hiện nay có hỗ trợ GPU (Tesla K80) và TPU (TPUv2). Do được phát triển dựa trên Jupiter Notebook nên việc sử dụng Google Colab cũng tương tự như việc sử dụng Jupyter Notebook. Google Colab là một công cụ lý tưởng để rèn luyện kĩ năng lập trình với ngôn ngữ Python thông qua các thư viện của deep learning. Google Colab cài đặt sẵn cho những thư viện rất phổ biến trong nghiên cứu Deep Learning như PyTorch, TensorFlow, Keras và OpenCV.

Google Colab cho phép chạy các dòng code python thông qua trình duyệt, đặc biệt phù hợp với Data analysis, machine learning và giáo dục. Colab không cần yêu cầu cài đặt hay cấu hình máy tính, mọi thứ có thể chạy thông qua trình duyệt, có thể sử dụng tài nguyên máy tính từ CPU tốc độ cao và cả GPUs và cả TPUs đều được cung cấp.

Em sử dụng công cụ Google Colab để huấn luyện mô hình học máy vì sức mạnh từ GPU mà nền tảng này cung cấp, giúp cho việc huấn luyện mô hình đạt tốc độ và hiệu quả cao.

1.3.2 Kafka

Hình 1.6 Kafka logo

Kafka là dự án mã nguồn mở, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao và là hệ thống nhắn tin nhanh. Vì tính đáng tin cậy, Kafka đang dần được thay thế cho hệ thống nhắn tin truyền thống. Nền tảng được sử dụng cho các hệ thống nhắn tin thông thường trong các ngữ cảnh khác nhau. Đây là hệ quả khi khả năng mở rộng ngang và chuyển giao dữ liệu đáng tin cậy là những yêu cầu quan trọng nhất.

Kafka là một hệ thống message theo cơ chế Publish-Subcribe (xuất bản/ đăng

ký), cho phép các nhà sản xuất (gọi là producer) đẩy các message lên Kafka mà một, hoặc nhiều người tiêu thụ (gọi là consumer) có thể đọc, xử lý được những message đó.

Em sử dụng Kafka để lưu trữ các log được gửi từ đơn vị USPS, sau đó gửi các log này đến API của model để lấy được trạng thái.

1.3.3 Elasticsearch

Hình 1.7 Elasticsearch logo

Elasticsearch là một công cụ tìm kiếm dựa trên nền tảng Apache Lucene. Công cụ này cung cấp một bộ máy tìm kiếm dạng phân tán, có đầy đủ công cụ với một giao diện web HTTP có hỗ trợ dữ liệu JSON. Elasticsearch được phát triển bằng Java và được phát hành dạng nguồn mở theo giấy phép Apache.

Elasticsearch là một search engine (công cụ tìm kiếm), hoạt động như một web server, có khả năng tìm kiếm nhanh chóng (near realtime) thông qua giao thức RESTful. Ngoài ra, Elasticsearch còn có khả năng phân tích và thống kê dữ liệu.

Em sử dụng Elasticsearch để lưu trữ dữ liệu (dạng json) chứa log và trạng thái mà model dự đoán của đơn hàng.

1.3.4 PostgreSQL

Hình 1.8 PostgreSQL logo

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ – đối tượng, được phát triển bởi Khoa Điện toán, Đại học California – Hoa Kỳ dựa trên Postgres bản 4.2. Chương trình này đã mở đường cho nhiều khái niệm về hệ quản trị dữ liệu thương mại sau này. PostgreSQL là mã nguồn mở miễn phí, được xây dựng theo chuẩn SQL99. Người dùng có thể tự do sử dụng, chỉnh sửa và phân bổ PostgreSQL theo nhiều hình thức khác nhau.

Ban đầu, hệ quản trị được thiết kế để chạy trên các nền tảng tương tự như Unix. Sau này, PostgreSQL được điều chỉnh trở nên linh động và chạy trên nhiều nền tảng khác nhau như Windows, Mac OS X, Solaris với nhiều tính năng và đặc điểm nổi bật.

So với nhiều hệ quản trị cơ sở dữ liệu khác, PostgreSQL không quá yêu cầu về công tác bảo trì bởi tính ổn định cao, có thể phát triển nhiều ứng dụng khác nhau với chi phí tương đối thấp.

Em sử dụng PostgreSQL để lưu trữ trạng thái mới nhất của các đơn hàng, phục vụ cho như cầu vẽ biểu đồ, báo cáo.

1.3.5 NiFi

Hình 1.9 NiFi logo

Apache NiFi là một phần mềm mã nguồn mở viết bằng Java, được tạo ra để tự động hóa luồng dữ liệu giữa các hệ thống phần mềm với nhau. Nền tảng này được xây dựng từ năm 2006 dựa trên phần mềm "NiagaraFiles" phát triển bởi anh NSA, sau đó được chuyển sang mã nguồn mở vào năm 2014 (Nguồn Wiki).

NiFi được biết đến với khả năng xây dựng luồng chuyển dữ liệu tự động giữa các hệ thống. Đặc biết là hỗ trợ rất nhiều kiểu nguồn và đích khác nhau như:

- Các loại cơ sở dữ liệu dạng quan hệ: Oracle, MySql, Postgre, …
- Các loại cơ sở dữ liệu NoSQL: Mongo, HBase, Cassandra, …
- Từ các nguồn web như: HTTP, web-socket
- Lấy hoặc đẩy dữ liệu streaming vào Kafka
- Hay là từ: FTP, log

Ngoài việc lấy ra và đẩy vào dữ liệu thì NiFi còn các chức năng như routing (định tuyến) dữ liệu theo thuộc tính và nội dung, xử lý dữ liệu như: lọc, chỉnh sửa, thêm bớt nội dung của dữ liệu trước khi đưa đến nơi lưu trữ.

Em sử dụng NiFi để tạo một luồng chuyển dữ liệu tự động từ Elasticsearch sang PostgreSQL.

1.3.6 Grafana

Hình 1.10 Grafana logo

Grafana là một nền tảng mã nguồn mở chuyên phục vụ mục đích theo dõi, giám sát và đánh giá các số liệu thu được thông qua các biểu đồ trực quan. Nền tảng cung cấp nhiều loại biểu đồ đa dạng như biểu đồ đường, biểu đồ tròn, biểu đồ cột, bảng biểu, … và các thông tin dữ liệu thu được được biểu diễn theo dòng thời gian giúp cho hiển thị tối ưu trên Grafana. Grafana có thế kết nối tới nhiều nguồn cơ sở dữ liệu đa dạng như: MySQL, PostgreSQL, ElasticSearch, Google Cloud Monitoring, … Bên cạnh đó, chương trình chạy trên nền Web, được thiết kế trực quan, dễ sử dụng và có thể được truy cập từ nhiều thiết bị khi dùng chung một mạng.

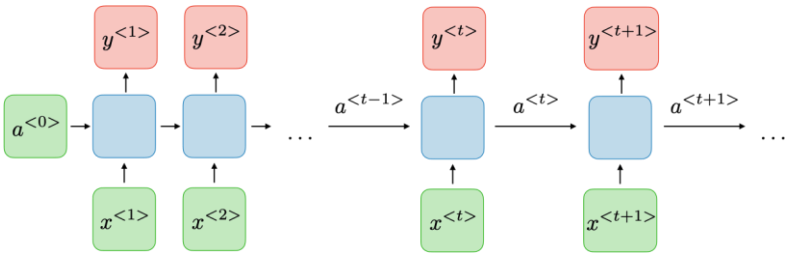
Em sử dụng Grafana để vẽ các đồ thị thống kê trạng thái của đơn hàng từ cơ sở dữ liệu Postgre, nhằm theo dõi và giám sát kết quả của các mô hình học máy.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Học máy LSTM

2.1.1 Kiến trúc RNN (Recurrent Neural Network)

RNN – Recurrent neural network (mạng nơ-ron hồi quy), là một lớp của mạng nơ-ron, cho phép các đầu ra trước đó được dùng như đầu cho các tầng ẩn kế tiếp của mô hình. Kiến trúc RNN được minh hoạ như sau:



Hình 2.1 Kiến trúc RNN

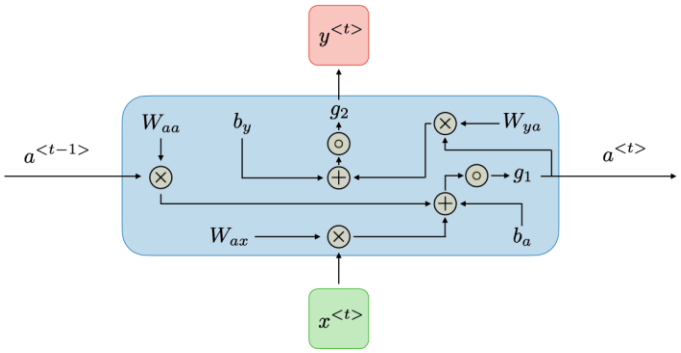
- Hàm lan truyền xuôi (forward propagation):

Sau mỗi bước t, giá trị kích hoạt $a^{<t>}$ và đầu ra $y^{<t>}$ được tính như sau:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$
$$\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Trong đó:

- $x^{<t>}$ là giá trị đầu vào
- g_1 và g_2 là các hàm kích hoạt
- W_{aa}, W_{ax}, Way, b_a và b_y là các siêu biến (hyper parameters)



Hình 2.2 Chi tiết trong ô nhớ (memory cell)

- Hàm mất mát (loss function):

Hàm mất mát L của cả quá trình được tính dựa trên sự mất mát tại mỗi thời điểm t.

Hàm mất mát tại thời điểm t:

$$L(\hat{y}, y) = -y^{<t>}\, log\, log\, (\hat{y}^{<t>}) - (1 - y^{<t>})\, log\, log\, (1 - \hat{y}^{<t>})$$

Hàm mất mát của cả quá trình:

$$L(\hat{y}, y) = \sum_{t=0}^{T_y} L(\hat{y}^{<t>}, y^{<t>})$$

Trong đó:

- T_y là độ dài của chuỗi đầu ra

- Hàm lan truyền ngược (back propagation):

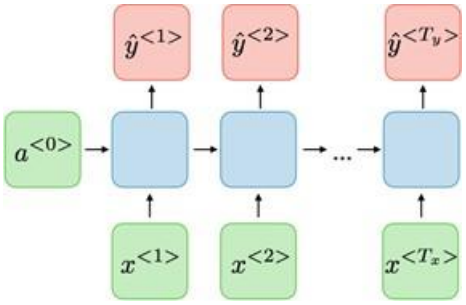
Tại mỗi thời điểm T, đạo hàm của hàm mất mát L với ma trận trọng số W được tính như sau:

$$\frac{\partial L^{(T)}}{\partial W} = \sum_{t=1}^{T} \frac{\partial L^{(T)}}{\partial W}|_{(t)}$$

Các mô hình của kiến trúc RNN:

1. Many-to-many (nhiều nhiều)

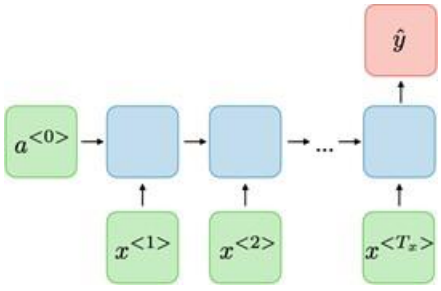
Với $T_x = T_y$: độ dài của chuỗi đầu vào bằng với độ dài của chuỗi đầu ra. Mô hình được dùng để giải quyết bài toán Named-entity recognition (nhận dạng tên thực thể)



Hình 2.3 Mô hình many-to-many

2. Many-to-one (nhiều một)

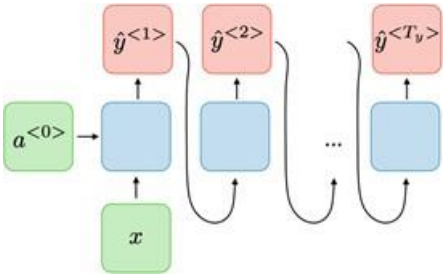
Với $T_x > 1$, $T_y = 1$: độ dài của chuỗi đầu vào bằng lớn hơn 1, độ dài của chuỗi đầu ra bằng 1. Mô hình được dùng để giải quyết bài toán Sentiment classification (phân loại cảm xúc)



Hình 2.4 Mô hình many-to-one

3. One-to-many (một một)

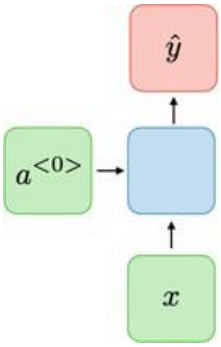
Với $T_x = 1$, $T_y > 1$: độ dài của chuỗi đầu vào bằng 1, độ dài của chuỗi đầu ra lớn hơn 1. Mô hình được dùng để giải quyết bài toán Music generation (tạo âm nhạc)



Hình 2.5 Mô hình one-to-many

4. One-to-one (một nhiều)

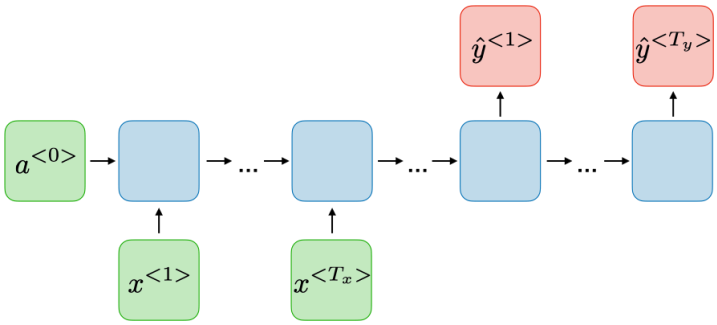
Với $T_x = T_y = 1$: độ dài của chuỗi đầu vào bằng độ dài của chuỗi đầu ra và bằng 1. Mô hình được dùng để giải quyết bài toán Binary classification (phân loại nhị phân)



Hình 2.6 Mô hình one-to-one

5. Many-to-many (nhiều nhiều)

Với $T_x \neq T_y$: độ dài của chuỗi đầu vào khác với độ dài của chuỗi đầu ra. Mô hình được dùng để giải quyết bài toán Machine translation (dịch máy)



Hình 2.7 Mô hình many-to-many

2.1.2 Biểu diễn từ trong mô hình (Word representation)

Biểu diễn từ trong xử lý ngôn ngữ tự nhiên (NLP) là một phần cơ bản trong việc xây dựng các khối (block) trong mô hình. Việc này có ảnh hưởng đáng kể tới hiệu năng của mô hình học sâu. Ý tưởng của đề xuất này là sẽ biểu diễn mỗi từ trong đoạn văn bản đầu vào thành một vector, các vector này có độ dài bằng nhau. Sau đây là các phương pháp cụ thể:

2.1.2.1. One-hot encoding

Ý tưởng chính của phương pháp này là tạo một vector bao gồm các giá trị 0 và

duy nhất một giá trị 1. Cụ thể, đối với một từ khi được biểu diễn bằng vector, chỉ cột tương ứng với chỉ số của từ đó trong từ điển có giá trị bằng 1, còn lại là bằng

0. Khi đó, vector có kích cỡ là 1x(N+1), trong đó N là kích cỡ của vector và thêm giá trị 1 là cho những từ nằm ngoài từ điển. Hãy xem ví dụ sau:

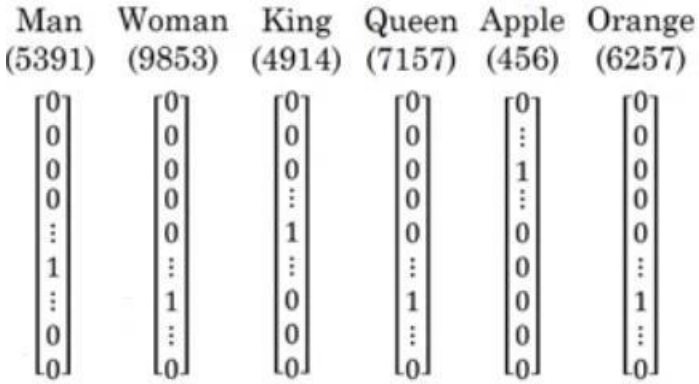
Ta có bộ từ điển tiếng Anh sau:

V = [a, aaron, …, zulu, <UNK>] , |V| = 10000

Trong đó:

- <UNK>: unknown word (từ không có trong từ điển)
- |V|: độ dài của từ điển

Sử dụng từ điển V để biểu diễn các từ sau: Man, Woman, King, Queen, Apple, Orange thành các vector. Thu được kết quả như sau:



Hình 2.8 Ví dụ cho One-hot encoding

Các vector có kích cỡ là |V| + 1 = 10001. Ví dụ vector biểu diễ từ Man có duy nhất một có gái trị một, vị trí của cột này là vị trí của từ Man trong từ điển V là 5391.

Nhược điểm có thể thấy của phương pháp này là vector của từ không có các đặc trưng về ngữ nghĩa để phân biệt với các từ khác và để đạt hiệu quả yêu cầu từ điển phải có kích cỡ lớn dẫn đến đòi hỏi bộ nhớ lớn để tính toán.

2.1.2.2. Word embedding

Ý tưởng chính của phương pháp này là biểu diễn các từ dưới dạng các vector đặc trưng. Mỗi thành phần trong vector là một đặc trưng được ẩn bên trong nghĩa của từ. Chúng có thể tiết lộ ngữ nghĩa hoặc ngữ nghĩa của từ. Trong ví dụ dưới đây là vector đặc trưng của các từ:

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

Hình 2.9 Ví dụ cho Word embedding

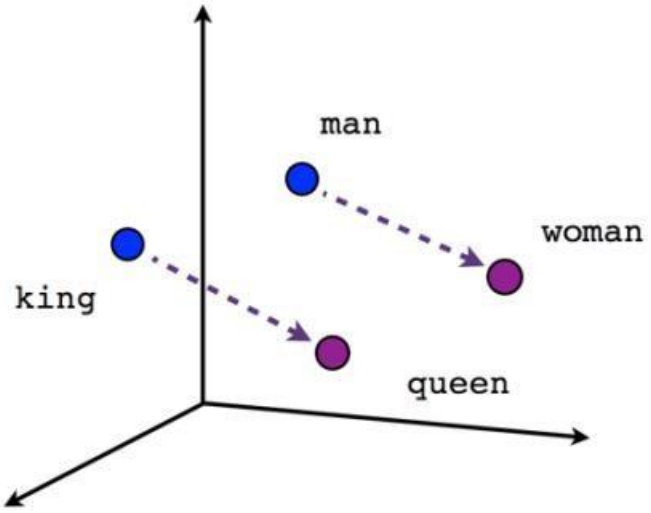
Các từ Man, Woman, King, Queen, Apple, Orange được biểu diễn thành các vector có các đặc trưng: Gender (giới tính), Royal (hoàng tộc), Age (tuổi) và Food (thực phẩm). Quan sát có thể thấy đặc trưng Royal của từ King và Queen có giá trị gần nhau, trong khi đó đặc trưng Food của hai từ Apple và Orange có giá trị gần nhau. Rõ ràng vấn đề ngữ nghĩa của từ được giải quyết với phương pháp Word embedding hiểu quả hơn hẳn với phương pháp One-hot encoding

2.1.2.3. Bài toán King Queen Man Woman

Bài toán này chứng minh sự khác biệt giữa từ King với Queen tương đồng với sự khác biệt giữa từ Man với Woman trong không gian vector. Có thể biểu diễn bằng công thức sau:

$e_{king} - e_{queen} \sim e_{man} - e_{woman}$

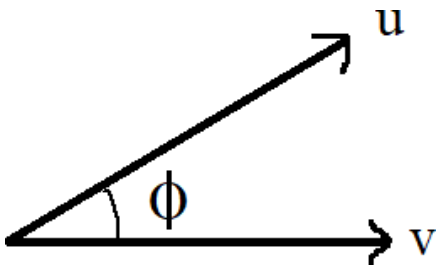
Trong đó: e là vector biểu diễn từ



Hình 2.10 Biểu diễn trong không gian vector

Trong không gian vector, công thức cô-sin dùng để so sánh sự tương đồng giữa

hai vector (Cosine similarity):



Hình 2.11 Biểu diễn hai vector u và v trong không gian vector

$$sin\ sin\ (u, v)\ = \frac{u * v}{||u|| * ||v||}\ (= cos\ (\Phi))$$

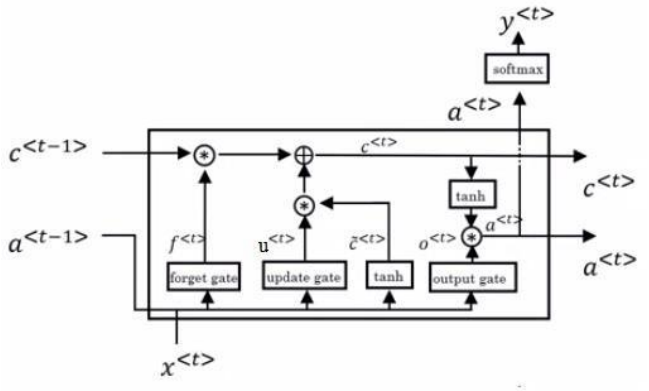
Với giá trị hàm cô-sin càng lớn chứng tỏ hai vector càng tương đồng.

2.1.2.4. Word2Vec

Word2Vec là thư viện được sử dụng rộng rãi cho việc biểu diễn từ, có thể kể đến mô hình Skip-Gram trong thư viện này. Word2Vec được tạo bởi đội ngũ được dẫn dắt bởi Tomas Mikolov tại Google. Chúng ta có thể huấn luyện Word2Vec trên tập dữ liệu của mình hoặc tải bộ vector đã được huấn luyện rồi. Hiện nay, Google đã công khai bộ vector này trên Google News dataset. Bộ này bao gồm khoảng 3 triệu từ và cụm từ được biểu diễn thành các vector 300 chiềuNgoài ra, còn có thư viện GloVe cũng được dùng phổ biến hiện nay.

2.1.3 Mô hình LSTM (Long-short term memory)

Mô hình LSTM kế thừa từ kiến trúc RNN, nhưng có cải tiến về cấu trúc trong các ô nhớ (memory cell), giúp cho việc ghi nhớ giữa các từ ở khoảng cách xa trong văn bản tốt hơn bằng việc bổ sung thêm forget gate, update gate. Hình vẽ dưới đây minh hoạ cho điều này:



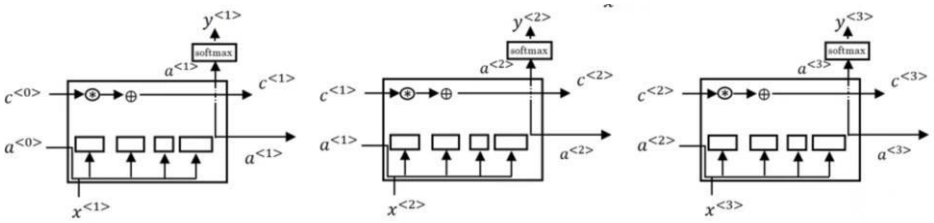
Hình 2.12 Ô nhớ (memory cell) trong LSTM

Trong đó:

$$\tilde{c}^{<t>} = tanh\,(W_c[a^{<t>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma\,(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma\,(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma\,(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * tanh\,(c^{<t>})$$

Trong đó:

- σ là hàm $sigmoid$: $s(x) = \frac{1}{1+e^{-x}}$
- $\Gamma_u, \Gamma_f, \Gamma_o$ lần lượt là đầu ra của update gate, forget gate và output gate Mô hình LSTM tổng quát như sau:



Hình 2.13 Mô hình LSTM

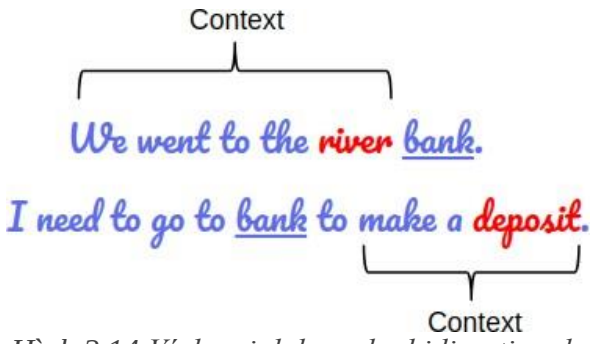
2.2 Học máy BERT

2.2.1 Giới thiệu về BERT

BERT được coi là state-of-the-art framework cho xử lý ngôn ngữ tự nhiên (NLP). BERT là viết tắt của Bidirectional Encoder Representations from Transformers (biểu diễn bộ mã hoá hai chiều từ kiến trúc Transformer). BERT đã được thiết kế huấn luyện trước trên các văn bản chưa được gán nhãn, bằng cách kết hợp ngữ cảnh từ hai bên trái và phải trong quá trình huấn luyện. Cụ thể, BERT đã được huấn luyện trước trên một tập dữ liêu lớn chưa được gắn nhãn bao gồm toàn bộ dữ liệu trên Wikipedia (khoảng 2500 triệu từ) và kho bản từ sách (khoảng 800 triệu từ). Kết quả là, mô hình pre-trained BERT (đã được huấn luyện) chỉ cần tinh chỉnh thêm một tầng đầu ra để giải quyết các yêu cầu của bài toán NLP.

BERT dựa trên kiến trúc Transformers, kiến trúc này được trình bày ở phần sau.

BERT là một mô hình "deeply bidirectional", điều này nghĩa là mô hình học những thông tin từ hai bên trái và phải của ngữ cảnh trong văn bản trong giai đoạn huấn luyện. Tính hai chiều của mô hình rất trọng cho việc hiểu được sâu sắc ý nghĩa từ. Cùng xem ví dụ minh hoạ dưới đây, hai câu cùng chứa từ "bank" nhưng ý nghĩa lại hoàn toàn khác nhau:



Hình 2.14 Ví dụ minh hoạ cho bidirectional

Nếu chỉ dùng một bên ngữ cảnh trái hoặc phải trong câu để dự đoán nghĩa của từ "bank" thì không chính xác. Trong câu thứ nhất, từ "bank" có nghĩa là bờ sông, ở câu thứ hai, từ bank có nghĩa là ngân hàng. Ý nghĩa của từ "bank" phụ thuộc vào các từ xung quang nó, cả bên trái và phải. Và đó chính xác là những gì BERT đã làm.

Khác với Word2Vec, mô hình pre-trained có tính chất "context-free" có nghĩa là mô hình tạo ra word embedding để biểu diễn cho từ trong từ điển, vì vậy từ "bank" trong "bank deposit" và "river bank" đều được biểu diễn bởi vector giống nhau, BERT thì khác, là mô hình "contextual" có nghĩa là vector biểu diễn cho từ được tạo ra dựa trên các từ khác trong cùng câu. Vì vậy vector biểu diễn cho từ "bank" trong hai câu ví dụ sẽ khác nhau.

Các mô hình pre-trained của BERT đã được Google công bố trên trang github của họ, bao gồm:

- `BERT-Large, Uncased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Large, Cased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Uncased` : 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Large, Uncased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Cased` : 12-layer, 768-hidden, 12-heads , 110M parameters
- `BERT-Large, Cased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Multilingual Cased (New, recommended)` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Multilingual Uncased (Orig, not recommended)` (Not recommended, use `Multilingual Cased` instead): 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Chinese` : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

Hình 2.15 Các mô hình pre-trained của BERT được Google công bố trên github

Bằng việc tải về từ trang github của Google, có thể khai báo và sử dụng mô hình phù hợp phục vụ cho công việc của mình.

2.2.2 Kiến trúc Transformer

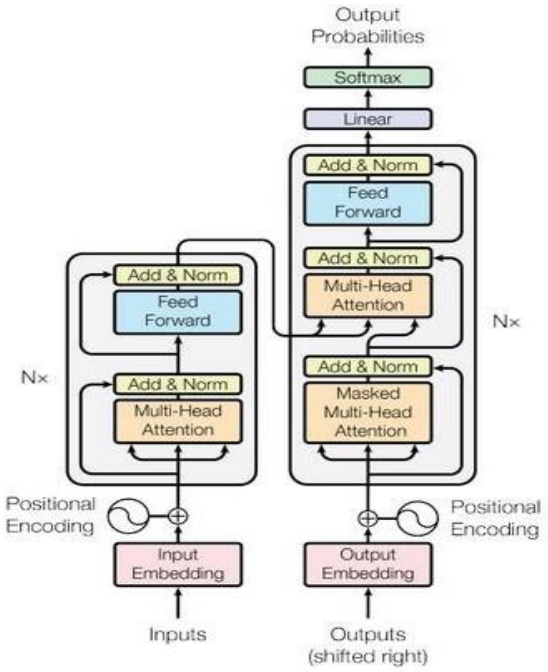
Transformer xuất hiện để giải quyết một số giới hạn của mô hình squence-to-sequence gặp phải (kiến trúc RNN dựa trên mô hình này) như:

- Xử lý thông tin liên hệ giữa các từ trong câu dài kém hiệu quả (long-range dependencies)
- Kiến trúc của mô hình chuỗi không thể tính toán song song mà phải tuần tự, dẫn đến tốc độ tính toán chậm

Transformer lần đầu tiên được đề cập trong bài báo "Attention Is All You Need".

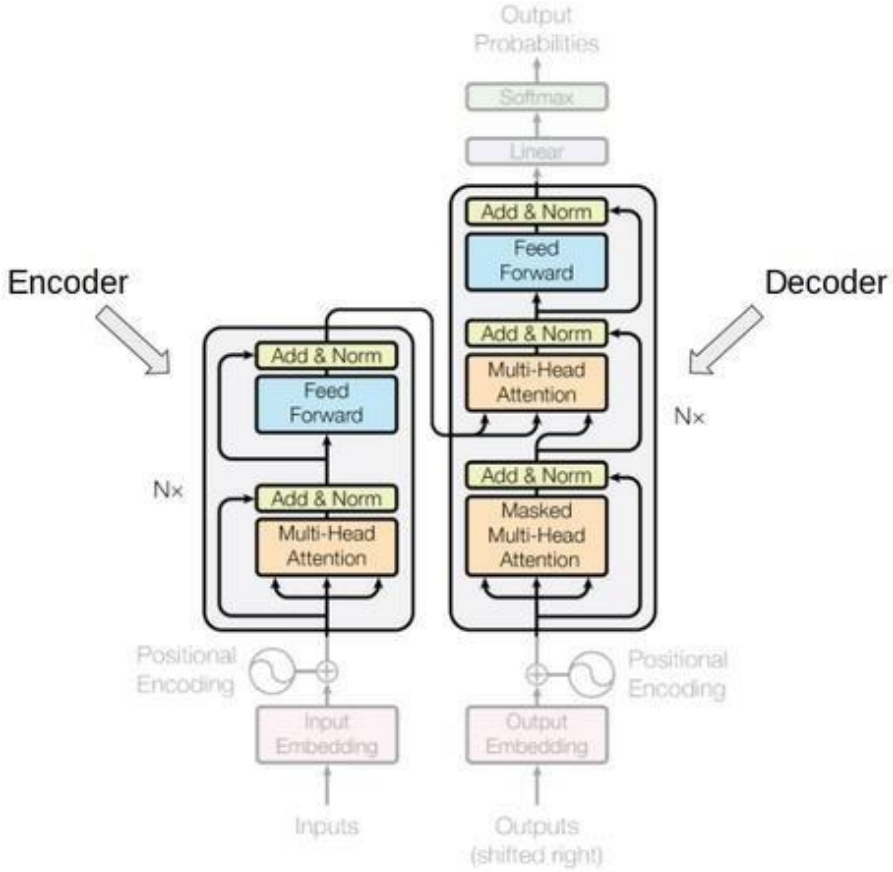
Trong bài báo có viết "Transformer là mô hình chuyển đổi đầu tiên dựa vào self-attention để tính toán biểu diễn cho đầu vào và đầu ra mà không cần dùng đến RNN hoặc tích chập"

Hình vẽ dưới đây minh hoạ cho kiến trúc Transformer:



Hình 2.16 Kiến trúc Transformer

Kiến trúc Transformer gồm hai phần chính là Encoder và Decoder:



Hình 2.17 Encoder và Decoder trong Transformer

Phần thứ nhất Encoder: bao gồm một chuỗi 6 tầng giống nhau. Trong mỗi tầng này có hai tầng con:

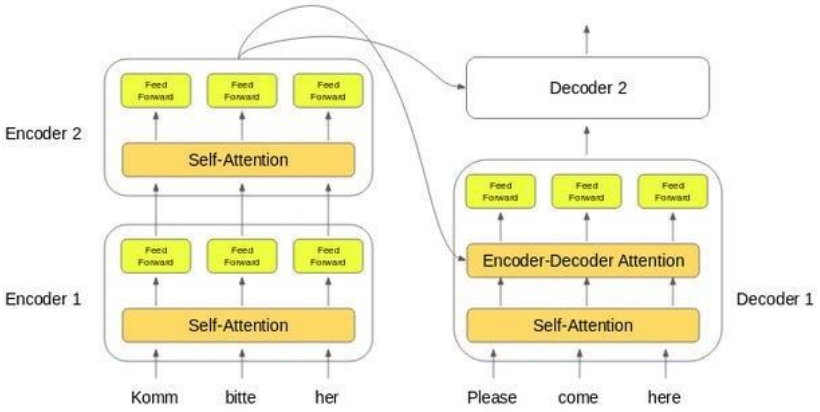
- Multi-Head Attention
- Feed Forward Nerual Network

Mỗi tầng con này đều được thêm vào tầng Normalization phía sau, vì vậy đầu ra của mỗi tầng con sẽ là LayerNorm (x + Sublayer(x)) với x là đầu vào của tầng con.

Phần thứ hai Decoder: cũng gồm một chuỗi 6 tầng giống với Encoder, ngoài ra trong mỗi tầng còn có thêm một tầng con là Masked Multi-Head Attention.

Công việc của encoder và decoder có thể được tóm tắt như sau:

- Word embeddings của chuỗi đầu vào được truyền vào khối encoder đầu tiên
- Sau đó, nó được biến đổi và lan truyền tới những khối encoder tiếp theo
- Đầu ra từ khối encoder cuối được truyền tới toàn bộ các khối decoder để tính toán, được minh hoạ ở hình bên dưới:

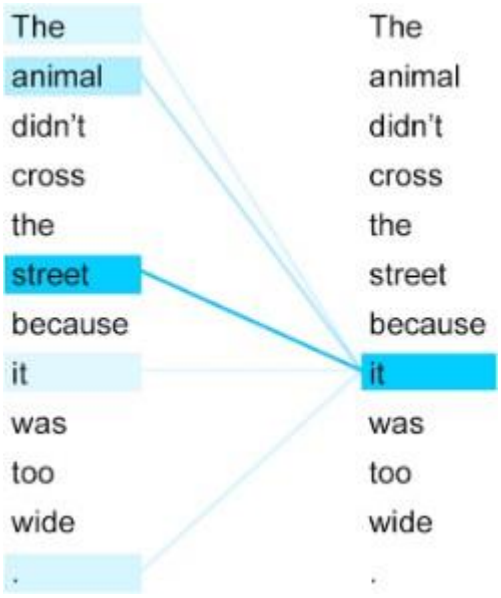


Hình 2.18 Đầu ra khối encoder cuối được đưa tới tất cả các khối decoder để tính toán

Trong mỗi khối decoder có thêm một tầng Encoder-Decoder Attention, giúp cho khối decoder tập trung vào những phần quan trọng của chuỗi đầu vào.

Một phần quan trọng của kiến trúc Transformer là Self-Attention, trong bài báo "Attention Is All You Need" có đề cập: "Self-Attention, đôi khi được gọi là intra-attention, là một cơ chế chú ý liên hệ đến những vị trí khác nhau trong một chuỗi đơn để tính toán nhằm biểu diễn cho chuỗi đó"

Hình vẽ dưới đây minh hoạ cho Self-Attention:



Hình 2.19 Ví dụ về Self-Attention

Nhờ vào cơ chế Self-Attention mà tính toán được từ "it" trong câu có liên hệ với từ "animal" mà không phải là "street". Self-Attention cho phép mô hình quan sát các từ trong câu để hiểu tốt hơn một từ cụ thể.

Việc tính toán Self-Attention được thể hiện như sau:

- Đầu tiên, mỗi khối encoder được cho đầu vào là 3 vector: query vector, key vector, value vector. Các vector này được khởi tạo ngẫu nhiên và luôn được cập nhật trong quá trình huấn luyện
- Tiếp đó, sẽ tính toán self-attention cho mỗi từ trong chuỗi đầu vào

Ta có ví dụ sau: câu "Action gets results", để tính self-attention cho từ "Action" cần tính toán với toàn bộ các từ còn lại trong câu mà có liên quan đến nó. Kết quả sẽ xác định tầm quan trọng của các từ đó với từ đang xét. Việc tính toán được thự hiện như sau:

Phép nhân ma trận của query vector (q1) với các key vector (k1, k2, k3) cho ra kết quả sau:

Word	q vector	k vector	v vector	score
Action	q_1	k_1	v_1	$q_1 . k_1$
gets		k_2	v_2	$q_1 . k_2$
results		k_3	v_3	$q_1 . k_3$

Hình 2.19 phép nhân ma trận của vector q1 với các vector k1, k2, k3

Sau đó kết quả được chia cho 8:

Word	q vector	k vector	v vector	score	score / 8
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$

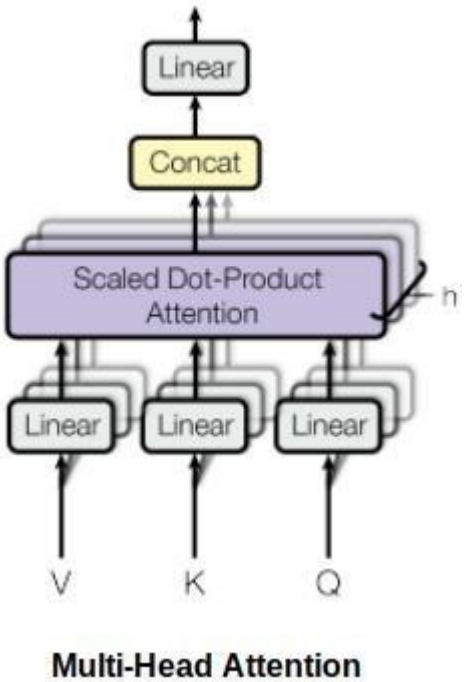
Hình 2.20 Kết quả được chia cho 8

Các kết quả được chuẩn hoá bằng hàm softmax rồi sau đó nhân với value vector và tổng của chúng là kết quả cuối cùng:

Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$	x_{11}	$x_{11} * v_1$	z_1
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$	x_{12}	$x_{12} * v_2$	
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$	x_{13}	$x_{13} * v_3$	

Hình 2.21 Kết quả cuối cùng của self-attention

Các từ còn lại được tính toán bằng cách tương tự. Hơn nữa, việc tính toán self-attention của các từ được thực hiện song song và độc lập. Sau đó, Multi- Head Attention thực hiện nối chúng lại với nhau và thực hiện phép biến đổi tuyến tính:



Multi-Head Attention

Hình 2.22 Tầng Multi-Head Attention

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1 Phân tích bài toán thực tế và hạ tầng

Trên cơ sở phân tích thực trạng trong mục *Đặt vấn đề*, doanh nghiệp cần phải xây dựng được một hệ thống bổ trợ giúp phân tích các log giao dịch và xác định chính xác và nhanh chóng thời điểm khách hàng trả hàng để báo động kịp thời cho nhà quản lý. Từ đó, giải pháp cải thiện giúp nâng cao chất lượng dịch vụ cho doanh nghiệp, nâng cao sự hài lòng cho khách hàng.

Ước lượng, giải pháp cải thiện cần có các tính năng như sau:

- **Hệ thống cập nhật, lưu trữ và thông báo sự kiện của đơn hàng**: vì các sự kiện xảy ra với số lượng lớn và liên tục trong hệ thống thương mại điện tử. Nên cần một hệ thống xử lý các sự kiện hoạt động ổn định và toàn vẹn.
- **Áp dụng học máy vào trích xuất trạng thái của đơn hàng và phát hiện đơn hàng sắp xếp sai thứ tự log**: để áp dụng các mô hình học máy đã huấn luyện vào trong hệ thống, cần xây dựng một service cung cấp API cho bên ngoài, khi có một yêu cầu từ bên ngoài chứa log của đơn hàng gọi đến API đó, các mô hình học máy bao gồm LSTM và BERT sẽ tiến hàng dự đoán và trả về kết quả.
- **Xử lý đơn hàng có log tiếng Trung**: khi đưa log có tiếng Trung vào cho mô hình học máy xử lý thì không thể được, vì ngôn ngữ đang sử dụng để huấn luyện là tiếng Anh. Nên cần phải dịch từ tiếng Trung sang tiếng Anh sau đó mới đưa vào mô hình để dự đoán.
- **Cơ sở dữ liệu**: với lượng dữ liệu lớn như trong thương mại điện tử, việc lưu trữ và truy vấn dữ cần phải nhanh chóng. Cơ sở dữ liệu dạng quan hệ (Relational Database) như MySQL, SQL Server, … cho thời gian truy vấn lâu dù cho có đánh Index trong bảng. Vì vậy, giải pháp là lựa chọn cơ sở dữ liệu dạng NoSQL như MongoDB hay Elasticsearch, các cơ sở dữ liệu dạng này cho phép truy vấn dữ liệu nhanh, và có thể mở rộng hoặc thu hẹp các trường dữ liệu.
- **Giám sát, thống kê số liệu**: để giám sát việc dự đoán của mô hình trong dữ liệu thực tế, cần phải thống kê các số liệu và kiểm tra qua các biểu đồ, đồ thị. Từ đó, có thể điều chỉnh các mô hình học máy sao cho phù hợp với dữ liệu thực tế.

3.2 Đề xuất giải pháp công nghệ

- **Kafka**: theo như các tìm hiểu về các hệ thống thương mại điện tử, hầu như họ đều sử dụng Kafka để giải quyết bài toán xử lý các sự kiện của đơn hàng. Số lượng sự kiện lớn và các sự kiện này diễn ra rời rạc và liên tục, yêu cầu hệ thống phải lưu trữ toàn vẹn và phản hồi nhanh chóng. Hệ thống messaging queue (hàng đợi thông điệp) như Kafka là một lựa chọn rất phù hợp cho công việc này.

- **Google Translate**: để xử lý các log tiếng Trung, sử dụng Google Translate vì nó đơn giản và hiệu quả. Để dùng được, cần đã đăng ký dịch vụ của nền tảng này (có trả phí), và được cung cấp một API key. Mỗi lần cần sử dụng, cần gửi log tiếng trung kèm với API key lên API mà nền tảng này cung cấp.

- **FastAPI**: khi tìm hiểu về các thư viện viết API trong python, một số thư viện có sẵn như: Flask, Falcon và FastAPI. Ưu điểm của các thư viện này là việc triển khai chúng đơn giản và không quá phức tạp. Tuy nhiên, lựa chọn thư viện FastAPI vì thư viện này cung cấp giao diện swagger (tài liệu API giúp xem được các API và kiểu dữ liệu trả về của nó) và cung cấp hàm bất đồng bộ (async/await).

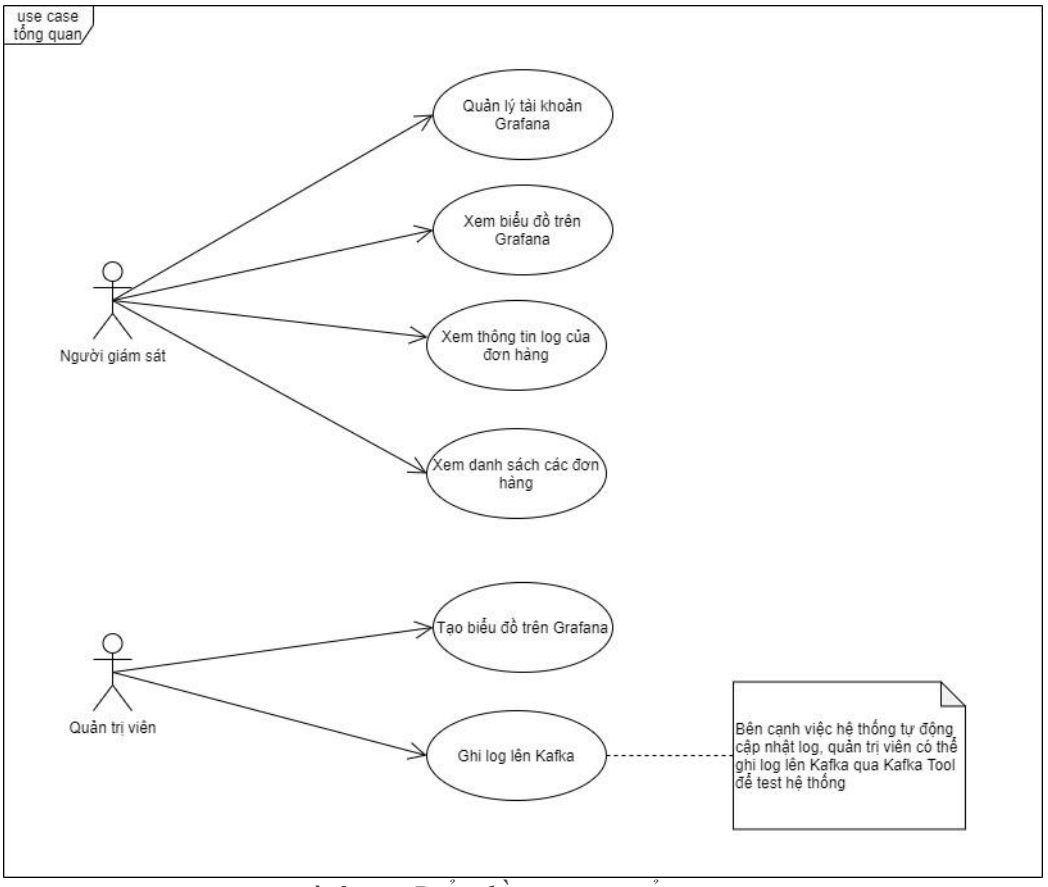
- **Elasticsearch**: công việc lưu trữ dữ liệu và vẽ biểu đồ để giám sát dữ liệu cũng rất quan trọng, vì vậy giải pháp là lựa chọn lưu trữ dữ liệu trên Elasticsearch và Postgres. Elasticsearch dùng để lưu trữ kết quả trả về của service học máy và log của đơn hàng, kiểu dữ liệu lưu mà hệ thống này lưu trữ là JSON, và theo index, document. Các index giống như bảng trong MySQL và các document giống như các hàng trong bảng. Ưu điểm của việc lưu trữ này là dữ liệu trong index có cấu trúc đa dạng, các document có thể có các trường khác nhau và việc mở rộng các trường được cho phép. Hơn nữa, việc kết nối tới hệ thống ElasticSearch cũng không phức tạp, và việc thêm/sửa/xoá dữ liệu đơn giản, chỉ việc gửi các lệnh HTTP Request tới hệ thống.

- **Postgres và Grafana**: sử dụng cơ sở dữ liệu Postgres kết hợp với phần mềm Grafana cho việc vẽ biểu đồ và đồ thị. Cơ sở dữ liệu Postgres lưu trữ mã đơn hàng cùng với các kết quả từ các mô hình học máy. Vì việc các câu lệnh truy vấn SQL trên Grafana đơn giản và dễ thực hiện hơn với câu lệnh truy vấn dạng Lucene của Elasticsearch. Bên cạnh đó, Grafana cung cấp nhiều loại biểu đồ trực quan như: biểu đồ cột, biểu đồ đường, biểu đồ tròn, biểu đồ nhiệt, bảng biểu, … nên lựa chọn Grafana rất phù hợp.

- **NiFi**: để truyền dữ liệu từ Elasticsearch sang Postgres, cụ thể là từ dữ liệu có kiểu là JSON sang dữ liệu dạng bảng. Qua quá trình tìm hiểu, NiFi là một công cụ mã nguồn mở rất hữu ích cho việc xây dựng luồng xử lý dữ liệu và truyền lượng dữ liệu lớn giữa các cơ sở dữ liệu. Công cụ này cung cấp giao diện kéo thả, rất trực quan cho người sử dụng. Tiếp đến là việc, mỗi lần lấy dữ liệu từ Elasticsearch không thể lấy toàn bộ mà chỉ lấy những dữ liệu mới được cập nhật, và NiFi có thể làm được điều này. Cụ thể, giải pháp tạo một luồng truyền dữ liệu và cập nhật một biến thời gian trong luồng. Mỗi lần lấy dữ liệu sẽ truy vấn tới biến này và chỉ lấy dữ liệu được cập nhật 5 phút gần nhất.

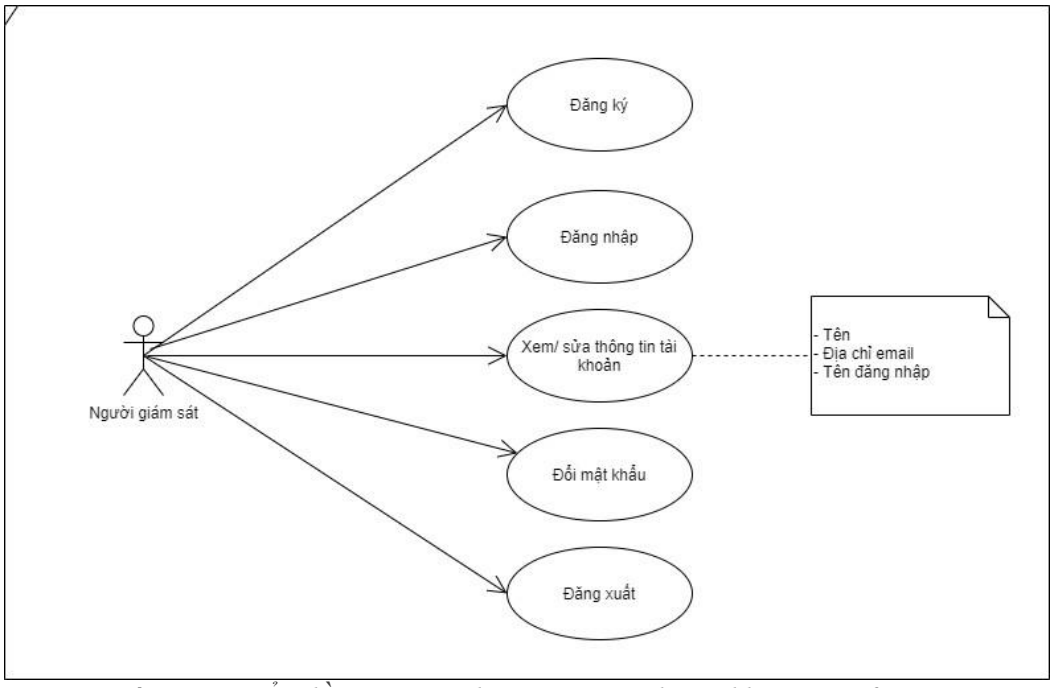
3.3 Thiết kế hệ thống

3.3.1 Biểu đồ use case tổng quan



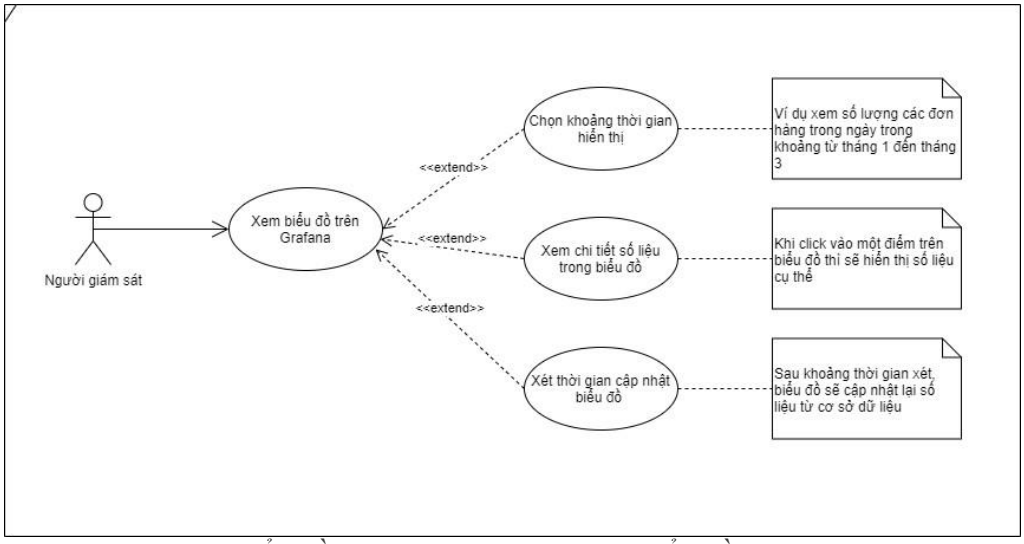
Hình 3.1 Biểu đồ use case tổng quan

3.3.2 Biểu đồ use case phân rã "quản lý tài khoản Grafana"



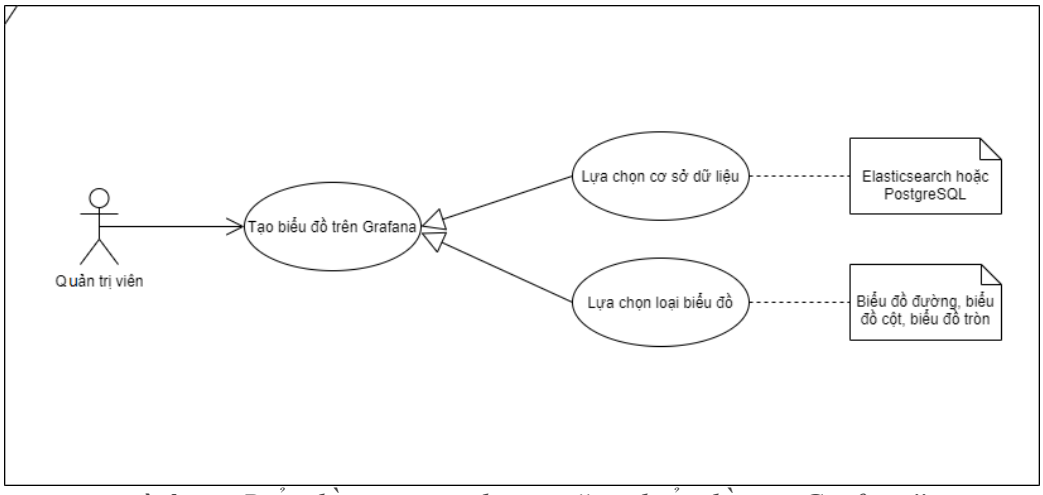
Hình 3.2 Biểu đồ use case phân rã "quản lý tài khoản Grafana"

3.3.3 Biểu đồ use case phân rã "xem biểu đồ trên Grafana"



Hình 3.3 Biểu đồ use case phân rã "xem biểu đồ trên Grafana"

3.3.4 Biểu đồ use case phân rã "tạo biểu đồ trên Grafana"



Hình 3.4 Biểu đồ use case phân rã "tạo biểu đồ trên Grafana"

3.3.5 Đặc tả use case "xem biểu đồ trên Grafana"

Mã Use case	UC01	Tên Use case	Xem biểu đồ trên Grafana
Tác nhân	Người giám sát		
Tiền điều kiện	đăng nhập Grafana thành công		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	Người giám sát	chọn dashboard scm-tracking-ml
	2	Hệ thống	kết nối tới CSDL
	3	Hệ thống	truy vấn tới CSDL để lấy dữ liệu
	4	Người giám sát	xem các đồ thị (hay panel) trong dashboard
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	2a	Hệ thống	thông báo lỗi nếu kết nối CSDL thất bại
	3a	Hệ thống	thông báo lỗi nếu câu lệnh truy lỗi

Hình 3.5 Đặc tả use case "xem biểu đồ trên grafana"

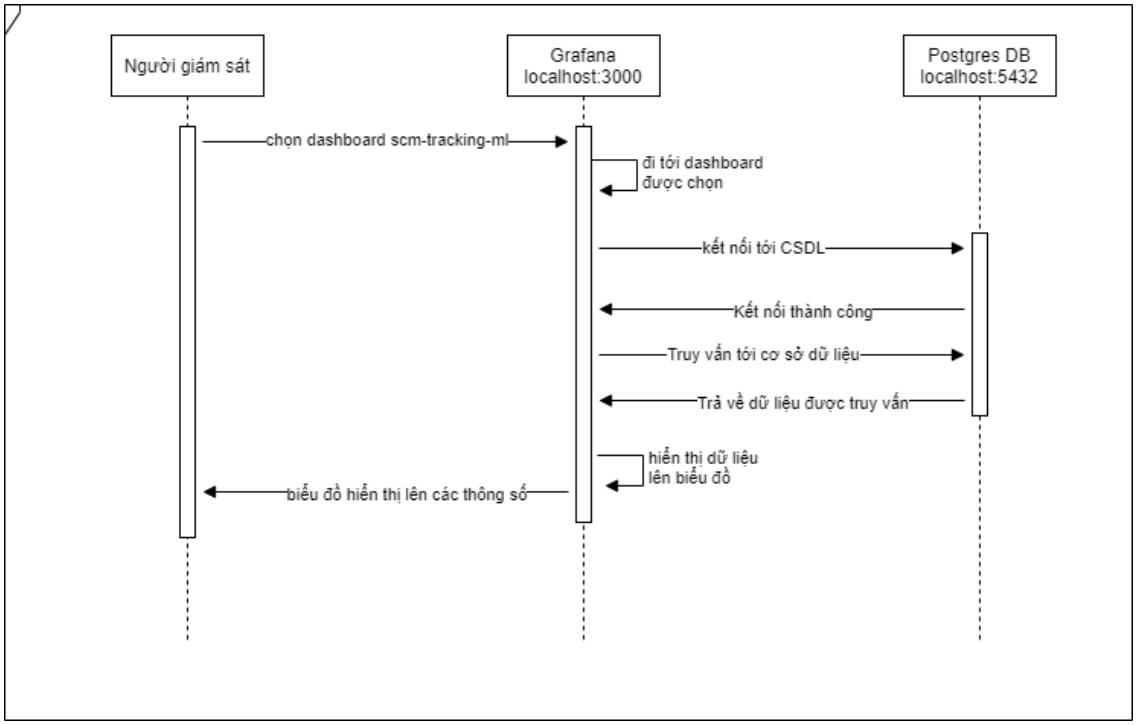
3.3.6 Đặc tả use case "tạo biểu đồ trên Grafana"

Mã Use case	UC01	Tên Use case	Tạo biểu đồ trên Grafana
Tác nhân	Quản trị viên		
Tiền điều kiện	đăng nhập Grafana thành công		
Luồng sự kiện chính	STT	Thực hiện bởi	Hành động
	1	Quản trị viên	chọn dashboard scm-tracking-ml
	2	Hệ thống	hiển thị dashboard
	3	Quản trị viên	thêm biểu đồ bằng cách kéo thả vào dashboard
	4	Hệ thống	thêm biểu đồ vào dashboard
	5	Quản trị viên	chọn vào biểu đồ để tuỳ chỉnh
	6	Hệ thống	hiển thị nội dung chỉnh sửa biểu đồ
	7	Quản trị viên	chọn cơ sở dữ liệu kết nối tới
	8	Hệ thống	kết nối tới cơ sở dữ liệu được chọn
	9	Quản trị viên	chọn loại biểu đồ phù hợp
	10	Quản trị viên	viết câu lệnh truy vấn tới cơ sở dữ liệu
	11	Hệ thống	hiển thị kết quả của lệnh truy vấn lên biểu đồ
	12	Quản trị viên	nhấn Save để lưu lại biểu đồ vừa tạo
Luồng sự kiện thay thế	STT	Thực hiện bởi	Hành động
	8a	Hệ thống	thông báo lỗi nếu kết nối CSDL thất bại
	11a	Hệ thống	thông báo lỗi nếu câu lệnh truy vấn viết không đúng

Hình 3.6 Đặc tả use case "tạo biểu đồ trên Grafana"

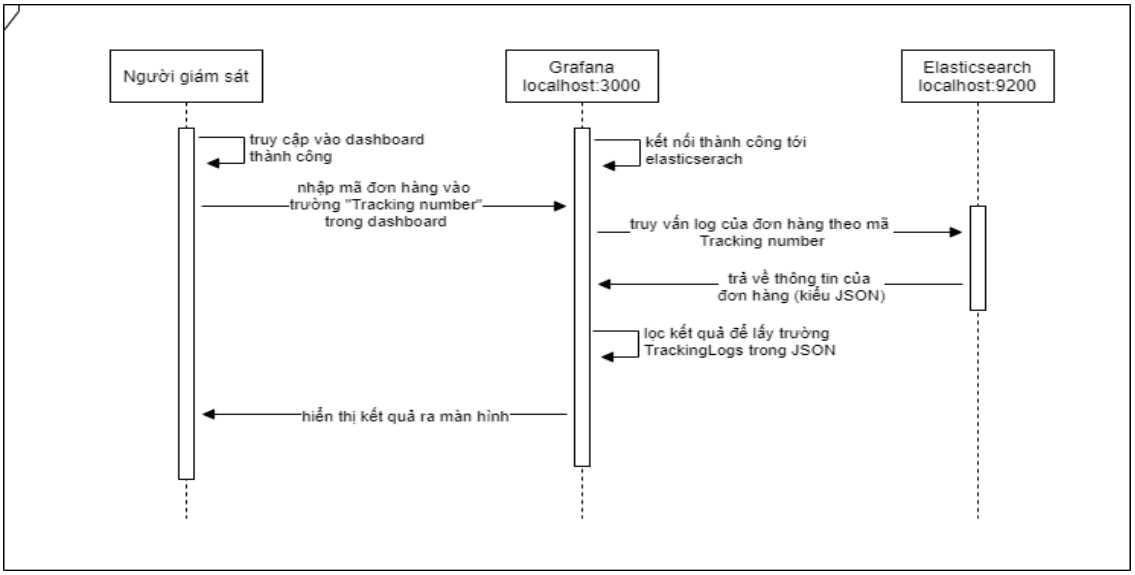
3.3.7 Biểu đồ tuần tự "xem biểu đồ trên Grafana"

Các biểu đồ trên Grafana chủ yếu lấy từ dữ liệu dạng bảng nên được truy vấn tới cơ sở dữ liệu Postgre.



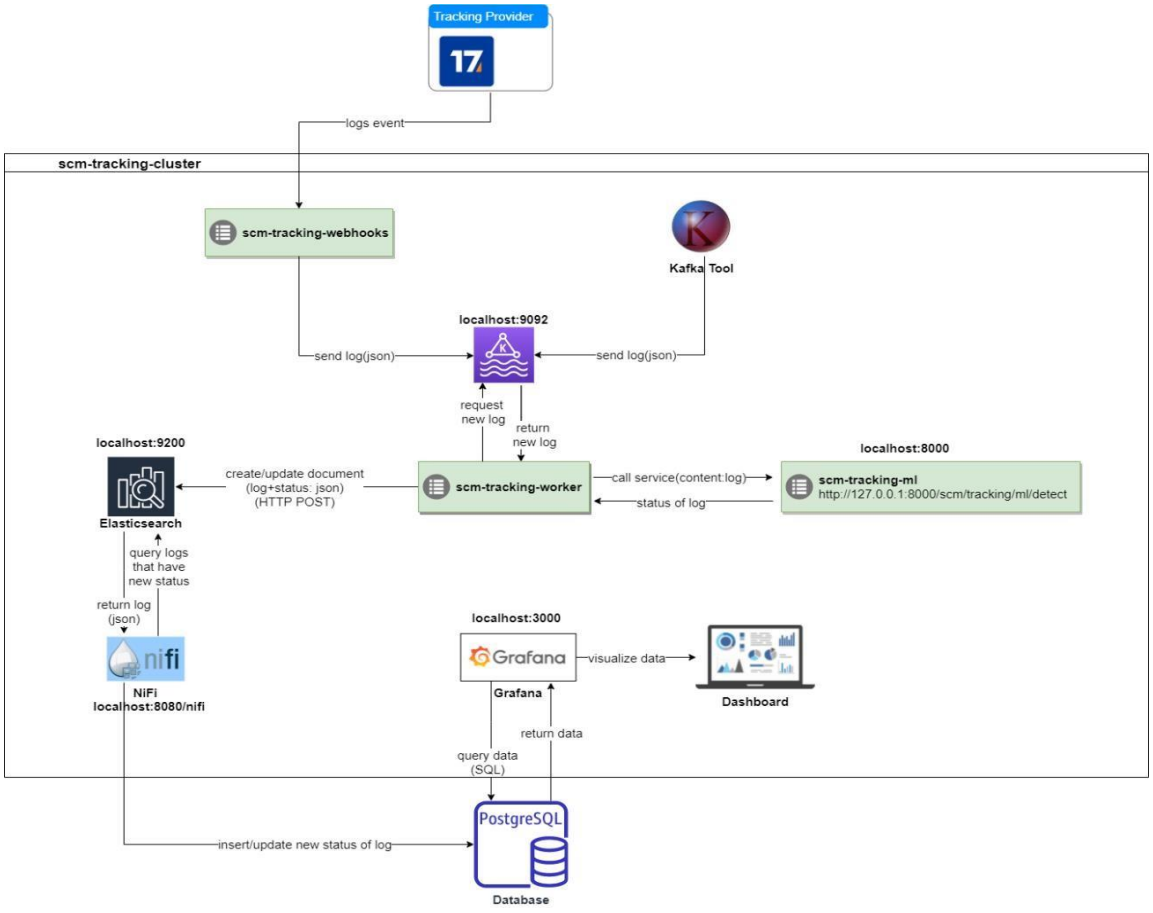
3.3.8 Biểu Đồ tuần tự "xem thông tin log của đơn hàng trên Grafana"

Thông tin log của đơn hàng có dạng JSON và được lưu trữ trong Elasticsearch. Vì vậy để hiển thị trên Grafana, quản trị viên sẽ truy vấn tới Elasticsearch.



3.4 Kiến trúc và các thành phần của hệ thống

Kiến trúc tổng quan của hệ thống được minh hoạ với hình vẽ dưới đây:



Hình 3.7 Kiến trúc hệ thống

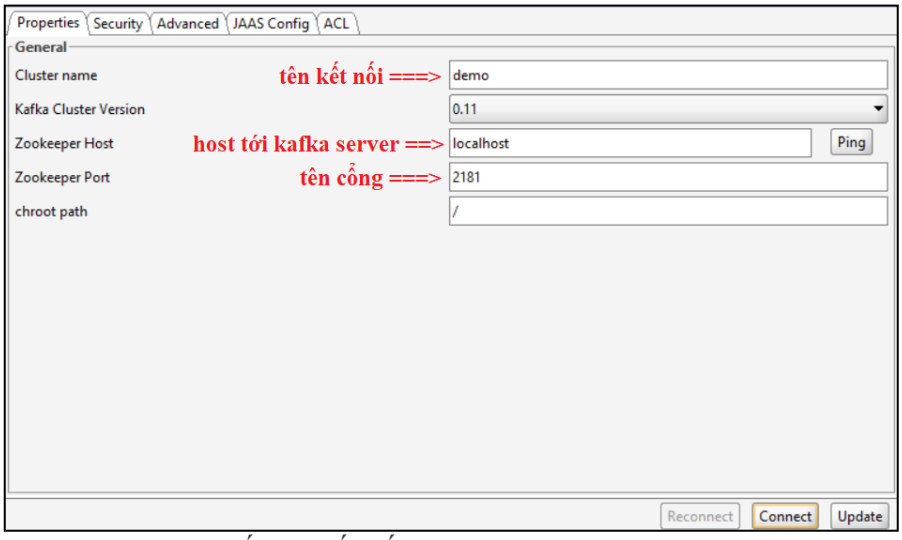
Hệ thống bao gồm các thành phần chính sau:

- Scm-tracking-webhooks
- Kafka tool
- Máy chủ Kafka
- Scm-tracking-worker
- Scm-tracking-ml
- Máy chủ Elasticsearch
- Hệ thống NiFi
- Cơ sở dữ liệu Postgre
- Máy chủ Grafana

Chi tiết về các thành phần sẽ được tác giả trình bày thành các đề mục dưới đây.

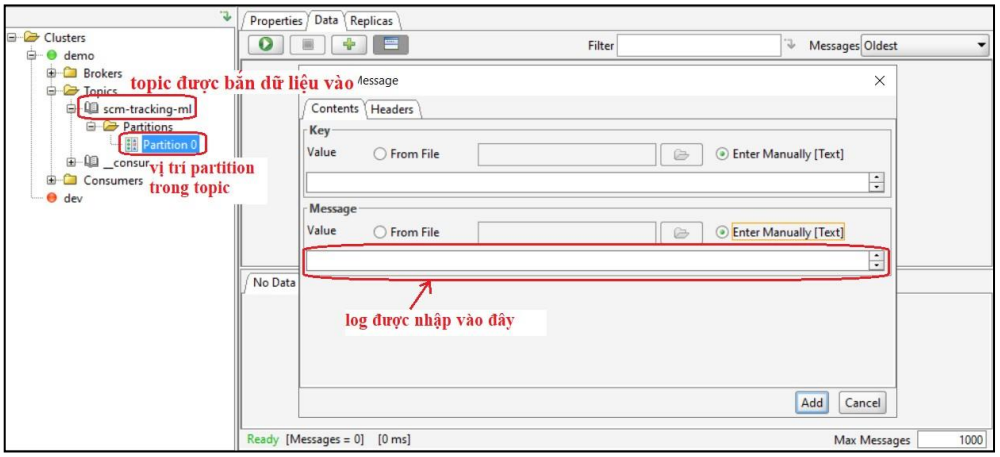
3.4.1 Kafka tool

Thay vì bắn sự kiện từ scm-tracking-webhooks tới máy chủ Kafka, mô hình sử dụng Kafka tool vì tính đơn giản và hiệu quả mà phần mềm đem lại. Đầu tiên, cần thiết lập kết nối từ Kafka tool tới máy chủ Kafka như sau:



Hình 3.8 Thiết lập kết nối từ Kafka Tool tới máy chủ Kafka

Sau khi kết nối tới Kafka thành công, truy cập vào đúng topic dùng cho lưu trữ các log của đơn hàng (scm-tracking-ml), và thực hiện ghi log lên topic:



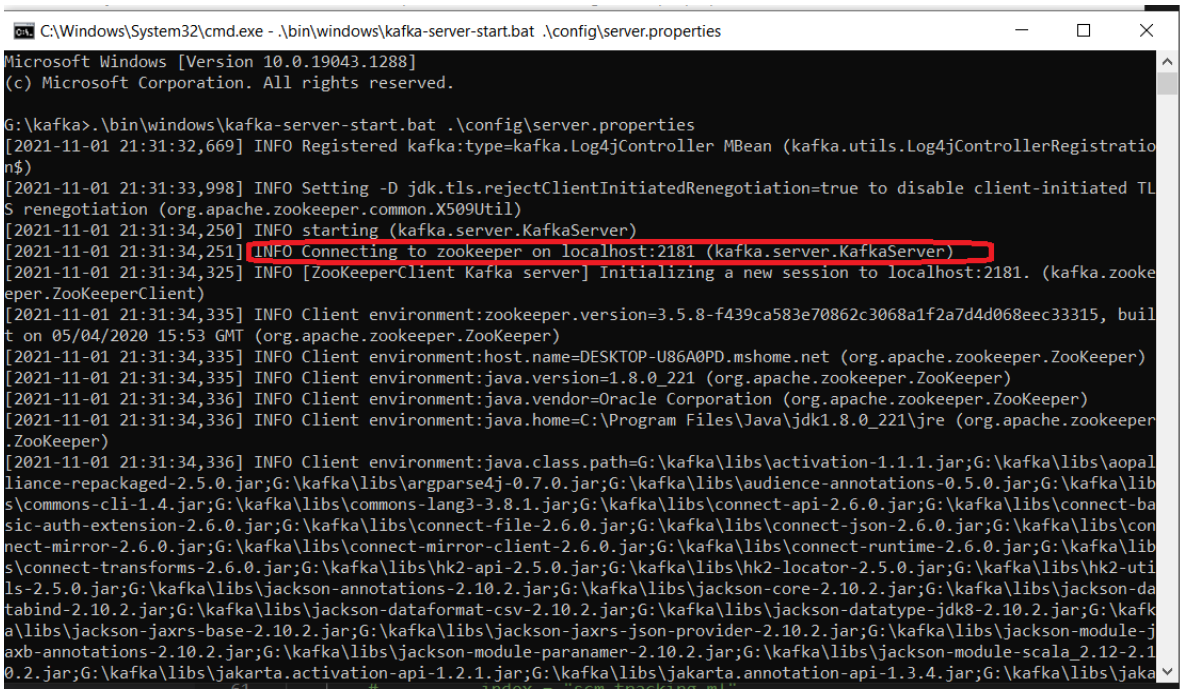
Hình 3.9 Ghi log lên Kafka

Log có kiểu dữ liệu là json với các trường thông tin sau:

```
{
  "sign": "",
  "event": "Tên sự kiện",
  "data": {
    "number": "mã tracking của đơn hàng",
    "tag": "",
    "track": {
        "z0": { // Log mới nhất của đơn hàng
            "a": "Thời gian",
            "c": "Địa điểm",
            "d": "",
            "z": "Thông tin của log"
        },
         "z1": [ // Danh sách các log của đơn hàng
            {
                "a": "Thời gian",
                "c": "Địa điểm",
                "d": "",
                "z": "Thông tin của log"
            }
          ]
        }
    }
}
```

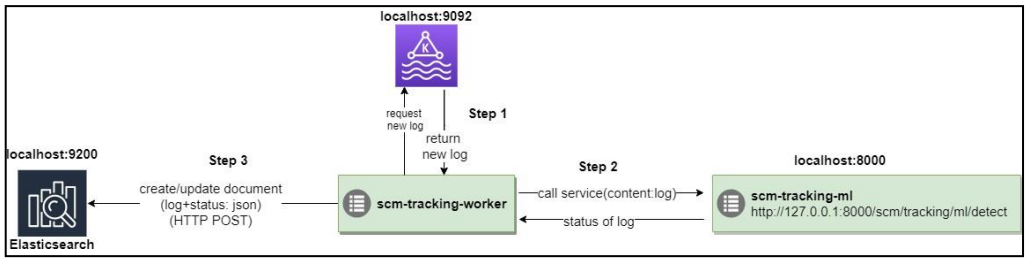
3.4.2 Máy chủ Kafka

Máy chủ Kafka được bật và đặt tại địa chỉ: localhost:2181. Máy chủ Kafka có nhiệm vụ lưu trữ log từ các nhà sản xuất (producer) tại topic scm-tracking-ml và phân phối cho các nhà tiêu thụ (consumer).



Hình 3.10 Máy chủ Kafka

3.4.3 Scm-tracking-worker



Hình 3.11 Scm-tracking-workers

Scm-tracking-worker có các công việc tuần tự như sau:

1. **Nhận log mới từ máy chủ Kafka:** scm-tracking-worker kết nối với máy chủ Kafka theo cơ chế polling, có nghĩa là sẽ tạo một vòng lặp thực hiện liên tục. Trong vòng lặp đó, bên phía consumer (scm-tracking-worker) sẽ gửi request tới máy chủ Kafka, nếu có log mới thì consumer sẽ nhận được và trả về, còn không sẽ đợi cho đến hết thời gian time-out và trả về không gì cả.

2. **Gửi log mới lên service machine learning để nhận về trạng thái của đơn hàng:** khi nhận được log mới từ máy chủ Kafka, scm-tracking- worker sẽ gửi request đến API của scm-tracking-ml: http://127.0.0.1:8000/scm/tracking/ml/detect. Nội dung của request chính là dữ liệu trả về từ consumer. Khi có dữ liệu trả về từ scm-tracking-ml, có trạng thái là 200 OK, thì sẽ bắn lên elasticsearch.

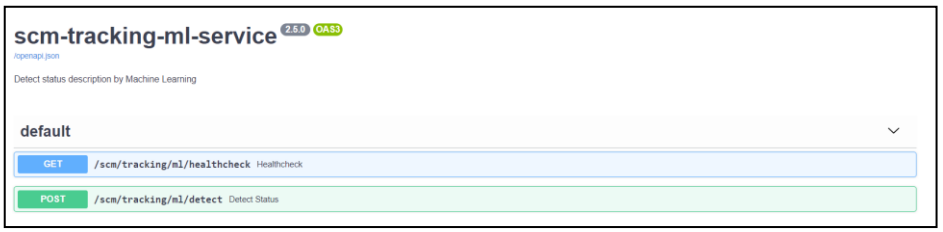
3. **Gửi log kèm theo trạng thái của đơn hàng lên máy chủ elasticsearch để lưu trữ:** scm-tracking-worker sẽ post dữ liệu dạng json lên địa chỉ của máy chủ elasticsearch: http://localhost:9200. Nội dung của dữ liệu gồm:

```
{
      "Id" : "mã tracking của đơn hàng",
      "TrackingNumber" : "mã tracking của đơn
      hàng",
      "update_time"   :   "thời   gian   cập   nhật   từ
                      worker", "last_log_time" : "thời gian của
      log mới nhất", "missing_order" : "giá trị là 0 hoặc 1"
      "lstm_score"  :  "trạng  thái  được  dự  đoán  từ  model
                   LSTM", "lstm_detail" : { // chi tiết kết quả
            "COMPLETED" : "số phần trăm",
            "DELIVERED_GUARANTEE" : "số phần
            trăm", "IN_US" : "số phần trăm",
            "RETURN_TO_SENDER" : "số phần trăm",
            "TRACKING_AVAILABLE" : "số phần
            trăm", "TRACKING_ONLINE" : "số phần
            trăm"
      },
      "bert_score" : "trạng thái được dự đoán từ model
      BERT", "bert_detail" : { // chi tiết kết quả
            COMPLETED" : "số phần trăm",
            "DELIVERED_GUARANTEE" : "số phần
            trăm", "IN_US" : "số phần trăm",
            "RETURN_TO_SENDER" : "số phần trăm",
            "TRACKING_AVAILABLE" : "số phần
            trăm", "TRACKING_ONLINE" : "số phần
            trăm"
      },
      "TrackLogs" : [ // danh sách các log của đơn hàng
            {
                  "a" : "thời gian",
                  "c" : "địa điểm",
                  "d" : "",
                  "z" : "thông tin của log"
            }
      ]
```

3.4.4 Scm-tracking-ml

Scm-tracking-ml được thiết kế để đưa ra các dự đoán của hai model LSTM và BERT để lấy trạng thái của đơn hàng và một model LSTM để dự đoán các log của đơn hàng có bị sắp xếp sai thứ tự không.

Scm-tracking-ml cung cấp các API sau:



Hình 3.12 Scm-tracking-ml

- http://127.0.0.1:8000/scm/tracking/ml/healthcheck : phương thức GET, dùng để kiểm tra xem server còn hoạt động không. Nội dung trả về:

```
{
	"hostname": "tên máy chủ",
	"status": "trạng thái",
	"timestamp": "thời gian",
	"results": []
}
```

- http://127.0.0.1:8000/scm/tracking/ml/detect : phương thức POST, trả về các kết quả dự đoán của các model. Nội dung gồm:

```
{
	"Id" : "mã tracking của đơn hàng",
	"TrackingNumber" : "mã tracking của đơn hàng",
	"missing_order" : "giá trị là 0 hoặc 1"
	"lstm_score" : "trạng thái được dự đoán từ model LSTM",
	"lstm_detail" : { // chi tiết kết quả
		"COMPLETED" : "số phần trăm",
		"DELIVERED_GUARANTEE" : "số phần trăm",
		"IN_US" : "số phần trăm",
		"RETURN_TO_SENDER" : "số phần trăm",
		"TRACKING_AVAILABLE" : "số phần trăm",
		"TRACKING_ONLINE" : "số phần trăm"
	},
	"bert_score" : " trạng thái được dự đoán từ model BERT",
	"bert_detail" : { // chi tiết kết quả
		COMPLETED" : "số phần trăm",
		"DELIVERED_GUARANTEE" : "số phần trăm",
		"IN_US" : "số phần trăm",
		"RETURN_TO_SENDER" : "số phần trăm",
		"TRACKING_AVAILABLE" : "số phần trăm",
		"TRACKING_ONLINE" : "số phần trăm"
	},
	"TrackLogs" : [ // danh sách các log của đơn hàng
		{
			"a" : "thời gian",
			"c" : "địa điểm",
			"d" : "",
			"z" : "thông tin của log"
		}
	]
}
```

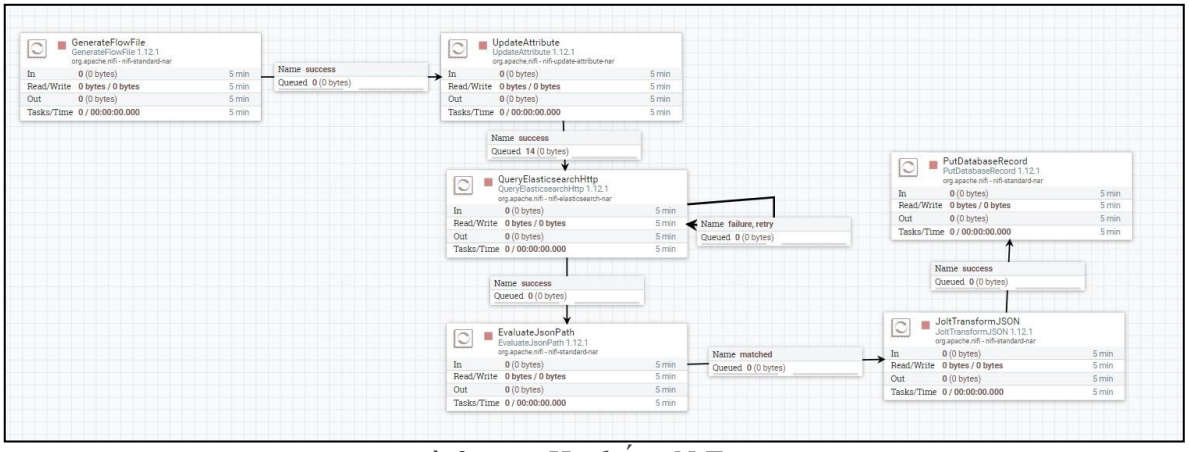
3.4.5 Máy chủ elasticsearch

Máy chủ elasticsearch tổ chức lưu trữ thông tin theo các index, trong index chưa là các document (dạng json). Các document có một id và phải là duy nhất trong index. Ở đây, trong index scm-tracking-ml, lưu trữ thông tin của các đơn hàng. Mỗi thông tin của đơn hàng được lưu được gọi là một document. Nội dung của một document gồm có:

```
{
      "_index" : "scm-tracking-ml",
      "_type" : "_doc",
      "_id" : "mã tracking của đơn hàng",
      {
            "Id" : "mã tracking của đơn hàng",
            "TrackingNumber" : "mã tracking của đơn hàng",
            "update_time" : "thời gian cập nhật từ worker",
            "last_log_time" : "thời gian của log mới nhất",
            "missing_order" : "giá trị là 0 hoặc 1"
            "lstm_score" : "trạng thái được dự đoán từ model LSTM",
            "lstm_detail" : { // chi tiết kết quả
                  "COMPLETED" : "số phần trăm",
                  "DELIVERED_GUARANTEE" : "số phần trăm",
                  "IN_US" : "số phần trăm",
                  "RETURN_TO_SENDER" : "số phần trăm",
                  "TRACKING_AVAILABLE" : "số phần trăm",
                  "TRACKING_ONLINE" : "số phần trăm"
            },
            "bert_score" : "trạng thái được dự đoán từ model BERT",
            "bert_detail" : { // chi tiết kết quả
                  COMPLETED" : "số phần trăm",
                  "DELIVERED_GUARANTEE" : "số phần trăm",
                  "IN_US" : "số phần trăm",
                  "RETURN_TO_SENDER" : "số phần trăm",
                  "TRACKING_AVAILABLE" : "số phần trăm",
                  "TRACKING_ONLINE" : "số phần trăm"
            },
            "TrackLogs" : [ // danh sách các log của đơn hàng
                  {
                        "a" : "thời gian",
                        "c" : "địa điểm",
                        "d" : "",
                        "z" : "thông tin của log"
                  }
            ]
}
```

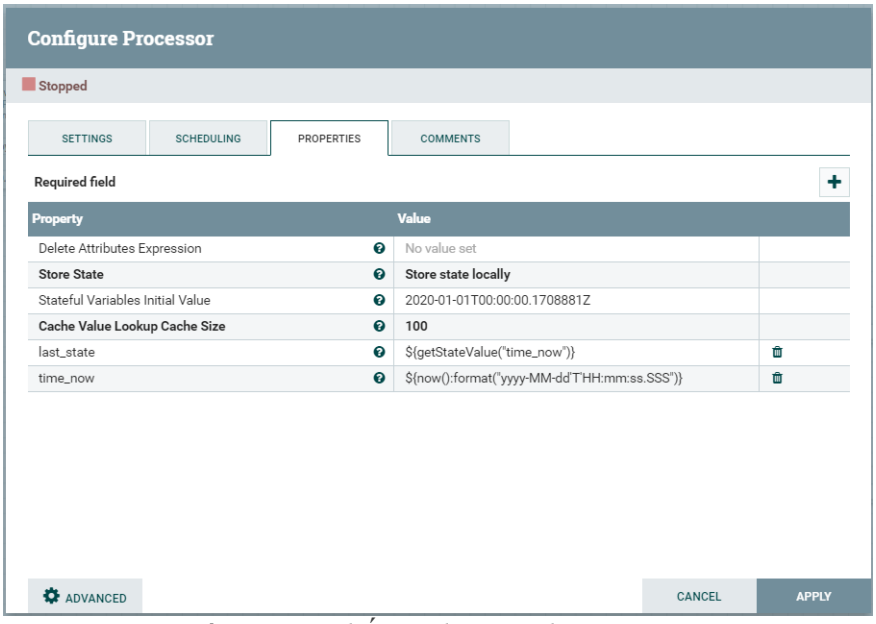
3.4.6 Hệ thống NiFi

Hệ thống NiFi có nhiệm vụ tạo một luồng chuyển dữ liệu từ elasticsearch sang cơ sở dữ liệu Postgres. Luồng dữ liệu được thiết lập chạy lại sau mỗi 5 phút.



Hình 3.13 Hệ thống NiFi

- Khối GenerateFlowFile: tạo luồng cho hệ thống
- Khối UpdateAttribute: cập nhật các biến có trong luồng.

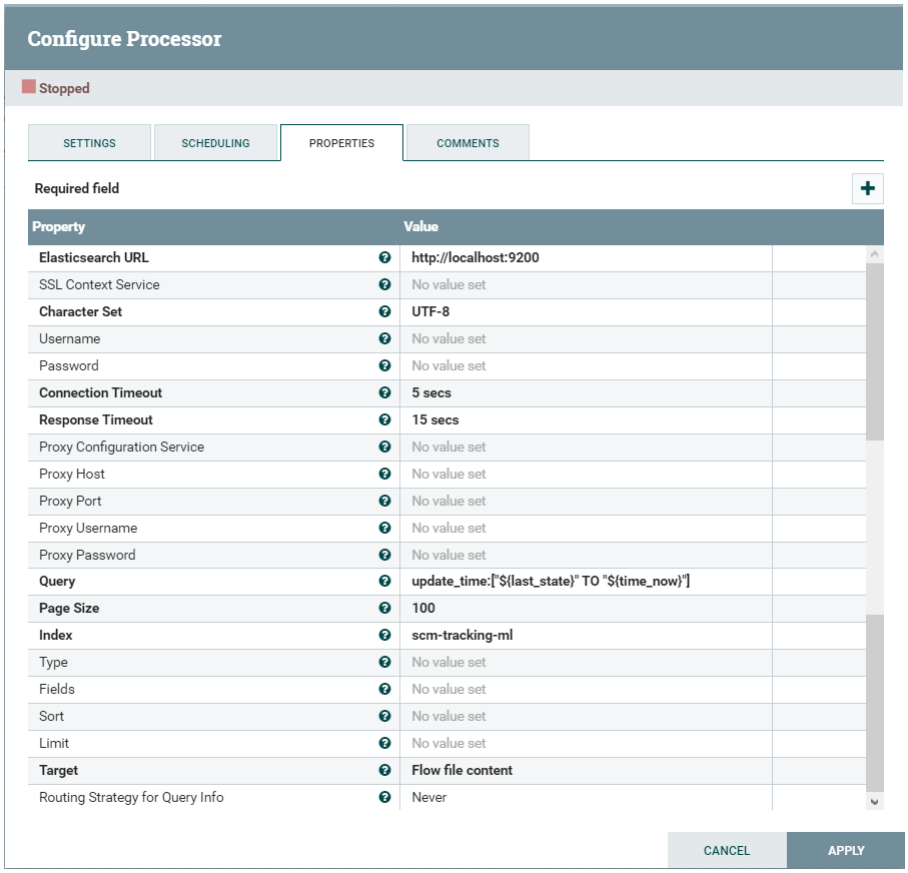


Hình 3.14 Khối UpdateAttribute

Trong đó:

- last_state: lưu trữ trạng thái gần đây nhất (cụ thể là thời gian chạy gần nhất)
- time_now: thời gian hiện tại

- Khối QueryElasticsearch: truy vấn lấy dữ liệu từ elasticsearch.

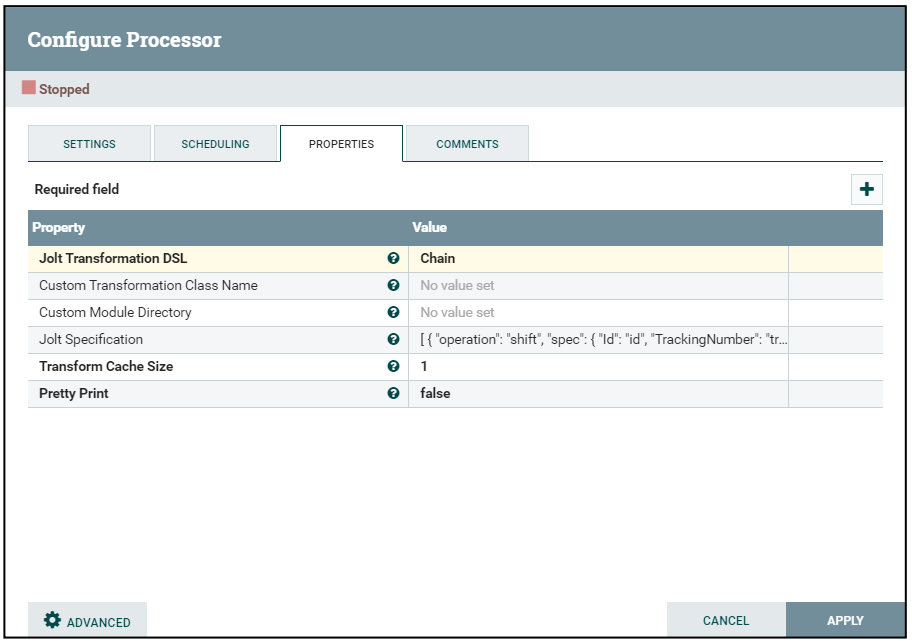


Hình 3.15 Khối QueryElasticsearchHttp

Trong đó:

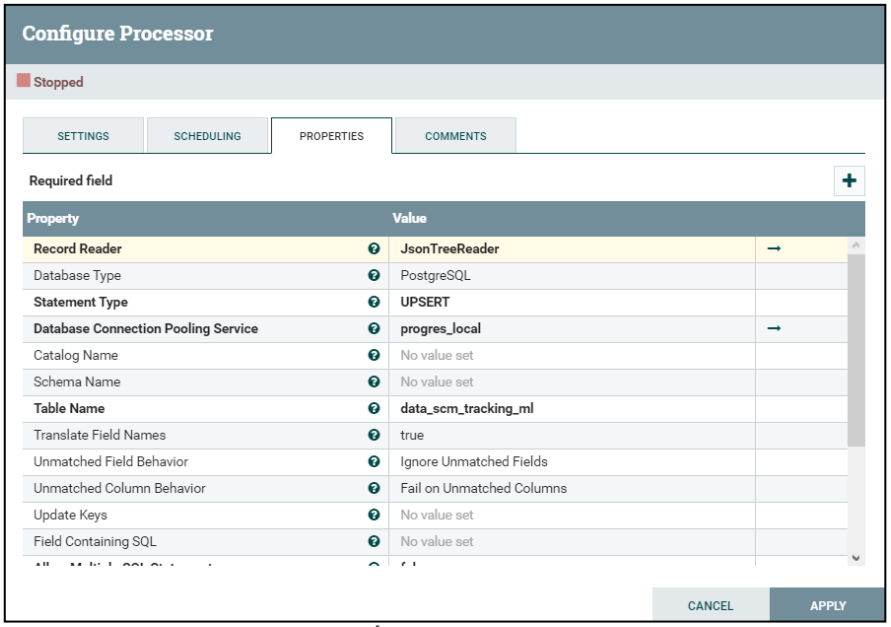
- ElasticsearchURL: đường link đến máy chủ elasticsearch
- Username: tên đăng nhập tới elasticsearch
- Password: mật khẩu tới elasticsearch
- Query: lấy những document có trường update_time nằm trong khoảng từ last_state tới time_now
- Page Size: số document trả về trong mỗi page
- Index: index trong elasticsearch

- Khối EvaluateJsonPath: kiểm tra các trường có trong đầu vào.
- Khối JoltTransfromJSON: thực hiện phép biến đổi chuỗi json đầu vào để lấy ra các trường thông tin hữu ích cho vào chuỗi json đầu ra



Hình 3.16 Khối JoltTransfromJSON

- Khối PutDatabaseRecord: update/ insert dữ liệu vào cơ sở dữ liệu Postgres



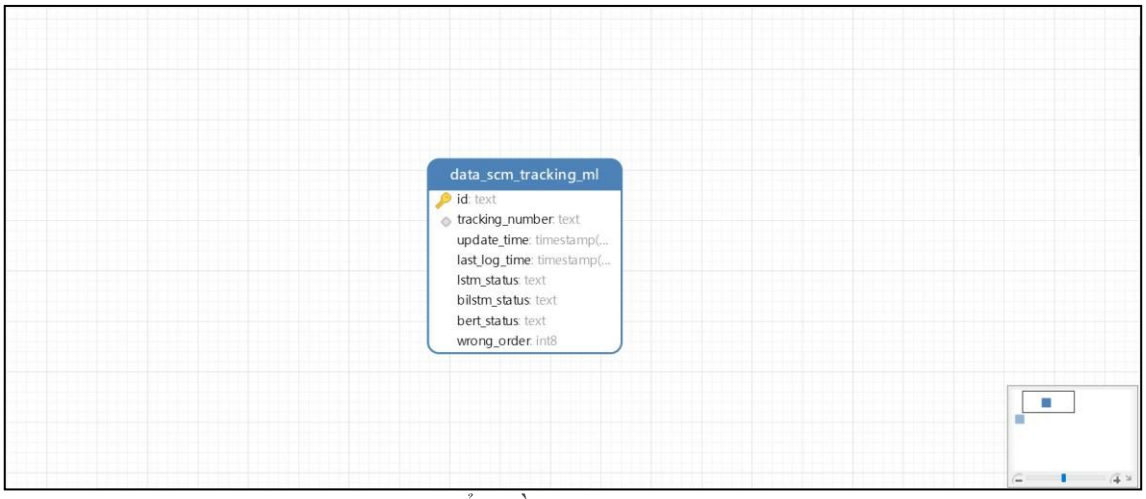
Hình 3.17 Khối PutDatabaseRecord

Trong đó:

- Record Reader: xác định kiểu dữ liệu đầu vào
- Database Type: kiểu cơ sở dữ liệu
- Statement Type: kiểu truy vấn, ở đây là UPSERT (update/insert)
- Database Connection Pooling Service: thiết lập kết nối tới cơ sở dữ liệu
- Table Name: tên bảng trong cơ sở dữ liệu

3.4.7 Cơ sở dữ liệu Postgres

Sơ đồ ERD (Entity-Relationship Diagram) cho cơ sở dữ liệu:



Hình 3.18 Biểu đồ ERD của CSDL Postgre

Cơ sở dữ liệu Postgres lưu trữ trạng thái cùng với giá trị missing_order (dùng để kiểm tra xem các log có sai thứ tự không) của đơn hàng được lưu vào bảng data_scm_tracking_ml. Cấu trúc bảng data_scm_tracking_ml như sau:

Name	Type	Length	Decimal	Not null	Key	Comment
id	text	0	0	☑	🔑 1	id của đơn hàng trong bảng
tracking_number	text	0	0	☑		mã tracking của đơn hàng
update_time	timestamp	6	0	☐		thời gian được cập nhật
last_log_time	timestamp	6	0	☐		thời gian xảy ra sự kiện mới nhất của đơn hàng
missing_order	int4	32	0	☐		kiểm tra các log có sai thứ tự không (0 hoặc 1)
lstm_status	text	0	0	☐		kết quả dự đoán của LSTM
bert_status	text	0	0	☐		kết quả dự đoán của BERT

Hình 3.19 Các trường trong bảng data_scm_tracking_ml

CSDL lưu trữ nội dung như sau:

Objects | data_scm_tracking_ml @postgres.public ...

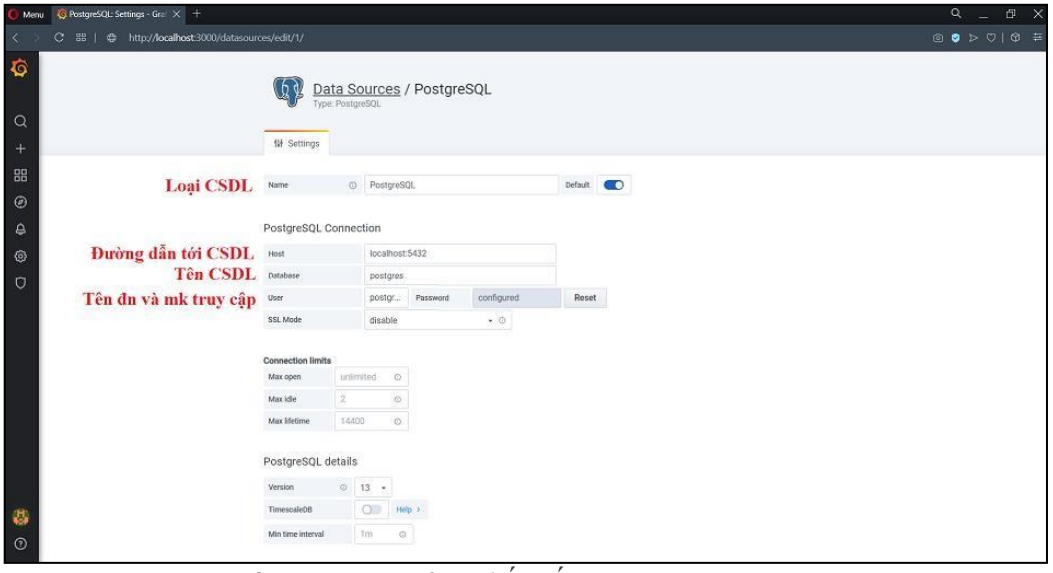
Begin Transaction | Text | Filter | Sort | Import | Export

id	tracking_number	update_time	last_log_time	missing_order	lstm_status	bilstm_status	bert_status
YT2100921272131268	YT2100921272131268	2021-01-29 14:14:00	2021-01-29 14:14:00	0	COMPLETED	COMPLETED	RETURN_TO_SENDER
YT2101621236004007	YT2101621236004007	2021-01-30 13:54:00	2021-01-30 13:54:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101921236003761	YT2101921236003761	2021-02-02 16:02:00	2021-02-02 16:02:00	1	COMPLETED	COMPLETED	RETURN_TO_SENDER
YT2100921272038747	YT2100921272038747	2021-01-26 13:39:00	2021-01-26 13:39:00	0	COMPLETED	IN_US	COMPLETED
YT2101121236000809	YT2101121236000809	2021-01-20 15:31:00	2021-01-20 15:31:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101521236002824	YT2101521236002824	2021-02-03 02:36:00	2021-02-03 02:36:00	0	IN_US	RETURN_TO_SENDER	RETURN_TO_SENDER
YT2101321236002069	YT2101321236002069	2021-01-27 10:44:00	2021-01-27 10:44:00	1	COMPLETED	COMPLETED	COMPLETED
YT2100621272139309	YT2100621272139309	2021-01-30 15:41:00	2021-01-30 15:41:00	0	COMPLETED	COMPLETED	RETURN_TO_SENDER
YT2101121236000668	YT2101121236000668	2021-01-28 13:00:00	2021-01-28 13:00:00	0	COMPLETED	COMPLETED	COMPLETED
YT2100921272130949	YT2100921272130949	2021-01-23 14:00:00	2021-01-23 14:00:00	0	COMPLETED	COMPLETED	COMPLETED
YT2100921272130541	YT2100921272130541	2021-01-23 11:02:00	2021-01-23 11:02:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101621272028001	YT2101621272028001	2021-01-30 15:02:00	2021-01-30 15:02:00	0	COMPLETED	RETURN_TO_SENDER	COMPLETED
YT2101721236001721	YT2101721236001721	2021-02-02 13:52:00	2021-02-02 13:52:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101221236001927	YT2101221236001927	2021-01-28 11:30:00	2021-01-28 11:30:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101721236000301	YT2101721236000301	2021-01-29 11:27:00	2021-01-29 11:27:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101721236000186	YT2101721236000186	2021-01-30 13:50:00	2021-01-30 13:50:00	1	COMPLETED	COMPLETED	COMPLETED
YT2100821272041448	YT2100821272041448	2021-01-22 10:08:00	2021-01-22 10:08:00	0	COMPLETED	COMPLETED	COMPLETED
YT2100721272135668	YT2100721272135668	2021-01-25 11:34:00	2021-01-25 11:34:00	0	COMPLETED	RETURN_TO_SENDER	COMPLETED
YT2100521272144096	YT2100521272144096	2021-01-19 18:49:00	2021-01-19 18:49:00	1	COMPLETED	COMPLETED	COMPLETED
YT2100921272026563	YT2100921272026563	2021-01-26 10:03:00	2021-01-26 10:03:00	0	COMPLETED	COMPLETED	COMPLETED
YT2100721272135941	YT2100721272135941	2021-01-21 16:42:00	2021-01-21 16:42:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101421236003919	YT2101421236003919	2021-01-25 14:57:00	2021-01-25 14:57:00	1	COMPLETED	COMPLETED	COMPLETED
YT2036321272026761	YT2036321272026761	2021-01-20 16:32:00	2021-01-20 16:32:00	1	COMPLETED	COMPLETED	RETURN_TO_SENDER
YT2100721272135997	YT2100721272135997	2021-01-25 13:23:00	2021-01-25 13:23:00	0	COMPLETED	COMPLETED	COMPLETED
YT2101221236003609	YT2101221236003609	2021-01-29 15:11:00	2021-01-29 15:11:00	1	COMPLETED	COMPLETED	IN_US
YT2101321272034504	YT2101321272034504	2021-01-29 11:26:00	2021-01-29 11:26:00	1	COMPLETED	COMPLETED	RETURN_TO_SENDER
YT2101321236000172	YT2101321236000172	2021-01-30 14:30:00	2021-01-30 14:30:00	0	COMPLETED	COMPLETED	COMPLETED
YT2103521272029230	YT2103521272029230	2021-01-02 03:39:40.478631	2021-01-02 03:39:40.478631	0	COMPLETED	(Null)	IN_US
YT2101721236001857	YT2101721236001857	2021-02-02 13:22:00	2021-02-02 13:22:00	0	COMPLETED	COMPLETED	RETURN_TO_SENDER
YT2102621272032706	YT2102621272032706	2021-01-02 04:18:38.631133	2021-01-02 04:18:38.631133	0	COMPLETED	(Null)	COMPLETED

Hình 3.20 CSDL Postgres

3.4.8 Máy chủ Grafana

Grafana có nhiệm vụ lấy dữ liệu từ Posgres nhằm vẽ các đồ thị để có thể giám sát.



Hình 3.21 Grafana kết nối tới CSDL Postures

Sau khi truy cập vào CSDL thành công, có thể viết các câu lệnh truy vấn để lấy ra dữ liệu làm biểu đồ:

Hình 3.22 Giao diện vẽ biểu đồ trên Grafana

CHƯƠNG 4. XÂY DỰNG MÔ HÌNH HỌC MÁY

4.1 Thu thập dữ liệu để huấn luyện

Trước hết, ta hiểu 17Track là gì? 17Track là một trong những nền tảng dịch vụ theo dõi lô hàng của bên thứ ba trên toàn thế giới. 17TRack được thiết kế cho các nhà bán lẻ điện tử và khách hàng cũng như được sử dụng để theo dõi lô hàng toàn cầu, đặc biệt là đối với các đơn hàng Thương mại điện tử được vận chuyển từ Trung Quốc ra nước ngoài (đó cũng chính là lý do mà một số logs đầu vào trong đồ án em có tiếng Trung). 17Track cung cấp một số phân tích dữ liệu trên một số phần của trang web. Các trang này cung cấp thông tin cơ bản về các mô hình vận chuyển phổ biến, bao gồm cả quốc gia nào đang vận chuyển số lượng bưu kiện nhiều nhất và năm và tháng tương ứng với thời điểm xuất phát các số liệu thống kê đó. Thông tin này đến từ dữ liệu theo dõi 17Track.

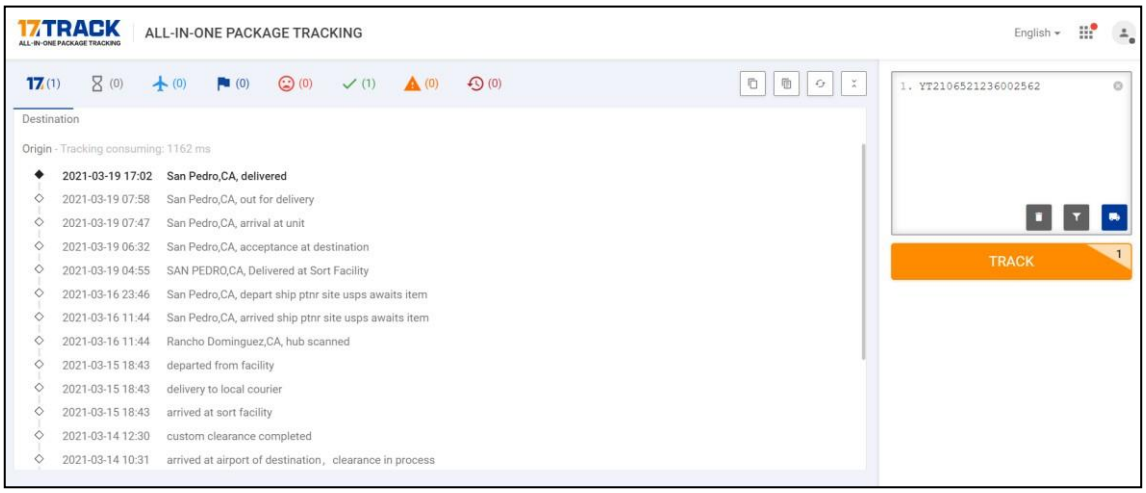
Từ những đơn hàng đã có trong hệ thống, thực hiện thu thập các mã của đơn hàng lại (các tracking_number), sau đó thực hiện cào dữ liệu qua API của nhà cung cấp 17Track để lấy thông tin các sự kiện của đơn hàng. Quá trình được thực hiện như sau:

Ban đầu, tập dữ liệu có khoảng 10 000 mã tracking_ number:

```
data["tracking_number"]

0       YT2101621236003930
1       YT2101621236003930
2       YT2101621236003930
3       YT2101621236003930
4       YT2101621236003930
              ...
9956    YT2101621272026865
9957    YT2101621272026865
9958    YT2101621272026865
9959    YT2101621272026865
9960    YT2101621272026865
Name: tracking_number, Length: 9961, dtype: object
```

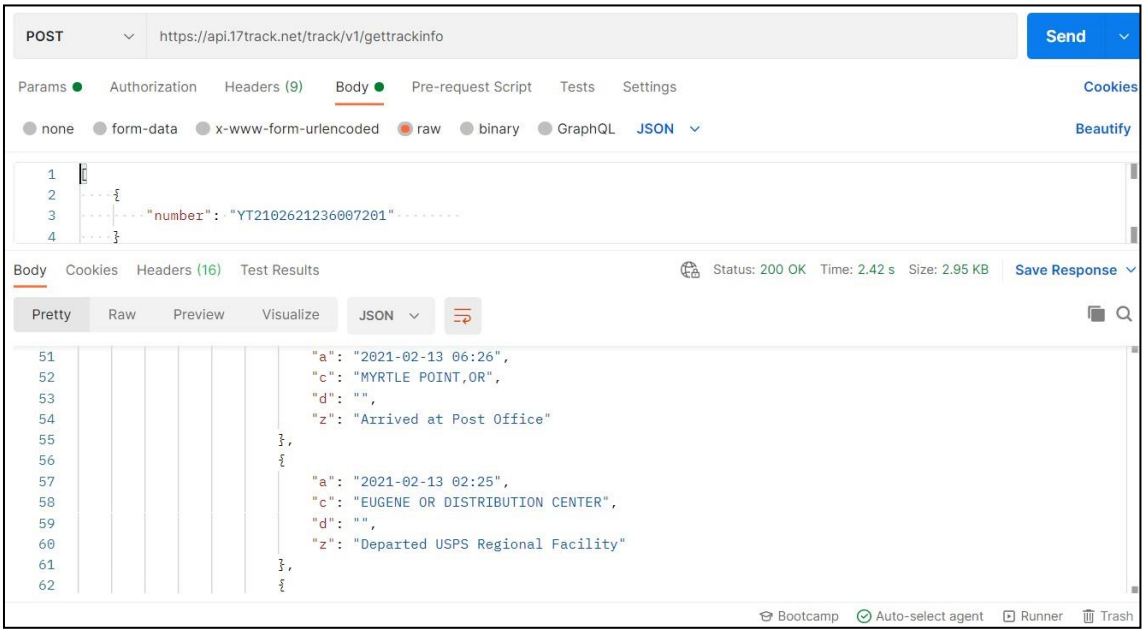
Hình 4.1 Danh sách tracking_number

Thông tin các sự kiện của đơn hàng từ 17Track như sau:



Hình 4.2 Thông tin đơn hàng từ 17Track

Sau khi đăng ký dịch vụ của 17Track, dịch vụ sẽ được cung cấp một API để lấy dữ liệu cùng với một mã token để xác thực người đăng ký:



Hình 4.3 Lấy thông tin của đơn hàng qua API

4.2 Đánh giá dữ liệu đầu vào

Ta sử dụng thư viện pandas của python để đọc dữ liệu huấn luyện (dữ liệu có dạng là DataFrame):

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9961 entries, 0 to 9960
Data columns (total 3 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   tracking_number     9961 non-null   object
 1   checkpoint_status   9961 non-null   object
 2   status_description  9961 non-null   object
dtypes: object(3)
memory usage: 233.6+ KB
```

Hình 4.4 Thông tin các trường của tập dữ liệu

	tracking_number	checkpoint_status	status_description
9956	YT2101621272026865	IN_US	ZHENGZHOU Arrived at Sort Facility ZHENGZHOU ...
9957	YT2101621272026865	IN_US	ZHENGZHOU Arrived at Sort Facility ZHENGZHOU ...
9958	YT2101621272026865	IN_US	ZHENGZHOU Arrived at Sort Facility ZHENGZHOU ...
9959	YT2101621272026865	IN_US	ZHENGZHOU Arrived at Sort Facility ZHENGZHOU ...
9960	YT2101621272026865	COMPLETED	ZHENGZHOU Arrived at Sort Facility ZHENGZHOU ...

Hình 4.5 Dữ liệu huấn luyện

Ta có thể thấy, tập dữ liệu chứa 9961 logs và có 3 trường:

- tracking_number: mã tracking của đơn hàng
- checkpoint_status: trạng thái của đơn hàng được gắn nhãn để huấn luyện
- status_description: nội dung chứa các log của đơn hàng

Sau đó, sẽ thực hiện kiểm tra xem trong tập dữ liệu có ô nào có giá trị null không, nếu có sẽ xoá cả hàng dữ liệu đó bởi vì giá trị null sẽ gây lỗi trong quá trình huấn luyện:

```
tracking_number        0
checkpoint_status      0
status_description     0
check_log              0
dtype: int64

total_missing: 0
total_cells: 39844
percent_missing: 0.000%
```

Hình 4.6 Kiểm tra dữ liệu huấn luyện

Sau khi kiểm tra xong, nhóm các tracking_number theo trạng thái để quan sát:

checkpoint_status	tracking_number	status_description	check_log
COMPLETED	168	168	168
DELIVERED_GUARANTEE	1	1	1
IN_US	6357	6357	6357
RETURN_TO_SENDER	670	670	670
TRACKING_AVAILABLE	349	349	349
TRACKING_ONLINE	2416	2416	2416

Hình 4.7 Nhóm các tracking number theo trạng thái

Thêm vào đó, thực hiện chia tập dữ liệu thành hai tập: huấn luyện (train) và đánh giá (validation) theo tỉ lệ 9:1. Để thực hiện, cần sử dụng hàm train_test_split trong thư viện sklearn như sau:

```
[ ]  1 from sklearn.model_selection import train_test_split
     2
     3 X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.10, random_state = 42)
     4 # X_train = X
     5 # Y_train = Y
     6
     7 print(X_train.shape,Y_train.shape)
     8 print(X_test.shape,Y_test.shape)

    (8964, 250) (8964, 2)
    (997, 250) (997, 2)
```

Hình 4.8 Chia tập dữ liệu

4.3 Mô hình LSTM để trích xuất trạng thái đơn hàng Tiền xử lý dữ liệu (preprocessing data)

4.3.1 Sau khi có được dữ liệu đầu vào, thực hiện tiền xử lý để làm sạch dữ liệu trong

cột status_description (cột này chứa nội dung các log của đơn hàng). Các bước của quá trình tiền xử lý lần lượt là:

1. Lower: đưa toàn bộ nội dung về dạng chữ thường (bỏ hết chữ cái in hoa)
2. Xoá hết các ký tự đặc biệt trong văn bản như: [] \ / | @ #;
3. Chuyển hết các dấu chấm câu thành “<EOS>” (end of sentence)
4. Xoá kết các stopwords: các stopword là các từ xuất hiện nhiều trong tiếng Anh, chúng không mang nhiều ý nghĩa, như: a, an, the, of, in, on, out, …
5. Lemmatization: đưa các từ trong câu về dạng nguyên thể, ví dụ như: rocks => rock, better => good

4.3.2 Mã hoá từ (word tokenization)

Qua bước tiền xử lý, tiếp tục tiến hành mã hóa dữ liệu. Thực hiện xây dựng một bộ từ điển chứa tất cả các từ vựng trong dữ liệu (vocabulary):

```
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

all_words = [word for tokens in clean_questions["tokens"] for word in tokens]
sentence_lengths = [len(tokens) for tokens in clean_questions["tokens"]]
VOCAB = sorted(list(set(all_words)))

print('Words total: {}'.format(len(all_words)))
print('Vocabulary size: {}'.format(len(VOCAB)))
print('Max sentence length is {}'.format(max(sentence_lengths)) )
```

```
Words total: 772885
Vocabulary size: 759
Max sentence length is 361
```

Hình 4.9 Bộ từ điển dữ liệu

Kết quả của đoạn code cho thấy: bộ dữ liệu có tổng là 772855 từ, trong đó có 759 từ vựng và log dài nhất trong dữ liệu có 361 từ. Tiếp đó, phải giới hạn số lượng từ trong một log xuống còn 250, vì với số từ dài hơn, kích thước của mô hình sẽ rất lớn và việc huấn luyện trở nên rất lâu. Vị vậy, sẽ lấy nội dung từ cuối lên đầu trong log để không vượt quá 250 từ.

```
clean_questions["sentence_length"].describe()
```

```
count    9961.000000
mean       62.872704
std        38.617841
min         1.000000
25%        29.000000
50%        61.000000
75%        96.000000
max       250.000000
Name: sentence_length, dtype: float64
```

Hình 4.10 Sau khi giới hạn kích thước log

Kết quả cho thấy: log dài nhất có đọ dài 250 từ, 75% log có đọ dài là 96 từ. Sau đó sẽ thêm một thành phần vào cuối bộ từ điển là OOV (out of vocab), những từ không có trong từ điển sẽ được sẽ được coi là thành phần này.

Sau khi xây dựng xong bộ từ điển, tiến hành mã hoá dữ liệu. Các log trong dữ liệu sẽ được mã hoá thành các vector số, mỗi phần tử trong vector sẽ đại diện cho vị trí ứng với từ trong từ điển. Ví dụ: câu “I am hungry” sẽ được mã hoá thành vector [5, 3, 4], với 5 là vị trí của từ “I” trong từ điển.

4.3.3 Biểu diễn câu (word embedding)

Ta sử dụng bộ dữ liệu được huấn luyện rồi của GolVe, khi tải về bộ dữ liệu của GolVe sẽ có các lựa chọn: bộ dữ liệu huấn luyện từ 50 chiều, 100 chiều, 200 chiều và 300 chiều. Ở đây, mô hình sử dụng bộ 100 chiều. Tại bước này, tạo một ma trận

embedding chứa 100 đặc trưng của mỗi từ trong từ điển. Với bộ từ điển chứa 761 từ, thì kích cỡ của ma trận embedding là (761, 100), những từ trong từ điển không có trong bộ của GloVe sẽ cho vector biểu diễn của từ đó bằng 0.

4.3.4 Mô hình huấn luyện

Tại bước này, tiến hành xây dựng các tầng cho mô hình huấn luyện. Đầu tiên, khai báo mô hình với kiểu là sequence model:

```
model = Sequential()
```

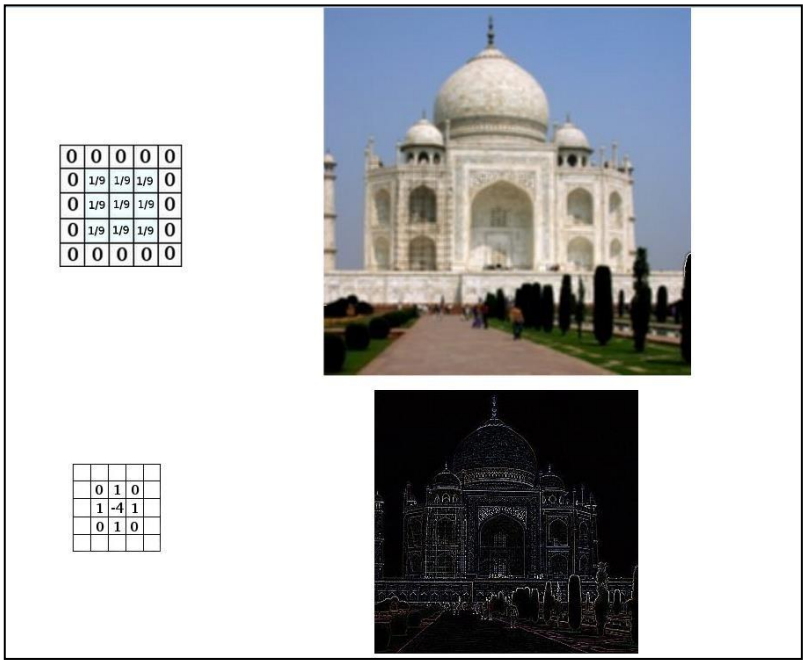
Sau đó, thêm tầng embedding vào mô hình:

```
model.add(Embedding(
      input_dim=VOCAB_SIZE,
      output_dim=EMBEDDING_DIM,
      weights=[embedding_matrix],
      input_length=MAX_SEQUENCE_LENGTH,
      trainable=True))
```

Trong đó:

- Input_dim: kích cỡ của từ điển
- Output_dim: số chiều của ma trận embedding
- Weights: trọng số là ma trận embedding
- Intput_length: độ dài của câu

Em áp dụng mô hình mạng CNN cho hai tầng tiếp theo, mạng CNN giúp cho mô hình có thể thu được các thông tin quan trọng của văn bản đầu vào và bỏ qua những



thông tin không cần thiết. Cùng xem ví dụ sau:

Hình 4.11 Ảnh được áp dụng mạng CNN

Khi áp dụng mạng CNN các đặc trưng chính của của ảnh được giữ lại, điều này cũng có hiệu quả với văn bản chữ.

Hai tầng được thêm lần lượt là: Tầng Conv1D:

```
model.add(Conv1D(
        filters=32,
        kernel_size=3,
        padding='same',
        activation='relu'))
```

Trong đó:

- Filters: số lượng lớp filter
- Kernel: kích cỡ lớp filter, ở đây là 3x3
- Padding: loại padding được áp dụng, ở đây là same, kích cỡ của ma trận đầu ra bằng với kích cỡ ma trận đầu vào khi qua lớp filter
- Activation: hàm kích hoạt được dùng

Tầng MaxPooling1D:

```
model.add(MaxPooling1D(pool_size=2))
```

Trong đó:

- Pool_size: kích cỡ của cửa sổ MaxPooling

Tầng tiếp theo được thêm vào mô hình là LSTM:

```
model.add(LSTM(
        EMBEDDING_DIM,
        dropout=0.2,
        recurrent_dropout=0.2))
```

Trong đó:

- Units: (=Embedding_DIM) số chiều của không gian đầu ra
- Dropout: có giá trị từ 0 đến 1, các giá trị trong ma trận mà nhỏ hơn giá trị được xét sẽ bằng 0 ((mặc định là 0))
- Recurrent_dropout: giá trị giữa 0 và 1, một phần nhỏ của các đơn vị cần giảm cho sự biến đổi tuyến tính của trạng thái hồi quy (mặc định là 0)

Tầng kế tiếp trong mô hình là Dense:

```
model.add(Dense(
        units=6,
        activation='softmax'))
```

Trong đó:

- Units: số chiều của không gian đầu ra
- Activation: hàm kích hoạt, nếu không được khai báo thì hàm kích hoạt sẽ là a(x) = x

Trong quá trình huấn luyện, thiết lập thêm EarlyStopping, điều này giúp cho trong quá trình huấn luyện nếu độ chính xác không cải thiện qua nhiều lần huấn luyện thì quá trình sẽ dừng lại, và mô hình tốt nhất sẽ được lưu lại.

```
callbacks = [EarlyStopping(
                monitor='val_loss',
                verbose=1,
                patience=5),
            ModelCheckpoint(
                filepath=bestmodel_filepath,
                monitor='val_loss',
                save_best_only=True)]
```

Tổng quan về mô hình LSTM:

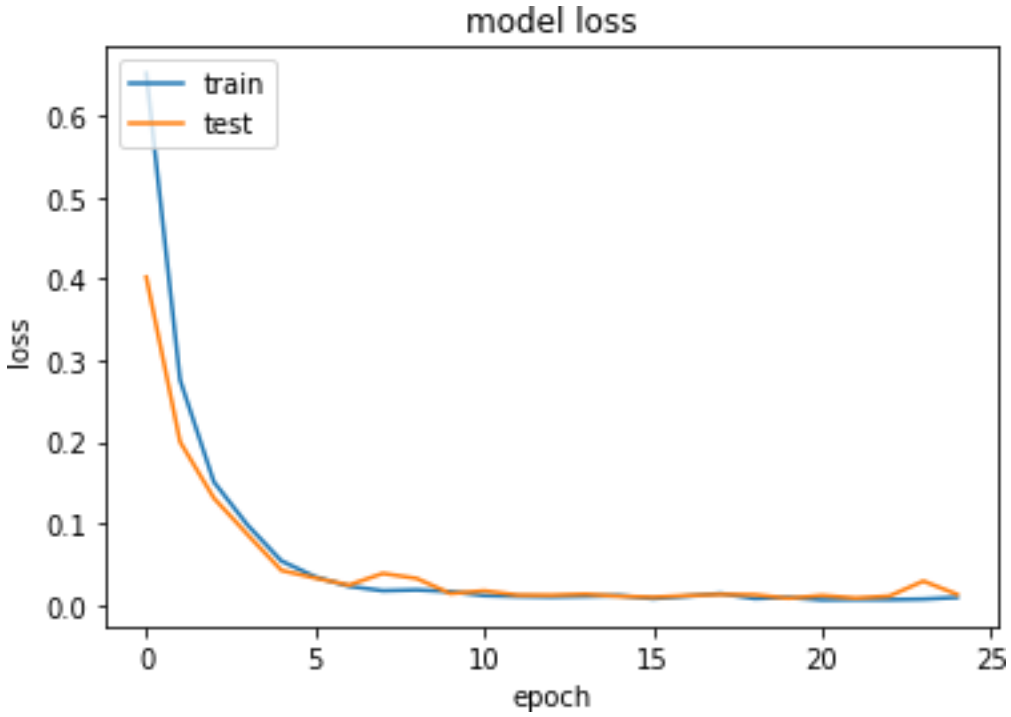
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 250, 100)          76100
_________________________________________________________________
conv1d (Conv1D)              (None, 250, 32)           9632
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 125, 32)           0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense (Dense)                (None, 2)                 202
=================================================================
Total params: 139,134
Trainable params: 139,134
Non-trainable params: 0
```

Hình 4.12 Tổng quan về mô hình LSTM

Số lượng tham số của cả mô hình là 139134 và các tham số này đều được cập nhật trong quá trình huấn luyện.

4.3.5 Kết quả huấn luyện mô hình

Hình dưới đây mô tả độ biến thiên của hàm Loss sau 25 epoch:



Hình 4.13 Độ biến thiên của hàm Loss trong quá trình huấn luyện

Ta có thể thấy giá trị của hàm Loss giảm dần qua các epoch, và sau 25 epoch quá trình huấn luyện dừng lại vì độ chính xác của mô hình không được cải thiện (sử dụng Early stopping). Độ chính xác của mô hình trên tập huấn luyện đạt được là 99.33%.

4.4 Mô hình BERT để trích xuất trạng thái đơn hàng

Để sử dụng mô hình BERT, giải pháp đã sử dụng thư viện transformers của HuggingFace được công bố Github, thư viện này cung cấp các mô hình BERT đã được huấn luyện để giải quyết một số bài toán trong xử lý ngôn ngữ tự nhiên như: phân loại câu (sequence classification), nhận dạng thực thể có tên (named - entity recognization), mô hình ngôn ngữ (language modeling), trả lời câu hỏi (exactive question awsering), …

Mô hình áp dụng cho bài toán của mình là phân loại câu (sequence classification). Để sử dụng được thư viện cần phải tải về bằng câu lệnh sau:

```
!pip install transformers

Collecting transformers
  Downloading https://files.pythonhosted.org/packages/d8/b2/57495b5309f09fa501866e225c84532d1fd89536ea62406b2181933fb418/transformers-4.5.1-py3-none-any.whl (2.1MB)
     |████████████████████████████████| 2.1MB 8.3MB/s
Collecting tokenizers<0.11,>=0.10.1
  Downloading https://files.pythonhosted.org/packages/ae/04/5b870f26a858552025a62f1649c20d29d2672c02ff3c3fb4c688ca46467a/tokenizers-0.10.2-cp37-cp37m-manylinux2010_x86_64.whl (3.3MB)
     |████████████████████████████████| 3.3MB 39.0MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.19.5)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.41.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2019.12.20)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from transformers) (3.10.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from transformers) (20.9)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.0.12)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.0)
Collecting sacremoses
  Downloading https://files.pythonhosted.org/packages/75/ee/67241dc87f266093c533a2d4d3d69438e57d7a90abb216fa076e7d475d4a/sacremoses-0.0.45-py3-none-any.whl (895kB)
     |████████████████████████████████| 901kB 50.6MB/s
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata; python_version < "3.8"->transformers) (3.4.1)
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from importlib-metadata; python_version < "3.8"->transformers) (3.7.4.3)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->transformers) (2.4.7)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2020.12.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2.10)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers) (1.0.1)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers) (7.1.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers) (1.15.0)
Installing collected packages: tokenizers, sacremoses, transformers
Successfully installed sacremoses-0.0.45 tokenizers-0.10.2 transformers-4.5.1
```

Hình 4.14 câu lệnh tải thư viện transformers

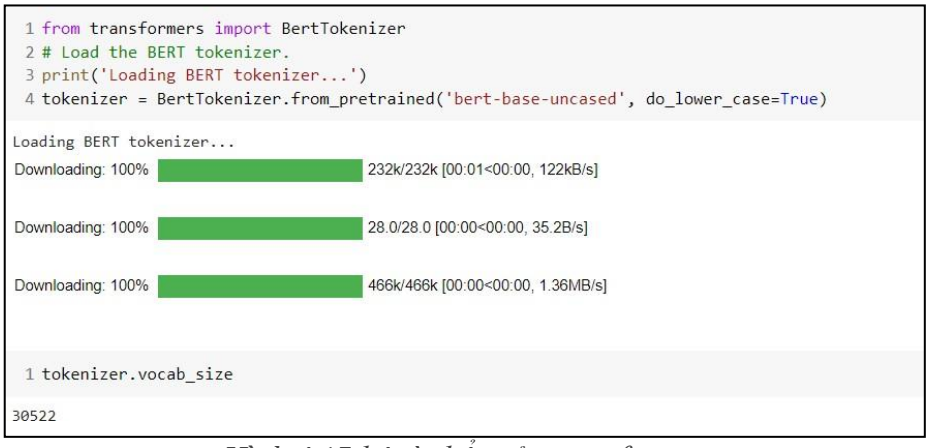
4.4.1 Tiền xử lý dữ liệu (preprocessing data)

Dữ liệu được làm "sạch" qua các bước sau:

1. Lower: đưa toàn bộ nội dung về dạng chữ thường (bỏ hết chữ cái in hoa)
2. Xoá hết các ký tự đặc biệt trong văn bản như: [] \ / | @ # ,
3. Chuyển hết các dấu chấm câu thành "<EOS>" (end of sentence)

4.4.2 Mã hoá từ (word tokenization)

Ta sử dụng bộ từ điển của thư việc transformer:



Hình 4.15 bộ từ điển của transformer

Bộ từ điển này được huấn luyện qua tập dữ liệu lớn trên Wiki và có 30522 từ.

4.4.3 Biểu diễn câu

Sau khi có bộ từ điển, tiến hành mã hoá câu:

```
# Tokenize all of the sentences and map the tokens to thier word IDs.
input_ids = []
# For every sentence...
for sent in sentences:
    # `encode` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    encoded_sent = tokenizer.encode(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        # This function also supports truncation and conversion
                        # to pytorch tensors, but we need to do padding, so we
                        # can't use these features :( .
                        max_length = MAX_LEN_SENTENCE,          # Truncate all sentences.
                        #return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_sent)
```

Hình 4.16 Mã hoá câu trong BERT

Hình dưới đây minh hoạ cho việc mã hoá câu:

```
shenzhen arrived at sort facility shenzhen EOS shenzhen shipment has been processed
Token IDs: [101, 26555, 3369, 2012, 4066, 4322, 26555, 1041, 2891, 26555, 22613, 2038]
```

Hình 4.17 Ví dụ mã hoá câu trong BERT

Sau đó thêm phần đệm cho câu, những câu không đủ 256 từ sẽ được thêm phần đệm để đủ độ dài 256:

```
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="post")
print('\Done.')


Padding/truncating all sentences to 256 values...

Padding token: "[PAD]", ID: 0
\Done.
```

Hình 4.18 Thêm phần đệm cho câu

Cuối cùng, tạo attention mask (mặt lạ chú ý):

```
# Create attention masks
attention_masks = []
# For each sentence...
for sent in input_ids:

    # Create the attention mask.
    #   - If a token ID is 0, then it's padding, set the mask to 0.
    #   - If a token ID is > 0, then it's a real token, set the mask to 1.
    att_mask = [int(token_id > 0) for token_id in sent]

    # Store the attention mask for this sentence.
    attention_masks.append(att_mask)
```

Hình 4.19 Attention mask

Attention mask được dùng trong quá trình đưa dữ liệu đầu vào huấn luyện, nó là mặt lạ chứa các số 0 và 1, giá trị 0 ám chỉ phần đệm trong câu và 1 chỉ từ có trong từ

điển. Khi đưa dữ liệu vào huấn luyện, vector của câu sẽ nhân với attention mask để lấy ra những thành phần có nghĩa trong vector (không phải là phần đệm).

4.4.4 Mô hình huấn luyện

Để sử dụng được mô hình đã được huấn luyện từ thư viện transformers, cần phải khai báo:

```
from transformers import BertForSequenceClassification, AdamW, BertConfig
# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = NUMBER_OF_LABELS, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)
# Tell pytorch to run this model on the GPU.
model.cuda()
```

Hình 4.20 Khai báo mô hình BERT

Tổng quan về mô hình được miêu tả ở hình dưới đây:

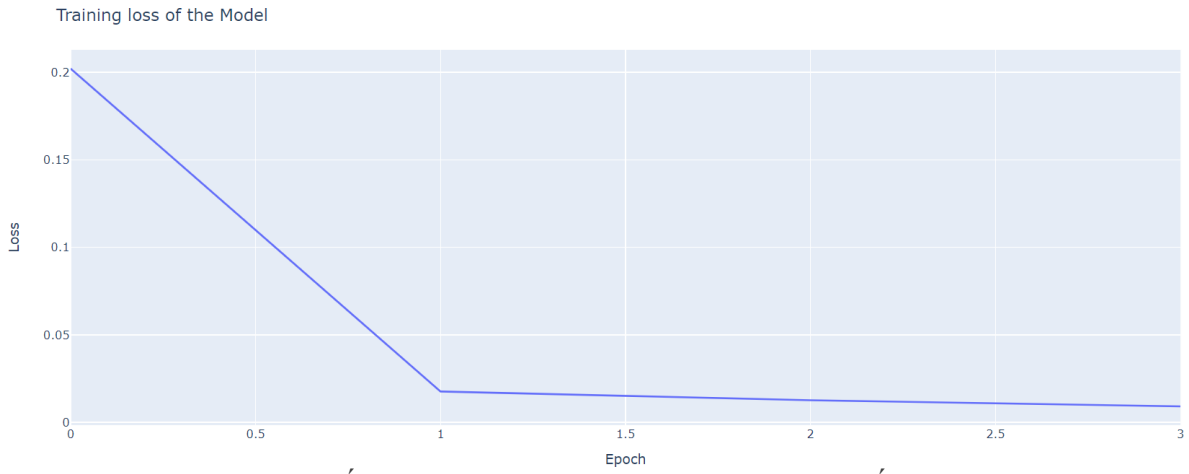
```
The BERT model has 201 different named parameters.

==== Embedding Layer ====

bert.embeddings.word_embeddings.weight                  (30522, 768)
bert.embeddings.position_embeddings.weight                (512, 768)
bert.embeddings.token_type_embeddings.weight                (2, 768)
bert.embeddings.LayerNorm.weight                            (768,)
bert.embeddings.LayerNorm.bias                              (768,)

==== First Transformer ====

bert.encoder.layer.0.attention.self.query.weight          (768, 768)
bert.encoder.layer.0.attention.self.query.bias              (768,)
bert.encoder.layer.0.attention.self.key.weight            (768, 768)
bert.encoder.layer.0.attention.self.key.bias                (768,)
bert.encoder.layer.0.attention.self.value.weight          (768, 768)
bert.encoder.layer.0.attention.self.value.bias              (768,)
bert.encoder.layer.0.attention.output.dense.weight        (768, 768)
bert.encoder.layer.0.attention.output.dense.bias            (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight      (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias        (768,)
bert.encoder.layer.0.intermediate.dense.weight           (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias                (3072,)
bert.encoder.layer.0.output.dense.weight                 (768, 3072)
bert.encoder.layer.0.output.dense.bias                      (768,)
bert.encoder.layer.0.output.LayerNorm.weight                (768,)
bert.encoder.layer.0.output.LayerNorm.bias                  (768,)

==== Output Layer ====

bert.pooler.dense.weight                                  (768, 768)
bert.pooler.dense.bias                                      (768,)
classifier.weight                                           (6, 768)
classifier.bias                                                (6,)
```

Hình 4.21 Tổng quan mô hình BERT

Theo như hướng dẫn của tài liệu, mô hình BERT đã được pre-training nên chỉ cần thêm tầng output cho mô hình. Hơn nữa, tài liệu cũng hướng dẫn chỉ cần huấn luyện mô hình thêm 1 đến 4 lần nữa, nếu nhiều hơn thì độ chính xác cũng không được cải thiện.

4.4.5 Kết quả huấn luyện mô hình

Hình dưới đây mô tả độ biến thiên của hàm Loss sau 4 epoch:



Hình 4.22 Độ biến thiên của hàm Loss trong quá trình huấn luyện

Giá trị của hàm Loss giảm dần sau 4 epoch, và độc hính xác của mô hình đạt được trên tập huấn luyện là 99.77%.

4.5 Mô hình LSTM để kiểm tra các log đơn hàng có sắp xếp sai thứ thự

Mô hình LSTM này được triển khai giống như mô LSTM trích xuất trạng thái, tuy nhiên bộ dữ liệu huấn luyện có thêm trường wrong_order có hai nhãn là 0 và 1 để đánh dấu đơn hàng có sắp xếp thứ tự các log đúng hay không.

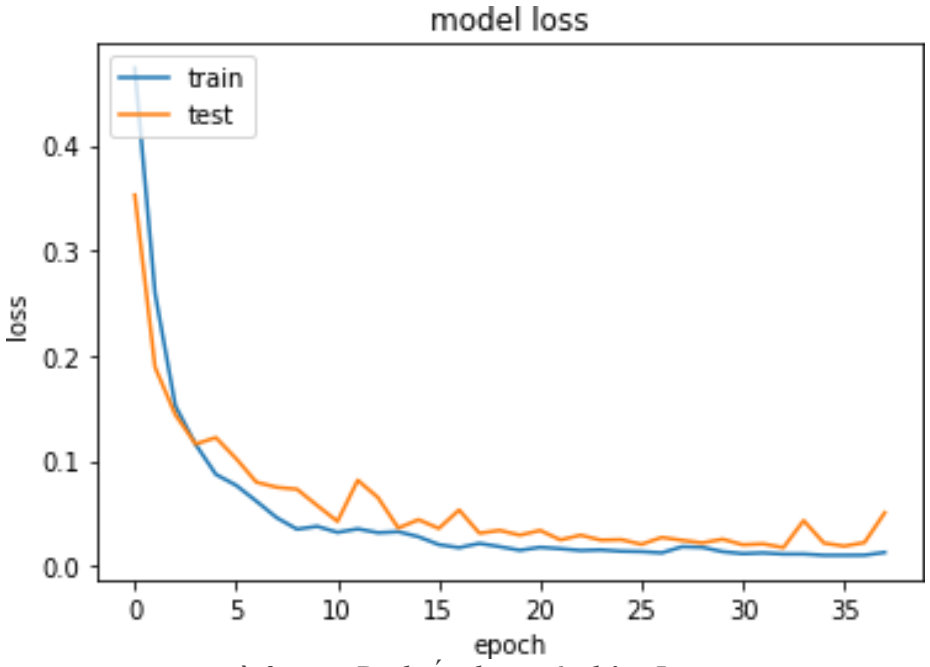
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9961 entries, 0 to 9960
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   tracking_number     9961 non-null   object
 1   checkpoint_status   9961 non-null   object
 2   status_description  9961 non-null   object
 3   wrong_order         9961 non-null   int64
dtypes: int64(1), object(3)
memory usage: 311.4+ KB
```

Hình 4.23 Thông tin dữ liệu đầu vào

	tracking_number	checkpoint_status	status_description	wrong_order
0	YT2101621236003930	TRACKING_ONLINE	SHENZHEN Arrived at Sort Facility SHENZHEN .	0
1	YT2101621236003930	TRACKING_ONLINE	SHENZHEN Arrived at Sort Facility SHENZHEN	0
2	YT2101621236003930	TRACKING_ONLINE	SHENZHEN Arrived at Sort Facility SHENZHEN	0
3	YT2101621236003930	TRACKING_ONLINE	SHENZHEN Departed Facility In processing cent...	1
4	YT2101621236003930	IN_US	SHENZHEN Arrived at Sort Facility SHENZHEN	0

Hình 4.24 Dữ liệu đầu vào

Hình dưới đây mô tả độ biến thiên của hàm Loss sau 35 epoch:



Hình 4.25 Độ biến thiên của hàm Loss

Độ chính xác mà mô hình đạt được trên tập huấn luyện là 98.77%.

CHƯƠNG 5. ĐÁNH GIÁ KẾT QUẢ VÀ TỔNG KẾT

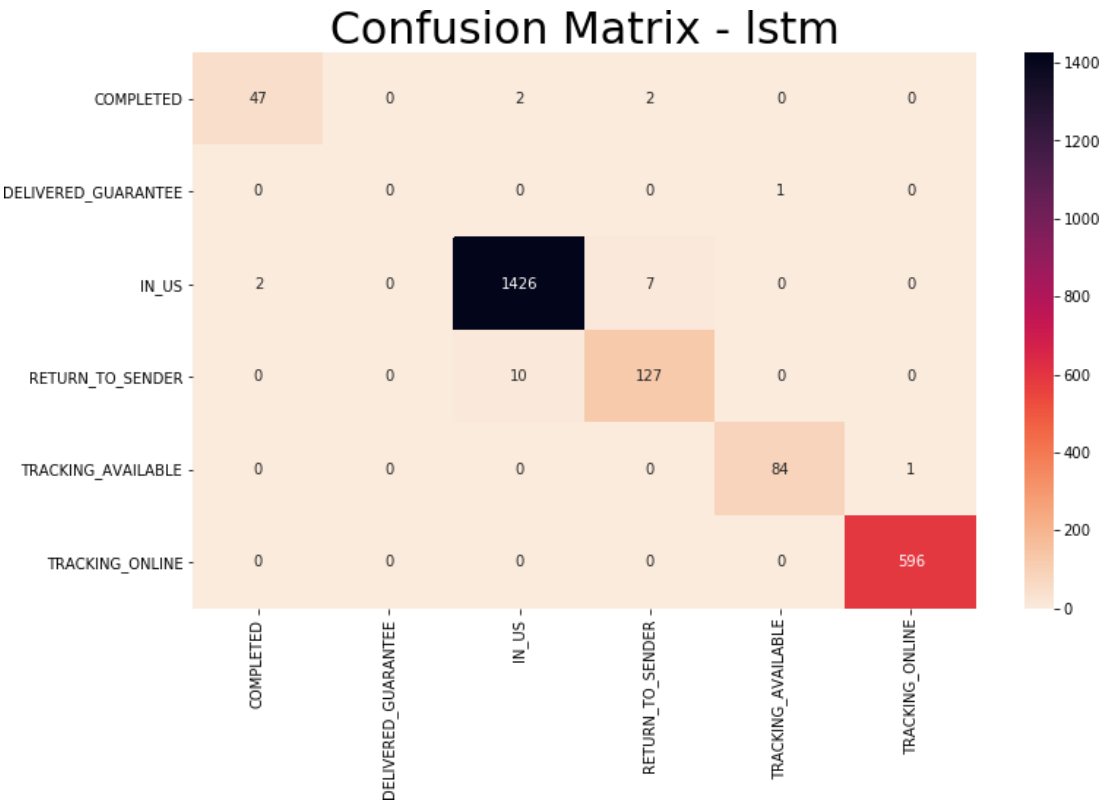
5.1 Đánh giá độ chính xác của các mô hình trên tập dữ liệu Test

Sau khi huấn luyện các mô hình xong, thực hiện đánh giá độ chính xác của các mô hình trên tập dữ kiệu test với khoảng 2305 logs, tập dữ liệu test được tách riêng biệt với tập dữ liệu được cho vào huấn luyện mô hình. Kết quả đạt được cụ thể như sau:

- Mô hình LSTM trích xuất trạng thái:

- Độ chính xác trên tập test đạt: 98.915%

- Ma trận Confusion dưới đây mô tả kết quả dự đoán của mô hình trên tập test

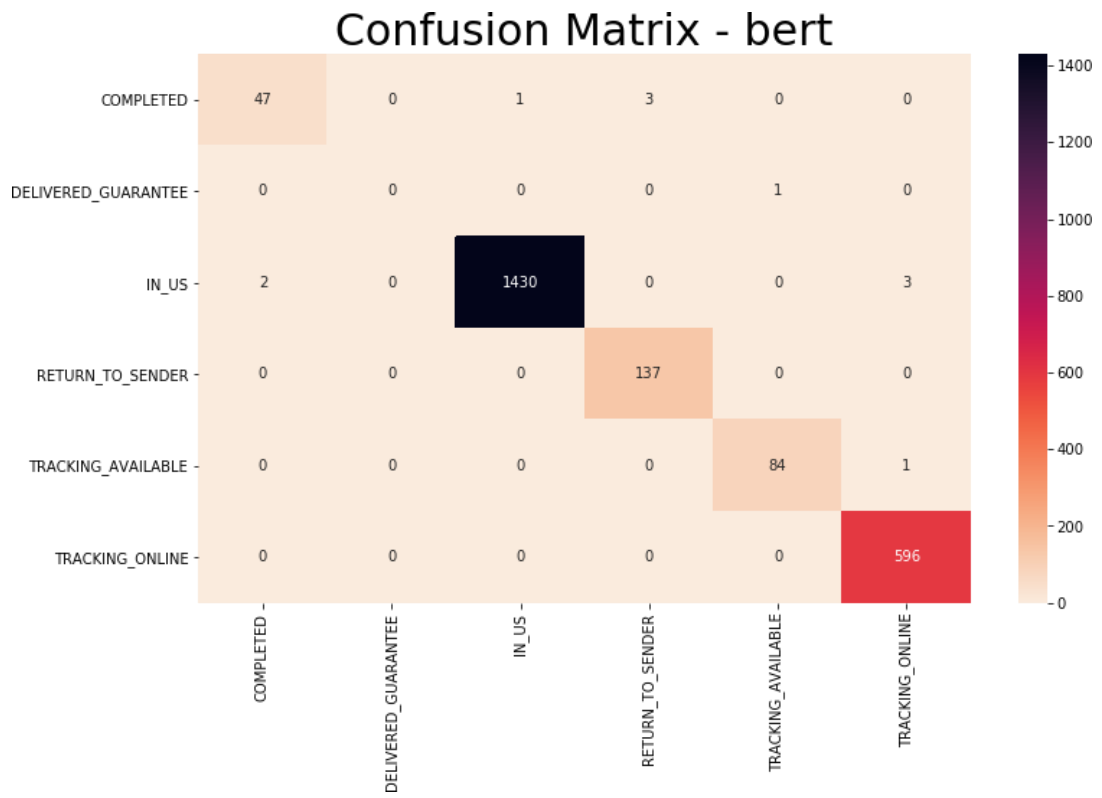


Hình 5.1 Confusion matrix mô hình LSTM

- Mô hình BERT trích xuất trạng thái:

- Độ chính xác trên tập test đạt: 99.523%

- Ma trận Confusion dưới đây mô tả kết quả dự đoán của mô hình trên tập test

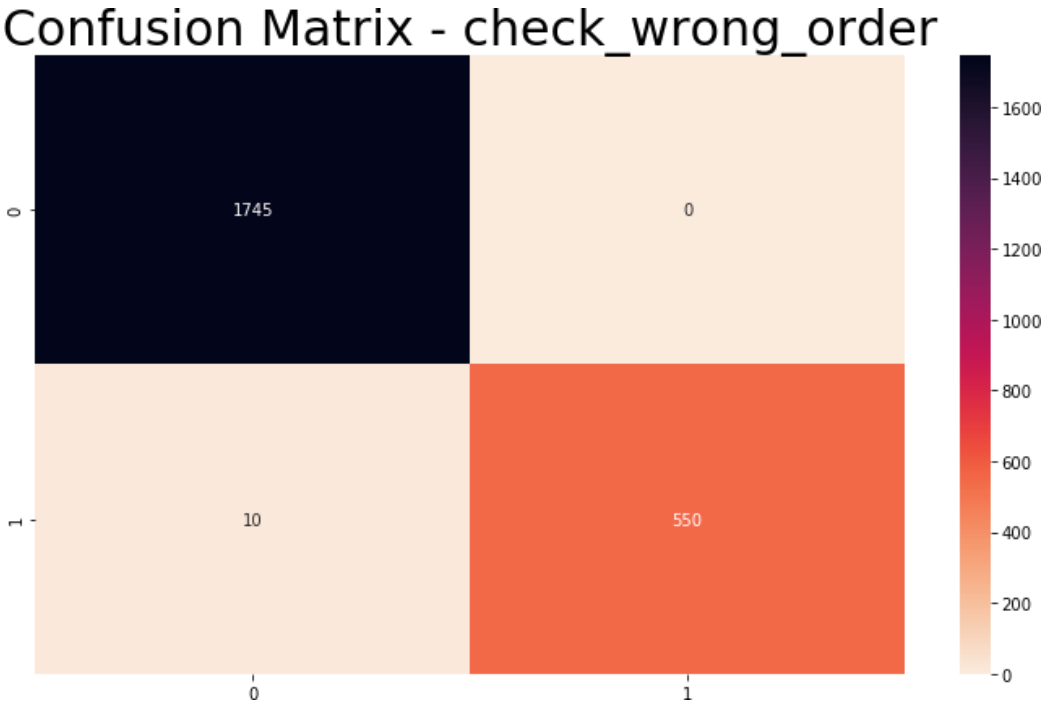


Hình 5.2 Confusion matrix mô hình BERT

- Mô hình LSTM trích xuất trạng thái:

- Độ chính xác trên tập test đạt: 99.566%

- Ma trận Confusion dưới đây mô tả kết quả dự đoán của mô hình trên tập test



Hình 5.3 Confusion matrix mô hình LSTM

5.2 Vận hành hệ thống

5.2.1 Hệ thống thực tế

Như đã trình bày ở phần "Giới thiệu mô hình kinh doanh" và "Đặt vấn đề", hệ thống trung bình mỗi ngày tiếp nhận khoảng 2000 đơn hàng với xấp xỉ khoảng 3500 log phát sinh tại Mỹ và Trung Quốc. Thời gian xử lý mỗi log của đơn hàng qua API mất vào khoảng 1.35s.

Sau mỗi ngày, tổng hợp lại kết quả dự đoán của mô hình từ cơ sở dữ liệu và so sánh với kết quả của hệ thống cũ đang chạy (dung rule engine: sử dụng matching pattern để trích xuất trạng thái), độ chính xác của mô hình trong thực tế vào khoảng 85%. Việc độ chính xác giảm trong thực tế xuất phát từ việc gán nhãn dữ liệu chưa đúng cho một số log.

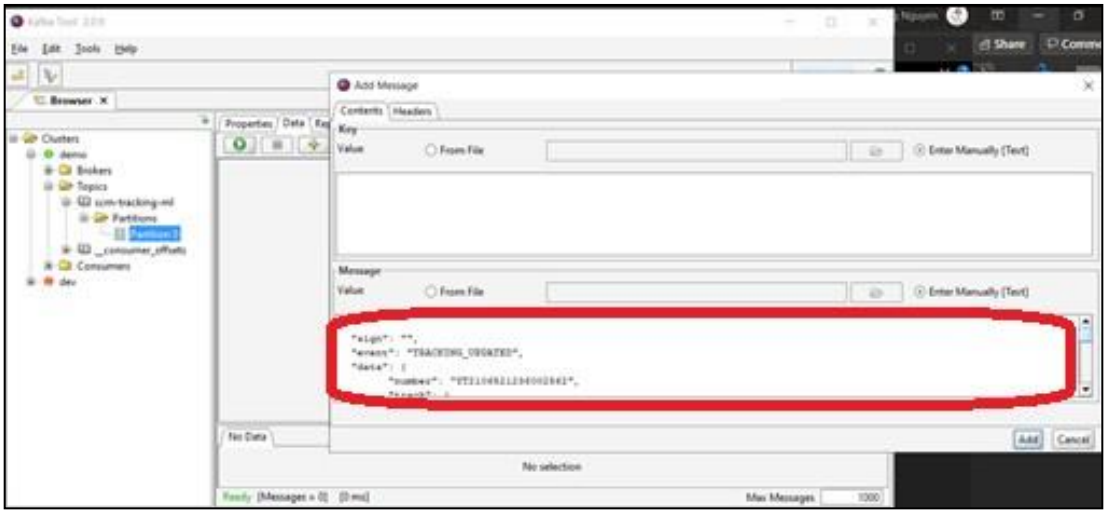
Hệ thống học máy trong thực tế vẫn đang trong quá trình thử nghiệm và theo dõi để cải thiện độ chính xác cùng với tối ưu hóa thời gian xử lý.

5.2.2 Hệ thống mô phỏng

5.2.2.1. Ghi log lên Kafka và xem kết quả trên Grafana

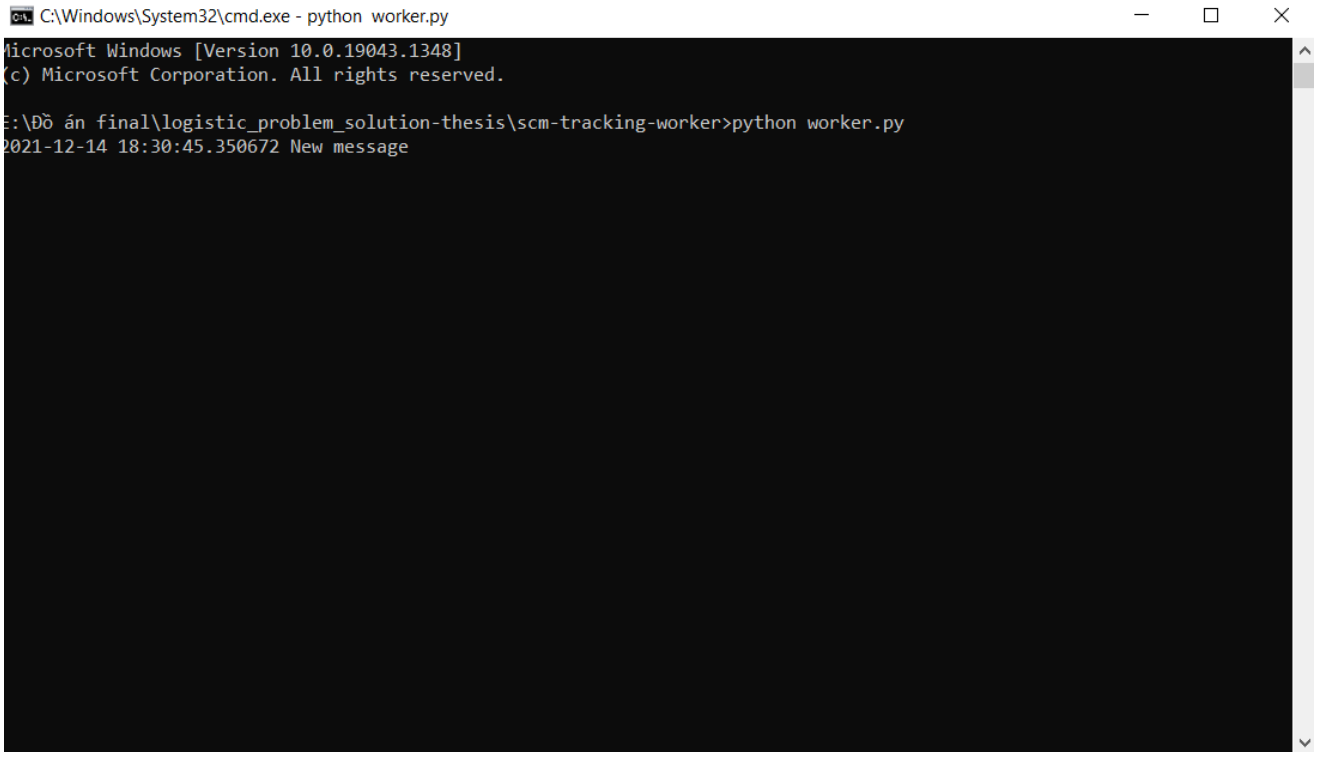
Để kiểm nghiệm hệ thống có vận hành đúng không, sẽ ghi log của đơn hàng có mã là "YT2106521236002572" trên Kafka tool và quan sát trên Grafana có hiển thị kết quả của đơn hàng không. Quá trình vận hành như sau:

Sử dụng Kafka tool để kết nối tới máy chủ Kafka và chọn topic scm-tracking-ml. Sau đó, ghi log của đơn hàng qua Kafka tool.



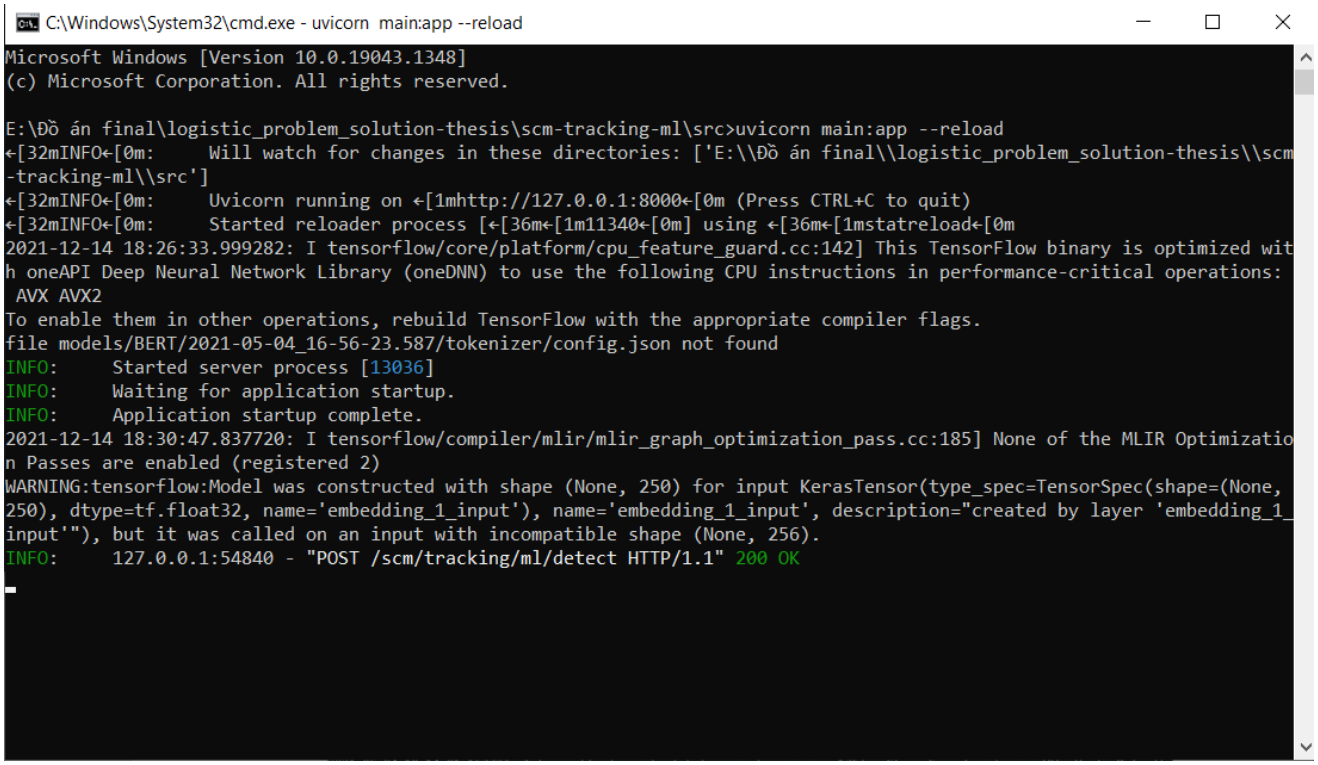
Hình 5.4 Ghi log trên Kafka tool

Sau khi log được bắn lên máy chủ Kafka, worker đang chạy sẽ nhận được log và thực hiện gọi service học máy để nhận được kết quả dự đoán trạng thái từ hai mô hình LSTM và BERT cùng với kết quả dự đoán log có sắp xếp sai thứ tự không.



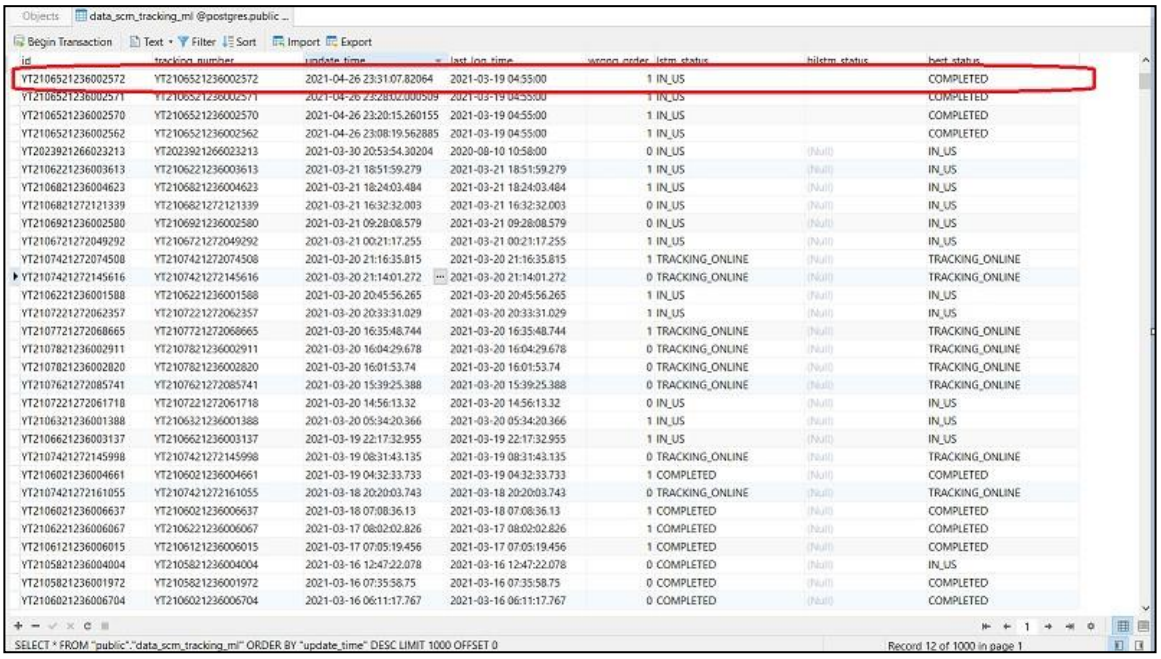
Hình 5.5 Worker bắt được log trên Kafka server

Service học máy báo nhận được yêu cầu và trả kết quả về worker:



Hình 5.6 Service ML trả về kết quả thành công

Sau đó worker gửi kết quả dự đoán cùng với log của đơn hàng lên elasticsearch và NiFi sẽ chuyển dữ liệu này tới CSDL Postgres. Kết quả được cập nhật như hình dưới đây:



id	tracking_number	update_time	last_log_time	wrong_order	lstm_status	bilstm_status	bert_status
YT2106521236002572	YT2106521236002572	2021-04-26 23:31:07.82064	2021-03-19 04:55:00		1 IN_US		COMPLETED
YT2106521236002571	YT2106521236002571	2021-04-26 23:28:02.000509	2021-03-19 04:55:00		1 IN_US		COMPLETED
YT2106521236002570	YT2106521236002570	2021-04-26 23:20:15.260155	2021-03-19 04:55:00		1 IN_US		COMPLETED
YT2106521236002562	YT2106521236002562	2021-04-26 23:08:19.562885	2021-03-19 04:55:00		1 IN_US		COMPLETED
YT2023921266023213	YT2023921266023213	2021-03-30 20:53:54.30204	2020-08-10 10:58:00		0 IN_US	(Null)	IN_US
YT2106221236003613	YT2106221236003613	2021-03-21 18:51:59.279	2021-03-21 18:51:59.279		1 IN_US	(Null)	IN_US
YT2106821236004623	YT2106821236004623	2021-03-21 18:24:03.484	2021-03-21 18:24:03.484		1 IN_US	(Null)	IN_US
YT2106821272121339	YT2106821272121339	2021-03-21 16:32:32.003	2021-03-21 16:32:32.003		0 IN_US	(Null)	IN_US
YT2106921236002580	YT2106921236002580	2021-03-21 09:28:08.579	2021-03-21 09:28:08.579		0 IN_US	(Null)	IN_US
YT2106721272049292	YT2106721272049292	2021-03-21 00:21:17.255	2021-03-21 00:21:17.255		1 IN_US	(Null)	IN_US
YT2107421272074508	YT2107421272074508	2021-03-20 21:16:35.815	2021-03-20 21:16:35.815		1 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2107421272145616	YT2107421272145616	2021-03-20 21:14:01.272	2021-03-20 21:14:01.272		0 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2106221236001588	YT2106221236001588	2021-03-20 20:45:56.265	2021-03-20 20:45:56.265		1 IN_US	(Null)	IN_US
YT2107221272062357	YT2107221272062357	2021-03-20 20:33:31.029	2021-03-20 20:33:31.029		1 IN_US	(Null)	IN_US
YT2107721272068665	YT2107721272068665	2021-03-20 16:35:48.744	2021-03-20 16:35:48.744		1 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2107821236002911	YT2107821236002911	2021-03-20 16:04:29.678	2021-03-20 16:04:29.678		0 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2107821236002820	YT2107821236002820	2021-03-20 16:01:53.74	2021-03-20 16:01:53.74		0 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2107621272085741	YT2107621272085741	2021-03-20 15:39:25.388	2021-03-20 15:39:25.388		0 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2107221272061718	YT2107221272061718	2021-03-20 14:56:13.32	2021-03-20 14:56:13.32		0 IN_US	(Null)	IN_US
YT2106321236001388	YT2106321236001388	2021-03-20 05:34:20.366	2021-03-20 05:34:20.366		1 IN_US	(Null)	IN_US
YT2106621236003137	YT2106621236003137	2021-03-19 22:17:32.955	2021-03-19 22:17:32.955		1 IN_US	(Null)	IN_US
YT2107421272145998	YT2107421272145998	2021-03-19 08:31:43.135	2021-03-19 08:31:43.135		0 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2106021236004661	YT2106021236004661	2021-03-19 04:32:33.733	2021-03-19 04:32:33.733		1 COMPLETED	(Null)	COMPLETED
YT2107421272161055	YT2107421272161055	2021-03-18 20:20:03.743	2021-03-18 20:20:03.743		0 TRACKING_ONLINE	(Null)	TRACKING_ONLINE
YT2106021236006637	YT2106021236006637	2021-03-18 07:08:36.13	2021-03-18 07:08:36.13		1 COMPLETED	(Null)	COMPLETED
YT2106221236006067	YT2106221236006067	2021-03-17 08:02:02.826	2021-03-17 08:02:02.826		1 COMPLETED	(Null)	COMPLETED
YT2106121236006015	YT2106121236006015	2021-03-17 07:05:19.456	2021-03-17 07:05:19.456		1 COMPLETED	(Null)	COMPLETED
YT2105821236004004	YT2105821236004004	2021-03-16 12:47:22.078	2021-03-16 12:47:22.078		0 COMPLETED	(Null)	IN_US
YT2105821236001972	YT2105821236001972	2021-03-16 07:35:58.75	2021-03-16 07:35:58.75		0 COMPLETED	(Null)	COMPLETED
YT2106021236006704	YT2106021236006704	2021-03-16 06:11:17.767	2021-03-16 06:11:17.767		0 COMPLETED	(Null)	COMPLETED

Hình 5.7 Đơn hàng được cập nhật trên CSDL Postgres

Khi có dữ liệu mới từ Postgres, Grafana sẽ cập nhật lên các biểu đồ:

Hình 5.8 Kết quả của đơn hàng YT2106521236002572 trên các biểu đồ

Update Status

time	Tracking number	LSTM Status	BERT Status
2021-04-27 06:31:07	YT2106521236002572	IN_US	COMPLETED
2021-04-27 06:28:02	YT2106521236002571	IN_US	COMPLETED
2021-04-27 06:20:15	YT2106521236002570	IN_US	COMPLETED
2021-04-27 06:08:19	YT2106521236002562	IN_US	COMPLETED
2021-03-31 03:53:54	YT2023921266023213	IN_US	IN_US
2021-03-22 01:51:59	YT2106221236003613	IN_US	IN_US
2021-03-22 01:24:03	YT2106821236004623	IN_US	IN_US
2021-03-21 23:32:32	YT2106821272121339	IN_US	IN_US
2021-03-21 16:28:08	YT2106921236002580	IN_US	IN_US
2021-03-21 07:21:17	YT2106721272049292	IN_US	IN_US
2021-03-21 04:16:35	YT2107421272074508	TRACKING_ONLINE	TRACKING_ONLINE
2021-03-21 04:14:01	YT2107421272145616	TRACKING_ONLINE	TRACKING_ONLINE

Hình 5.9 Kết quả của đơn hàng YT2106521236002572

5.2.2.2. *Xử lý log tiếng Trung*

Em thực hiện bắn một log của đơn hàng có nội dung bằng tiếng Trung lên máy chủ Kafka và quan sát kết quả trên Grafana:


```
"ylt2": "2020-12-17 08:29:24",
"z9": [],
"z1": [
 {
  "a": "2020-08-10 10:58",
  "c": "美国",
  "d": "",
  "z": "【美国】已妥投"
 },
 {
  "a": "2020-08-10 07:10",
  "c": "美国",
  "d": "",
  "z": "【美国】安排投递"
 },
 {
  "a": "2020-08-10 06:11",
  "c": "美国",
  "d": "",
  "z": "已到达【美国】投递局"
 },
 {
  "a": "2020-08-08 11:49",
  "c": "美国",
  "d": "",
  "z": "到达【美国】二级处理中心"
 },
 {
  "a": "2020-08-08 11:49",
  "c": "美国",
  "d": "",
  "z": "到达境外经转局"
 },
```


Hình 5.10 Log tiếng Trung

Tracking log of tracking_number YT44958033224343


```
[
  {
    "a": "2020-08-10 10:58",
    "c": "United States",
    "d": "",
    "z": "[United States] has been properly voted"
  },
  {
    "a": "2020-08-10 07:10",
    "c": "United States",
    "d": "",
    "z": "[United States] Arrange delivery"
  },
  {
    "a": "2020-08-10 06:11",
    "c": "United States",
    "d": "",
    "z": "Already arrived at [United States] Delivery Office"
  },
  {
    "a": "2020-08-08 11:49",
    "c": "United States",
    "d": "",
    "z": "Arrive at the secondary processing center [US]"
  },
  {
    "a": "2020-08-08 11:49",
    "c": "United States",
    "d": "",
    "z": "Arrival at overseas transit bureau"
  },
```


Hình 5.11 Log sau khi được xử lý tiếng Trung trên Grafana

5.2.2.3. Kiểm tra đơn hàng log có sắp xếp sai thứ tự

Bên cạnh đó, các biểu đồ và bảng thống kê số lượng đơn hàng có log sắp xếp sai thứ tự được thể hiện như sau:

Hình 5.12 Biểu đồ thống kê số lượng đơn hàng sai thứ tự log

List of wrong order items

time ↓	tracking_number
2021-04-27 06:31:07	YT2106521236002572
2021-04-27 06:28:02	YT2106521236002571
2021-04-27 06:20:15	YT2106521236002570
2021-04-27 06:08:19	YT2106521236002562
2021-03-22 01:51:59	YT2106221236003613
2021-03-22 01:24:03	YT2106821236004623
2021-03-21 07:21:17	YT2106721272049292
2021-03-21 04:16:35	YT2107421272074508
2021-03-21 03:45:56	YT2106221236001588
2021-03-21 03:33:31	YT2107221272062357
2021-03-20 23:35:48	YT2107721272068665
2021-03-20 12:34:20	YT2106321236001388
2021-03-20 05:17:32	YT2106621236003137
2021-03-19 11:32:33	YT2106021236004661
2021-03-18 14:08:36	YT2106021236006637

Hình 5.13 Danh sách đơn hàng sai thứ tự log

5.2.2.4. Xem log của đơn hàng trên Grafana

Ngoài ra, có thể xem log của đơn hàng bằng cách nhập mã đơn hàng lên vào ô tracking_number trên màn hình Grafana, sau đó log của đơn hàng sẽ được truy vấn trên elasticsearch và hiển thị trong bảng “Tracking logs”:

Hình 5.14 Nhập mã đơn hàng vào ô tracking_number

Tracking logs of YT2106521236002572


```
[
 {
  "a": "2021-03-19 04:55",
  "c": "SAN PEDRO,CA",
  "d": "",
  "z": "Delivered at Sort Facility"
 },
 {
  "a": "2021-03-16 23:46",
  "c": "San Pedro,CA",
  "d": "",
  "z": "depart ship ptnr site usps awaits item"
 },
 {
  "a": "2021-03-16 11:44",
  "c": "San Pedro,CA",
  "d": "",
  "z": "arrived ship ptnr site usps awaits item"
 },
 {
  "a": "2021-03-16 11:44",
  "c": "Rancho Dominguez,CA",
  "d": "",
  "z": "hub scanned"
 },
 {
  "a": "2021-03-15 18:43",
  "c": "",
  "d": "",
  "z": "departed from facility"
```


Hình 5.15 Nội dung log của đơn hàng

5.3 Định hướng phát triển

Qua quá trình thử nghiệm và vận hành hệ thống, nhận thấy hệ thống vẫn còn một vài điểm cần phải cải thiện và tối ưu như sau:

- Thời gian phản hồi của API cần nhanh hơn: hiện tại là khoảng 2s.
- Khi chạy với dữ liệu thực tế, cần phải giám sát sau mỗi ngày để phát hiện những log mới phát sinh và các trường hợp ngoại lệ (log sai thứ tự), nhằm thu thập và huấn luyện cho mô hình đạt độ chính xác cao hơn.
- Khi dữ liệu trong cơ sở dữ liệu đủ lớn, cần tìm cách tối ưu hoá để luồng xử lý dữ liệu được trơn tru và hiệu suất cao.

TÀI LIỆU THAM KHẢO

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention Is All You Need, 2017.

[2] Đắm mình vào học sâu.

[3] Machine Learning cơ bản Vũ Hữu Tiệp.

[4] Deep Learning cơ bản Nguyễn Thanh Tuấn.

[5] Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale by Neha Narkhede, Gwen Shapira, Todd Palino.