AI VIET NAM – AI COURSE 2023

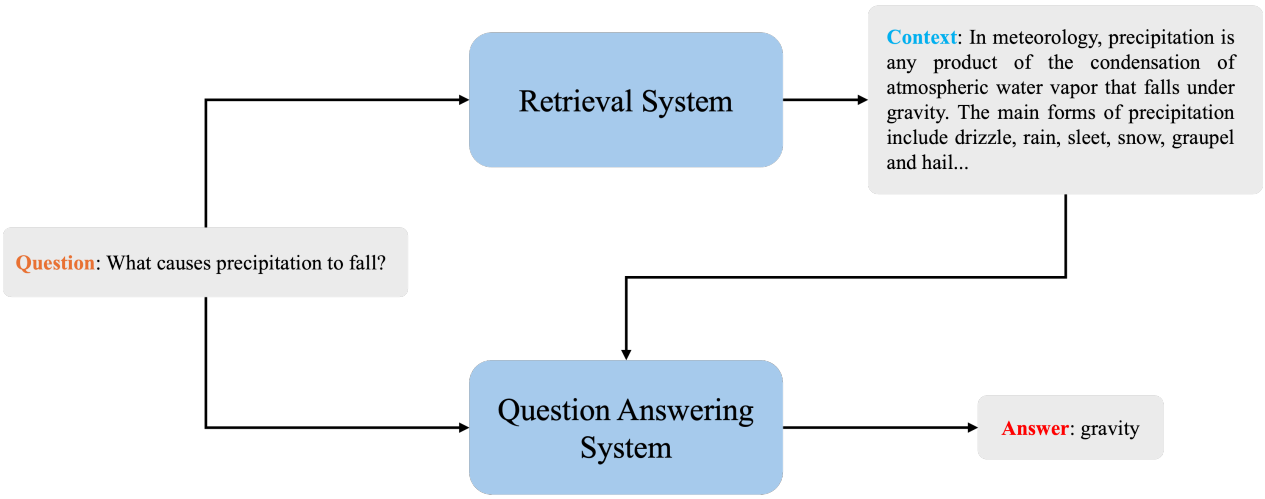
Xây dựng hệ thống hỏi-đáp mở với hệ cơ sở dữ liệu vector

Dinh-Thang Duong, Minh-Duc Bui và Quang-Vinh Dinh
PR-Team: Minh-Châu Phạm, Hoàng-Nguyên và Vũ Đăng-Nhã Nguyễn

Ngày 13 tháng 2 năm 2024

Phần I: Giới thiệu

Trong project này, chúng ta sẽ tập trung vào việc phát triển một hệ thống end-to-end hỏi đáp tự động, với khả năng trả lời một câu hỏi với nội dung bất kì. Hệ thống mà chúng ta cài đặt trong project này bao gồm hai phần chính là Retriever và Reader, với mục tiêu xây dựng một hệ thống toàn diện có khả năng rút trích thông tin từ văn bản và cung cấp câu trả lời cho các câu hỏi dựa trên nội dung của đoạn văn.

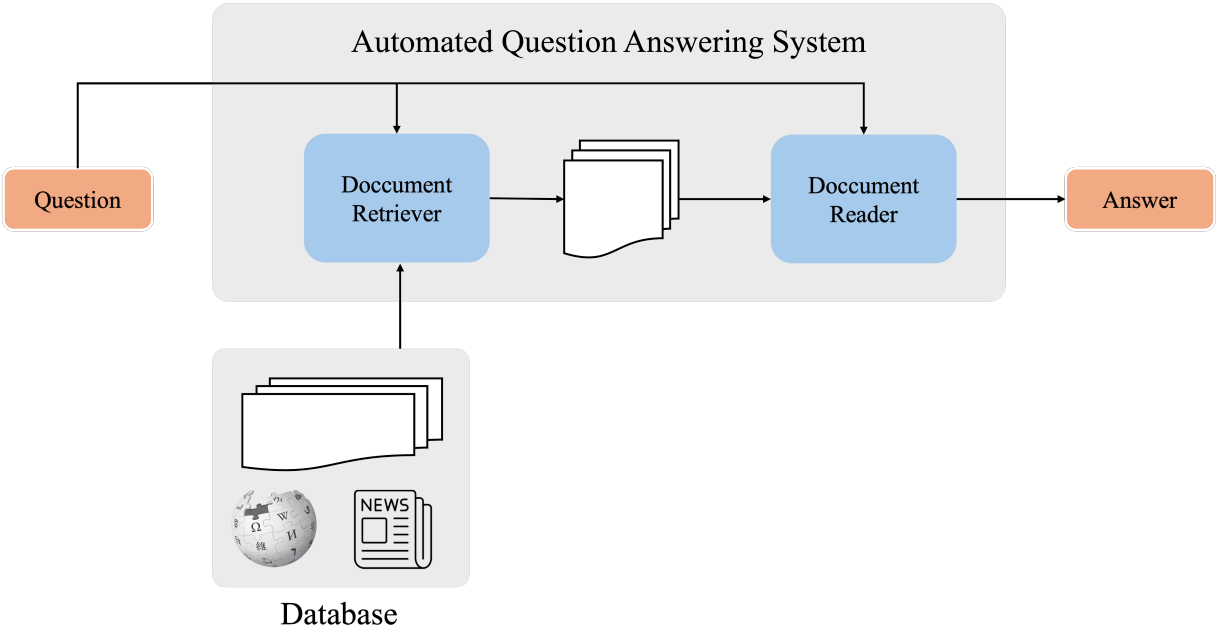


Cụ thể, ta sẽ xây dựng chương trình dựa vào dataset SQuAD2.0, một bộ dữ liệu về đọc hiểu, vector database là FAISS và mô hình BERT để thực hiện các nhiệm vụ cụ thể trong chương trình. Input và output của chương trình như sau:

- **Input:** Một câu hỏi.
- **Output:** Câu trả lời tương ứng.

Phần II: Nội dung

Để xây dựng một chương trình End-to-end Question Answering, chúng ta cần hoàn thiện hai module chính bao gồm Retriever và Reader:



Hình 1: Ảnh minh hoạt tổng quát về một hệ thống End-to-end QA.

Theo đó, nội dung của bài viết sẽ trình bày chương trình cài đặt cho từng thành phần như sau:

1. **Dataset description: SQuAD2.0** Stanford Question Answering Dataset (SQuAD) là bộ data theo hướng đọc hiểu, bao gồm các đoạn văn (passage) khác nhau về nhiều chủ đề, ứng với mỗi đoạn văn sẽ có một các câu hỏi ngắn kèm theo. Bảng 1 miêu tả cấu trúc chi tiết về dataset SQuAD2.0:

Context: The English name "Normans" comes from the French words Normans/Normanz, plural of Normant, modern French normand, which is itself borrowed from (Old Low Franconian Nortmann "Northman" or directly from Old Norse Nordmaðr, Latinized variously as Nortmannus, Normannus, or Nordmannus (recorded in Medieval Latin, **9th century**) to mean "Norseman, **Viking**".

Q1: What causes precipitation to fall?
A1: Viking

Q2: When was the Latin version of the word Norman first recorded?
A2: 9th century

Q3: When was the French version of the word Norman first recorded?
A3: No Answers

Context: In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

Q1: What causes precipitation to fall?
A1: Gravity

Q2: What is another main form of precipitation be- sides drizzle, rain, snow, sleet and hail?
A2: Graupel

Q3: Where do water droplets collide with ice crystals to form precipitation?
A3: Within a cloud

Hình 2: Ví dụ minh họa về dataset SQuAD2.0.

	SQuAD 2.0
Train	
Total examples	130,319
Negative examples	43,498
Total articles	442
Articles with negatives	285
Development	
Total examples	11,873
Negative examples	5,945
Total articles	35
Articles with negatives	35
Test	
Total examples	8,862
Negative examples	4,332
Total articles	28
Articles with negatives	28

Bảng 1: Thống kê số lượng sample của dataset SQuAD2.0.

Câu trả lời cho các câu hỏi ngắn là những từ/cụm từ có sẵn trong đoạn văn cho trước (không yêu cầu suy luận phức tạp), hoặc các câu hỏi không trả lời được dựa vào đoạn văn (answer là no answer). Bảng 2 thống kê về dataset SQuAD2.0:

Answer type	**Percentage**	**Example**
Date	8.9%	19 October 2023
Other Numeric	10.9%	12
Person	12.9%	Thomas Coke
Location	4.4%	Germany
Other Entity	15.3%	ABC Sports
Common Noun Phrase	31.8%	property damage
Adjective Phrase	3.9%	second-largest
Verb Phrase	5.5%	returned to Earth
Clause	3.7%	to avoid trivialization
Other	2.7%	quietly

Bảng 2: Thống kê các loại câu trả lời khác nhau của dataset SQuAD2.0.

2. **Reader: DistilBERT** Đầu tiên ta sẽ xây dựng model Reader hay chính là model QA trong project này.

   (a) **Install and import bibraries:** Đầu tiên ta sẽ install một số thư viện cần thiết mà Colab chưa hỗ trợ.

```
!pip install -qq datasets==2.16.1 evaluate==0.4.1 transformers[sentencepiece
    ]==4.35.2
!pip install -qq accelerate==0.26.1
!apt install git-lfs
```

   Sau đó ta sẽ tiến hành login vào HuggingFace để download dataset và model có sẵn. Khi chạy block code này thì HuggingFace sẽ đưa ra một đường dẫn đến trang HuggingFace để lấy mã token.

```
from huggingface_hub import notebook_login

notebook_login()
```

Cuối cùng ta sẽ import các thư viện chính được sử dụng trong phần này:

```
import numpy as np
from tqdm.auto import tqdm
import collections

import torch

from datasets import load_dataset
from transformers import AutoTokenizer
from transformers import AutoModelForQuestionAnswering
from transformers import TrainingArguments
from transformers import Trainer
import evaluate

device = torch.device("cuda") if torch.cuda.is_available() else \
        torch.device("cpu")
```

(b) **Setup config:** Tiếp theo ta sẽ setup một số config cơ bản:

```
# Sử dụng mô hình "distilbert-base-uncased" làm mô hình checkpoint
MODEL_NAME = "distilbert-base-uncased"

# Độ dài tối đa cho mỗi đoạn văn bản sau khi được xử lý
MAX_LENGTH = 384

# Khoảng cách giữa các điểm bắt đầu của các đoạn văn bản liên tiếp
STRIDE = 128
```

(c) **Setup Dataset:**

- **Download dataset:**

```
# Download squad dataset từ HuggingFace
DATASET_NAME = 'squad_v2'
raw_datasets = load_dataset(DATASET_NAME)
```

- **Load tokenizer and run some examples:**

```
# Load tokenizer để run một số example
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
```

(d) **Tokenize dataset:** Trong phần này ta sẽ tiến hành tokenize dataset cho tập train và tập val.

- **Tokenize train set:** Hàm preprocess_training_examples nhận dữ liệu đào tạo làm đầu vào và tiền xử lý để chuẩn bị cho việc huấn luyện mô hình hỏi đáp. Trong quá trình này, hàm trích xuất danh sách câu hỏi, mã hóa thông tin đầu vào bằng tokenizer, và trích xuất offset_mapping và sample_map để ánh xạ vị trí từ mã hóa về văn bản gốc. Hàm cũng xác định vị trí bắt đầu và kết thúc của câu trả lời trong ngữ cảnh và thêm thông tin về vị trí này vào biến inputs.

```
# Định nghĩa hàm preprocess_training_examples và nhận tham số examples
# là dữ liệu training
def preprocess_training_examples(examples):
    # Trích xuất danh sách câu hỏi từ examples và
    # loại bỏ các khoảng trắng dư thừa
    questions = [q.strip() for q in examples["question"]]
```

```
7
8       # Tiến hành mã hóa thông tin đầu vào sử dụng tokenizer
9       inputs = tokenizer(
10          questions,
11          examples["context"],
12          max_length=MAX_LENGTH,
13          truncation="only_second",
14          stride=STRIDE,
15          return_overflowing_tokens=True,
16          return_offsets_mapping=True,
17          padding="max_length",
18      )
19
20      # Trích xuất offset_mapping từ inputs và loại bỏ nó ra khỏi inputs
21      offset_mapping = inputs.pop("offset_mapping")
22
23      # Trích xuất sample_map từ inputs và loại bỏ nó ra khỏi inputs
24      sample_map = inputs.pop("overflow_to_sample_mapping")
25
26      # Trích xuất thông tin về câu trả lời (answers) từ examples
27      answers = examples["answers"]
28
29      # Khởi tạo danh sách các vị trí bắt đầu và kết thúc câu trả lời
30      start_positions = []
31      end_positions = []
32
33      # Duyệt qua danh sách offset_mapping
34      for i, offset in enumerate(offset_mapping):
35          # Xác định index của mẫu (sample) liên quan đến offset hiện tại
36          sample_idx = sample_map[i]
37
38          # Trích xuất sequence_ids từ inputs
39          sequence_ids = inputs.sequence_ids(i)
40
41          # Xác định vị trí bắt đầu và kết thúc của ngữ cảnh
42          idx = 0
43          while sequence_ids[idx] != 1:
44              idx += 1
45          context_start = idx
46          while sequence_ids[idx] == 1:
47              idx += 1
48          context_end = idx - 1
49
50          # Trích xuất thông tin về câu trả lời cho mẫu này
51          answer = answers[sample_idx]
52
53          if len(answer['text']) == 0:
54              start_positions.append(0)
55              end_positions.append(0)
56          else:
57              # Xác định vị trí ký tự bắt đầu và kết thúc của câu trả lời
58              # trong ngữ cảnh
59              start_char = answer["answer_start"][0]
60              end_char = answer["answer_start"][0] + len(answer["text"][0])
61
62              # Nếu câu trả lời không nằm hoàn toàn trong ngữ cảnh,
63              # gán nhãn là (0, 0)
64              if offset[context_start][0] > start_char
65                  or offset[context_end][1] < end_char:
```

```
                start_positions.append(0)
                end_positions.append(0)
            else:
                # Nếu không, gán vị trí bắt đầu và kết thúc dựa trên
                # vị trí của các mã thông tin
                idx = context_start
                while idx <= context_end and offset[idx][0] <= start_char:
                    idx += 1
                start_positions.append(idx - 1)

                idx = context_end
                while idx >= context_start and offset[idx][1] >= end_char:
                    idx -= 1
                end_positions.append(idx + 1)

    # Thêm thông tin vị trí bắt đầu và kết thúc vào inputs
    inputs["start_positions"] = start_positions
    inputs["end_positions"] = end_positions

    return inputs
```

Sau đó ta sẽ chạy đoạn hàm trên với từng dòng trong raw_dataset của tập train:

```
# Tạo một biến train_dataset và gán cho nó giá trị sau khi áp dụng
# hàm preprocess_training_examples lên tập dữ liệu "train"
#
# Bật chế độ xử lý theo từng batch bằng cách đặt batched=True
#
# Loại bỏ các cột không cần thiết trong
# tập dữ liệu "train" bằng cách sử dụng remove_columns

train_dataset = raw_datasets["train"].map(
    preprocess_training_examples,
    batched=True,
    remove_columns=raw_datasets["train"].column_names,
)

# In ra độ dài của tập dữ liệu "train" ban đầu và
# độ dài của tập dữ liệu đã được xử lý (train_dataset)
len(raw_datasets["train"]), len(train_dataset)
```

- **Tokenize val set:** Ta sẽ làm tương tự với tập val, hàm preprocess_validation_examples thực hiện việc tiền xử lý dữ liệu cho quá trình đánh giá mô hình. Hàm chuẩn bị danh sách câu hỏi, mã hóa các câu hỏi và văn bản liên quan bằng cách sử dụng tokenizer. Sau đó xác định ví dụ tham chiếu cho từng dòng đầu vào và điều chỉnh ánh xạ offset để loại bỏ các offset không phù hợp với sequence_ids. Cuối cùng là thêm thông tin về ví dụ tham chiếu vào đầu vào.

```
def preprocess_validation_examples(examples):
    # Chuẩn bị danh sách câu hỏi bằng cách
    # loại bỏ khoảng trắng ở đầu và cuối mỗi câu hỏi
    questions = [q.strip() for q in examples["question"]]

    # Sử dụng tokenizer để mã hóa các câu hỏi và văn bản liên quan
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=MAX_LENGTH,
        truncation="only_second",
        stride=STRIDE,
```

```
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    # Lấy ánh xạ để ánh xạ lại ví dụ tham chiếu cho từng dòng trong inputs
    sample_map = inputs.pop("overflow_to_sample_mapping")
    example_ids = []

    # Xác định ví dụ tham chiếu cho mỗi dòng đầu vào và
    # điều chỉnh ánh xạ offset
    for i in range(len(inputs["input_ids"])):
        sample_idx = sample_map[i]
        example_ids.append(examples["id"][sample_idx])

        sequence_ids = inputs.sequence_ids(i)
        offset = inputs["offset_mapping"][i]

        # Loại bỏ các offset không phù hợp với sequence_ids
        inputs["offset_mapping"][i] = [
            o if sequence_ids[k] == 1 else None \
                for k, o in enumerate(offset)
        ]

    # Thêm thông tin ví dụ tham chiếu vào đầu vào
    inputs["example_id"] = example_ids

    return inputs
```

Ta sẽ chạy đoạn hàm trên với từng dòng trong raw_dataset của tập validation:

```
# Tạo một biến validation_dataset và gán giá trị bằng việc sử dụng dữ liệu
# từ raw_datasets["validation"] sau khi áp dụng một hàm xử lý trước.

validation_dataset = raw_datasets["validation"].map(
    preprocess_validation_examples,
    batched=True,
    remove_columns=raw_datasets["validation"].column_names,
)

# In ra độ dài của raw_datasets["validation"]
# và validation_dataset để so sánh.
len(raw_datasets["validation"]), len(validation_dataset)
```

(e) **Train model:** Sau khi đã chuẩn bị xong dataset, ta sẽ tiến hành load model từ HuggingFace để chuẩn bị cho quá trình training:

```
# Load model
model = AutoModelForQuestionAnswering.from_pretrained(MODEL_NAME)
```

Tiếp theo ta sẽ định nghĩa một số parameter mà ta sẽ sử dụng để training model:

```
# Tạo đối tượng args là các tham số cho quá trình huấn luyện
args = TrainingArguments(
    output_dir="distilbert-finetuned-squadv2", # Thư mục lưu output
    evaluation_strategy="no",  # Chế độ đánh giá không tự động sau mỗi epoch
    save_strategy="epoch",  # Lưu checkpoint sau mỗi epoch
    learning_rate=2e-5,  # Tốc độ học
    num_train_epochs=3,  # Số epoch huấn luyện
    weight_decay=0.01,  # Giảm trọng lượng mô hình để tránh overfitting
```

```
    fp16=True,  # Sử dụng kiểu dữ liệu half-precision để tối ưu tài nguyên
    push_to_hub=True,  # Đẩy kết quả huấn luyện lên HuggingFace Hub
)
```

Cuối cùng ta sẽ khởi tạo class Trainer, đây là class chính để training model, ta sẽ không cần phải định nghĩa hàm train, đưa input vào mode, tính toán loss, update gradient nữa, hàm class này sẽ tự động làm giúp chúng ta. Sau khi đã khởi tạo thì chỉ cần gọi trainner.train() thì quá trình training model sẽ được tiến hành:

```
# Khởi tạo một đối tượng Trainer để huấn luyện mô hình
trainer = Trainer(
    model=model,  # Sử dụng mô hình đã tạo trước đó
    args=args,  # Các tham số và cấu hình huấn luyện
    train_dataset=train_dataset,  # Sử dụng tập dữ liệu huấn luyện
    eval_dataset=validation_dataset,  # Sử dụng tập dữ liệu đánh giá
    tokenizer=tokenizer,  # Sử dụng tokenizer để xử lý văn bản
)

# Bắt đầu quá trình huấn luyện
trainer.train()
```

[49354/49410 1:39:37 < 00:06, 8.26 it/s, Epoch 3.00/3]

Step	Training Loss
500	3.102900
1000	2.268800
1500	1.971300
2000	1.810200
2500	1.702800
3000	1.625000
3500	1.574800
4000	1.566400
4500	1.495000
5000	1.480800
5500	1.438300
6000	1.402200
6500	1.418000
7000	1.426900
45500	0.686000
46000	0.696700
46500	0.674500
47000	0.651100
47500	0.676700
48000	0.651300
48500	0.673300
49000	0.697000

Sau khi quá trình training hoàn tất, ta sẽ đưa weight, config của model lên HuggingFace Hub để lưu lại:

```
# Gửi dữ liệu đào tạo lên Hub
trainer.push_to_hub(commit_message="Training complete")
```

(f) **Evaluate model:** Để đánh giá performance của model ta sẽ sử dụng metric squad từ thư viện evaluate:

```
# Tải metric "squad" từ thư viện evaluate
metric = evaluate.load("squad_v2")
```

Hàm compute_metrics nhận các đầu vào như start_logits, end_logits, features, và examples, và thực hiện các bước sau để tính toán các độ đo và kết quả đánh giá mô hình hỏi đáp. Trong quá trình tính toán, hàm này tạo một danh sách các câu trả lời dự đoán dựa trên các logits được dự đoán bởi mô hình. Điều này bao gồm việc xác định vị trí bắt đầu và kết thúc tốt nhất cho các câu trả lời và đánh giá xem chúng có hợp lệ hay không dựa trên độ dài tối đa cho câu trả lời. Cuối cùng, hàm tính toán các độ đo và trả về kết quả đánh giá mô hình hỏi đáp dựa trên các câu trả lời dự đoán và câu trả lời lý thuyết từ ví dụ.

```
N_BEST = 20 # Số lượng kết quả tốt nhất được lựa chọn sau khi dự đoán
MAX_ANS_LENGTH = 30 # Độ dài tối đa cho câu trả lời dự đoán

def compute_metrics(start_logits, end_logits, features, examples):
    # Tạo một từ điển mặc định để ánh xạ mỗi ví dụ
    # với danh sách các đặc trưng tương ứng
    example_to_features = collections.defaultdict(list)
    for idx, feature in enumerate(features):
        example_to_features[feature['example_id']].append(idx)

    predicted_answers = []
    for example in tqdm(examples):
        example_id = example['id']
        context = example['context']
        answers = []

        # Lặp qua tất cả các đặc trưng liên quan đến ví dụ đó
        for feature_index in example_to_features[example_id]:
            start_logit = start_logits[feature_index]
            end_logit = end_logits[feature_index]
            offsets = features[feature_index]['offset_mapping']

            # Lấy các chỉ số có giá trị lớn nhất cho start và end logits
            start_indexes = np.argsort(start_logit)[-1:-N_BEST-1:-1].tolist()
            end_indexes = np.argsort(end_logit)[-1:-N_BEST-1:-1].tolist()
            for start_index in start_indexes:
                for end_index in end_indexes:
                    # Bỏ qua các câu trả lời
                    # không hoàn toàn nằm trong ngữ cảnh
                    if offsets[start_index] is None or \
                        offsets[end_index] is None:
                        continue
                    # Bỏ qua các câu trả lời có độ dài > max_answer_length
                    if end_index - start_index + 1 > MAX_ANS_LENGTH:
                        continue

                    # Tạo một câu trả lời mới
                    text = context[
                        offsets[start_index][0]:offsets[end_index][1]
                    ]
                    logit_score = start_logit[start_index] + \
                                  end_logit[end_index]
                    answer = {
                        'text': text,
                        'logit_score': logit_score,
                    }
                    answers.append(answer)

        # Chọn câu trả lời có điểm số tốt nhất
        if len(answers) > 0:
            best_answer = max(answers, key=lambda x: x['logit_score'])
```

```
            answer_dict = {
                'id': example_id,
                'prediction_text': best_answer['text'],
                'no_answer_probability': 1 - best_answer['logit_score']
            }
        else:
            answer_dict = {
                'id': example_id,
                'prediction_text': '',
                'no_answer_probability': 1.0
            }
        predicted_answers.append(answer_dict)

    # Tạo danh sách câu trả lời lý thuyết từ các ví dụ
    theoretical_answers = [
        {'id': ex['id'], 'answers': ex['answers']} for ex in examples
    ]
    # Sử dụng metric.compute để tính toán các độ đo và trả về kết quả
    return metric.compute(
        predictions=predicted_answers,
        references=theoretical_answers
    )
```

Sau khi đã định nghĩa hàm evaluation, ta sẽ tiến hành predict model trên tập validation rồi đưa vào hàm compute_metrics:

```
# Thực hiện dự đoán trên tập dữ liệu validation
predictions, _, _ = trainer.predict(validation_dataset)

# Lấy ra thông tin về các điểm bắt đầu và
# điểm kết thúc của câu trả lời dự đoán
start_logits, end_logits = predictions

# Tính toán các chỉ số đánh giá sử dụng hàm compute_metrics
results = compute_metrics(
    start_logits,
    end_logits,
    validation_dataset,
    raw_datasets["validation"]
)
results
```

```
100% 11873/11873 [00:19<00:00, 686.51it/s]
{'exact': 47.452202476206516,
 'f1': 51.28265600122848,
 'total': 11873,
 'HasAns_exact': 74.78070175438596,
 'HasAns_f1': 82.45259357331055,
 'HasAns_total': 5928,
 'NoAns_exact': 20.201850294365013,
 'NoAns_f1': 20.201850294365013,
 'NoAns_total': 5945,
 'best_exact': 64.46559420533984,
 'best_exact_thresh': -11.35546875,
 'best_f1': 66.16170764530801,
 'best_f1_thresh': -9.61328125}
```

(g) **Load model from hub:** Ở phần trước, xong khi training model xong thì ta đã đưa model

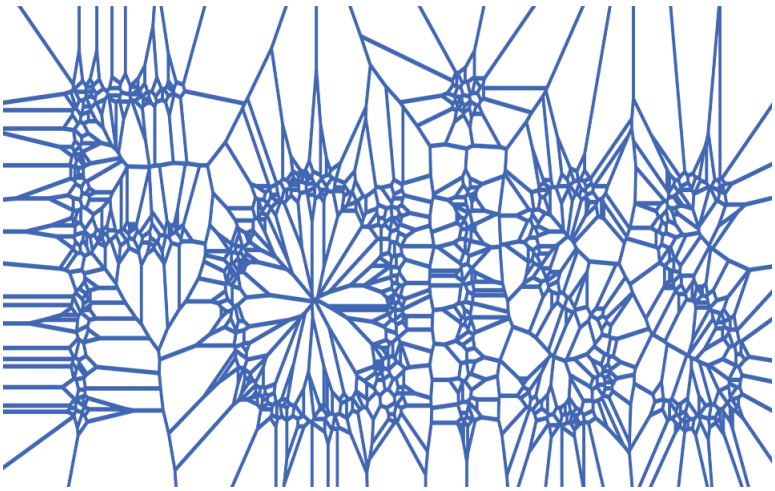
lên HuggingFace, nếu muốn sử dụng thì ta chỉ cần dùng class pipeline có sẵn của HuggingFace là đã có thể load model và tiến hành inference:

```
# Use a pipeline as a high-level helper
from transformers import pipeline

PIPELINE_NAME = 'question-answering'
MODEL_NAME = 'thangduong0509/distilbert-finetuned-squadv2'
pipe = pipeline(PIPELINE_NAME, model=MODEL_NAME)
```

Sau đây ta sẽ chạy thử một example để test model:

```
INPUT_QUESTION = 'What is my name?'
INPUT_CONTEXT = 'My name is AI Vietnam and I live in Vietnam.'
pipe(question=INPUT_QUESTION, context=INPUT_CONTEXT)

## >> Output: {'score': 0.97179114818573, 'start': 11, 'end': 21, 'answer': '
    AI Vietnam'}
```

3. **Retriever: Faiss** (Facebook AI Similarity Search) là một thư viện được phát triển bởi Facebook AI Research Team, hỗ trợ trong việc tìm kiếm tương đồng và phân cụm (clustering) các vector với tốc độ và độ chính xác cao. Các bạn có thể đọc thêm về Faiss tại đây.



Hình 3: Source

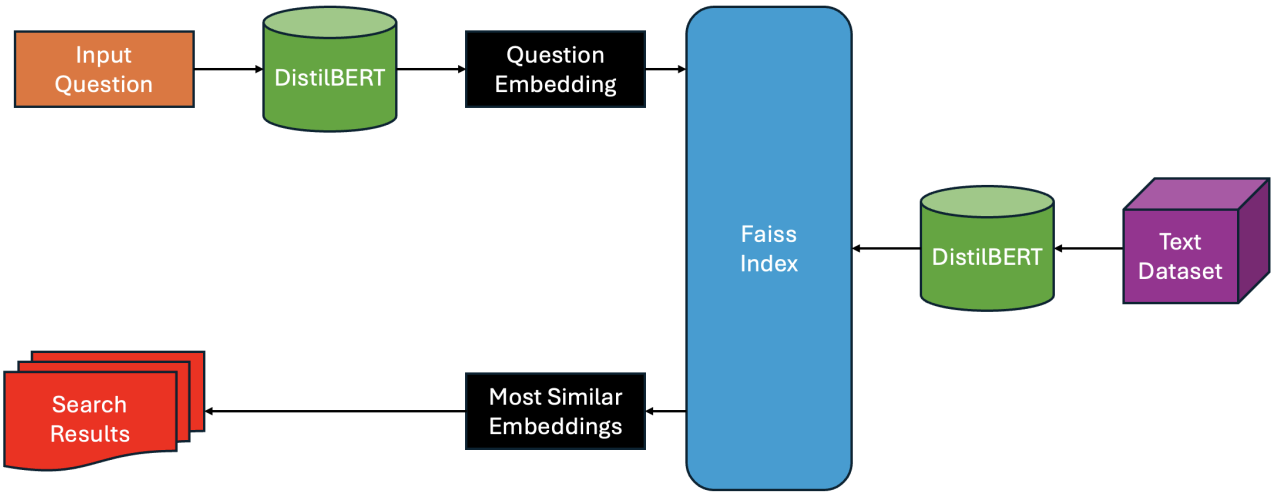
Tại đây, chúng ta sẽ ứng dụng Faiss để làm module Retriever cho hệ thống E2E QA của chúng ta. Với nhiệm vụ tìm kiếm các context phù hợp nhất cho câu hỏi đầu vào, ta sẽ cài đặt Faiss theo cách thức như sau:

(a) Với bộ dữ liệu SQuAD2.0, ta sẽ xây dựng một database chứa thêm cột đại diện cho vector embedding của câu hỏi.

(b) Thực hiện embedding các câu hỏi sử dụng DistilBERT.

(c) Thực hiện tìm kiếm tương đồng giữa các vector trong cột mới và vector câu hỏi đầu vào, từ đó tìm ra nội dung context có liên quan nhất.

Quy trình xử lý của Faiss trong bài có thể được tóm gọn qua ảnh sau:



Hình 4: Minh họa các bước xây dựng một vector database với Faiss

Để cài đặt Faiss phục vụ cho việc tìm kiếm các văn bản context của các câu hỏi có nội dung giống với câu hỏi đầu vào, ta thực hiện như sau:

(a) **Cài đặt và import các thư viện cần thiết:**

```
!pip install -qq transformers[sentencepiece]==4.35.2 datasets==2.16.1
    evaluate==0.4.1
!sudo apt-get install libomp-dev
!pip install -qq faiss-gpu
```

```
import numpy as np
import collections
import torch
import faiss
import evaluate

from datasets import load_dataset
from transformers import AutoTokenizer, AutoModel
from transformers import AutoModelForQuestionAnswering
from transformers import TrainingArguments
from transformers import Trainer
from tqdm.auto import tqdm

device = torch.device("cuda") if torch.cuda.is_available() else torch.device(
    "cpu")
```

(b) **Tải bộ dữ liệu:** Ta tải bộ dữ liệu SQuAD2.0:

```
DATASET_NAME = 'squad_v2'
raw_datasets = load_dataset(DATASET_NAME, split='train+validation')
raw_datasets
```

Để tận dụng toàn bộ ngữ liệu, chúng ta sẽ gom hai bộ dữ liệu train và validation trong bước tạo vector database này.

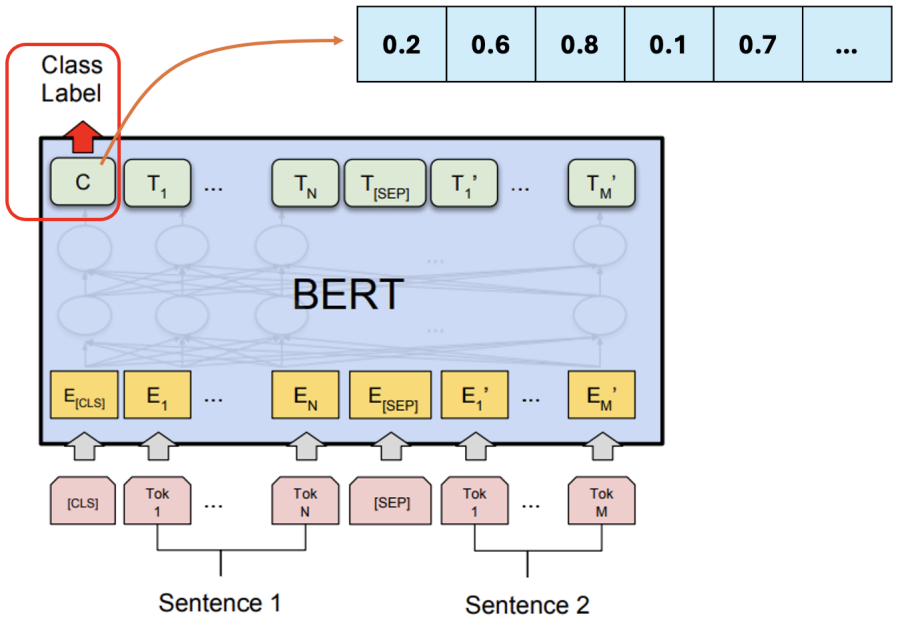
(c) **Loại bỏ các mẫu không có đáp án:** Các mẫu dữ liệu không có đáp án thường chứa các câu hỏi không liên quan đến đoạn văn ngữ cảnh. Vì vậy, ta sẽ loại bỏ các trường hợp này ra khỏi bộ dữ liệu:

```
raw_datasets = raw_datasets.filter(
    lambda x: len(x['answers']['text']) > 0
)
```

(d) **Khởi tạo mô hình:** Để tạo vector embedding cho các câu hỏi, ta sẽ sử dụng mô hình pre-trained DistilBERT:

```
MODEL_NAME = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModel.from_pretrained(MODEL_NAME).to(device)
```

(e) **Xây dựng hàm lấy vector embedding:** Ở đây, ta sẽ xây dựng một hàm trả về vector embedding cho một đoạn text đầu vào, cụ thể ở đây sẽ là câu hỏi. Để tạo vector embedding đại diện cho câu hỏi, ta sẽ sử dụng vector hidden state từ token [CLS] trong kết quả output của mô hình DistilBERT:



Hình 5: Ảnh minh họa vị trí của final hidden state của token CLS.

Đầu tiên, ta xây dựng hàm lấy final hidden state của token CLS:

```
def cls_pooling(model_output):
    return model_output.last_hidden_state[:, 0]
```

Sau đó, xây dựng hàm đưa một văn bản vào model để từ đó có thể gọi hàm `cls_pooling()`:

```
def get_embeddings(text_list):
    encoded_input = tokenizer(
        text_list,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )
    encoded_input = {k: v.to(device) for k, v in encoded_input.items()}
    model_output = model(**encoded_input)

    return cls_pooling(model_output)
```

(f) **Xây dựng vector database:** Với hàm tạo vector embedding đã triển khai, ta sẽ sử dụng nó để tạo một cột trong bảng dữ liệu dùng để chứa kết quả lời gọi hàm `get_embeddings()` với input là các câu hỏi của từng mẫu dữ liệu. Cụ thể, ta tạo cột mới tên là **question_embedding** và lưu vector embedding của câu hỏi như sau:

```
EMBEDDING_COLUMN = 'question_embedding'
embeddings_dataset = raw_datasets.map(
    lambda x: {
        EMBEDDING_COLUMN: get_embeddings(
            x['question']
        ).detach().cpu().numpy()[0]
    }
)
```

Sau đó, để có thể tìm kiếm các vector tương đồng, ta sẽ tạo một cấu trúc dữ liệu đặc biệt là Faiss Index như sau:

```
embeddings_dataset.add_faiss_index(column=EMBEDDING_COLUMN)
```

Cuối cùng, chúng ta đã có một vector database lưu trữ vector embedding của các câu hỏi trong bộ dữ liệu. Từ đây, ta sẽ tiến hành thử thực hiện truy vấn với một câu hỏi đầu vào bất kì như sau:

```
input_question = 'When did Beyonce start becoming popular?'

input_quest_embedding = get_embeddings([input_question])
input_quest_embedding = input_quest_embedding.cpu().detach().numpy()

TOP_K = 5
scores, samples = embeddings_dataset.get_nearest_examples(
    EMBEDDING_COLUMN, input_quest_embedding, k=TOP_K
)

for idx, score in enumerate(scores):
    print(f'Top {idx + 1}\tScore: {score}')
    print(f'Question: {samples["question"][idx]}')
    print(f'Context: {samples["context"][idx]}')
    print()
```

Khi chạy xong đoạn lệnh trên, ta được kết quả trả về như sau:

```
Top 1   Score: 0.0
Question: When did Beyonce start becoming popular?
Context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,

Top 2   Score: 2.6135313510894775
Question: When did Beyoncé rise to fame?
Context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,

Top 3   Score: 4.859482288360596
Question: When did Beyoncé release Formation?
Context: On February 6, 2016, one day before her performance at the Super Bowl, Beyoncé released a new single exclusively on mu

Top 4   Score: 5.054221153259277
Question: In which decade did Beyonce become famous?
Context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,

Top 5   Score: 5.170375347137451
Question: When did Beyonce begin her deals with name brands?
Context: The release of a video-game Starpower: Beyoncé was cancelled after Beyoncé pulled out of a $100 million with GateFive
```

Hình 6: Kết quả truy vấn được in ra màn hình

Có thể thấy, vì câu hỏi đầu vào có tồn tại trong vector database của chúng ta nên mẫu dữ liệu tương đồng nhất cũng chính là mẫu có câu hỏi tương tự.

(g) **Áp dụng mô hình hỏi-đáp để trả lời câu hỏi:** Như vậy, chúng ta đã có hai thành phần Retriever và Reader. Chúng ta sẽ viết một đoạn code ngắn để thực hiện chương trình End-to-End QA. Đầu tiên, khởi tạo mô hình hỏi-đáp đã huấn luyện:

```
from transformers import pipeline

PIPELINE_NAME = 'question-answering'
MODEL_NAME = 'thangduong0509/distilbert-finetuned-squadv2'
pipe = pipeline(PIPELINE_NAME, model=MODEL_NAME)
```

Tận dụng kết quả truy vấn vừa rồi (nằm ở biến `scores` và `samples`), chúng ta sẽ truyền vào mô hình QA hai thông tin gồm question và context:

```
print(f'Input question: {input_question}')
for idx, score in enumerate(scores):
    question = samples["question"][idx]
    context = samples["context"][idx]
    answer = pipe(
        question=question,
        context=context
    )
    print(f'Top {idx + 1}\tScore: {score}')
    print(f'Context: {context}')
    print(f'Answer: {answer}')
    print()
```

```
Input question: When did Beyonce start becoming popular?
Top 1   Score: 0.0
Context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Answer: {'score': 0.6091989278793335, 'start': 276, 'end': 286, 'answer': 'late 1990s'}

Top 2   Score: 2.6135313510894775
Context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Answer: {'score': 0.6103343963623047, 'start': 276, 'end': 286, 'answer': 'late 1990s'}

Top 3   Score: 4.859482288360596
Context: On February 6, 2016, one day before her performance at the Super Bowl, Beyoncé released a new single exclusively on mu
Answer: {'score': 0.9715896248817444, 'start': 3, 'end': 19, 'answer': 'February 6, 2016'}

Top 4   Score: 5.054221153259277
Context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Answer: {'score': 0.7218101620674133, 'start': 281, 'end': 286, 'answer': '1990s'}

Top 5   Score: 5.170375347137451
Context: The release of a video-game Starpower: Beyoncé was cancelled after Beyoncé pulled out of a $100 million with GateFive
Answer: {'score': 0.6330415606498718, 'start': 433, 'end': 452, 'answer': 'since the age of 18'}
```

Hình 7: Kết quả E2E QA được in ra màn hình

Phần III: Câu hỏi trắc nghiệm

1. So với QA, chương trình End-to-end QA có điểm gì khác biệt?
   (*a*) Mô hình trích xuất câu hỏi tốt hơn.
   (*b*) Sử dụng kiến trúc transformer.
   (*c*) Có sử dụng mô hình tìm kiếm context.
   (*d*) Tốc độ xử lý nhanh hơn.

2. Tại sao mô hình Transformer được sử dụng phổ biến trong bài toán Question Answering (QA)?
   (*a*) Do Transformer có khả năng tự học đặc trưng từ văn bản tự nhiên.
   (*b*) Do Transformer chứa nhiều kiến thức về dữ liệu.
   (*c*) Có sử dụng mô hình tìm kiếm context.
   (*d*) Do Transformer có khả năng xử lý dữ liệu dạng sequence.

3. Trong QA, tại sao phải sử dụng Transfer learning?
   (*a*) Transfer learning giúp mô hình học được kiến thức từ dữ liệu lớn.
   (*b*) Mô hình QA không cần sử dụng transfer learning.
   (*c*) Transfer learning làm gia tăng khả năng xử lý của CPU.
   (*d*) Transfer learning giúp mô hình bị overfitting.

4. Mô hình End-to-end QA khác biệt với QA truyền thống ở điểm nào?
   (*a*) Mô hình End-to-end QA sử dụng mô hình tìm kiếm context.
   (*b*) Mô hình End-to-end QA có khả năng tự động xây dựng câu hỏi.
   (*c*) Mô hình End-to-end QA sử dụng RNN để trích xuất câu trả lời.
   (*d*) Mô hình End-to-end QA chỉ hoạt động trên dữ liệu có cấu trúc.

5. Tham số stride có nghĩa là gì trong đoạn code sau:

```
inputs = tokenizer(
    question,  # Danh sách các câu hỏi
    context,  # Nội dung liên quan đến câu hỏi
    max_length=MAX_LENGTH,  # Độ dài tối đa cho đầu ra mã hóa
    truncation="only_second",  # Cắt bớt dữ liệu chỉ cho phần thứ hai (context)
    stride=STRIDE,
    return_overflowing_tokens=True,  # Trả về các tokens vượt quá độ dài tối đa
    return_offsets_mapping=True,  # Trả về bản đồ vị trí của các tokens trong văn
     bản gốc
    padding="max_length" # Điền vào dữ liệu để có cùng độ dài max_length
)
```

   (*a*) Độ dài bước nhảy trong trường hợp dữ liệu dài hơn max_length
   (*b*) Độ dài bước nhảy trong trường hợp dữ liệu ngắn hơn max_length
   (*c*) Độ dài bước nhảy trong trường hợp dữ liệu bằng max_length
   (*d*) Độ dài bước nhảy trong trường hợp bất kỳ

6. Tham số fp16=True có nghĩa là gì trong đoạn code sau:

```
# Tạo đối tượng args là các tham số cho quá trình huấn luyện
args = TrainingArguments(
    output_dir="distilbert-finetuned-squadv2",  # Thư mục lưu trữ kết quả huấn luyện
    evaluation_strategy="no",  # Chế độ đánh giá không tự động sau mỗi epoch
    save_strategy="epoch",  # Lưu checkpoint sau mỗi epoch
    learning_rate=2e-5,  # Tốc độ học
    num_train_epochs=3,  # Số epoch huấn luyện
    weight_decay=0.01,  # Giảm trọng lượng mô hình để tránh overfitting
    fp16=True,
    push_to_hub=True,  # Đẩy kết quả huấn luyện lên một nơi chia sẻ trực tuyến (Hub)
)
```

(*a*) Sử dụng kiểu dữ liệu 32-bit để tối ưu hóa tài nguyên

(*b*) Sử dụng kiểu dữ liệu double để tối ưu hóa tài nguyên

(*c*) Sử dụng kiểu dữ liệu float để tối ưu hóa tài nguyên

(*d*) Sử dụng kiểu dữ liệu half-precision để tối ưu hóa tài nguyên

7. Trong đoạn code sau đây, biến PIPELINE_NAME dùng để làm gì?

```
# Use a pipeline as a high-level helper
from transformers import pipeline

PIPELINE_NAME = 'question-answering'
MODEL_NAME = 'thangduong0509/distilbert-finetuned-squadv2'
pipe = pipeline(PIPELINE_NAME, model=MODEL_NAME)
```

(*a*) Xác định tên của task hiện tại, người dùng có thể đặt tên bất kỳ

(*b*) Xác định tên của task hiện tại, người dùng phải đặt đúng tên quy định của HuggingFace

(*c*) Tên của model, người dùng phải đặt tên đúng yêu cầu

(*d*) Tên của model, người dùng có thể đặt bất kỳ

8. Ưu điểm của vector database khi xử lý các loại dữ liệu phức tạp như hình ảnh, âm thanh và văn bản so với cơ sở dữ liệu quan hệ truyền thống là gì?

(*a*) Nó cung cấp khả năng chuẩn hóa dữ liệu và ràng buộc tính toàn vẹn tốt hơn.

(*b*) Nó cho phép tìm kiếm và truy vấn dữ liệu dựa trên nội dung một cách hiệu quả.

(*c*) Nó cung cấp các cơ chế kiểm soát giao dịch mạnh mẽ hơn.

(*d*) Nó tăng cường khả năng truy vấn SQL cho phân tích dữ liệu có cấu trúc.

9. Để tính sự tương đồng giữa hai vector, độ đo nào sau đây không thể áp dụng?

(*a*) Cosine Similarity

(*b*) Euclidean Distance

(*c*) Pearson Correlation Coefficient

(*d*) Maximum Likelihood Estimation

10. Khi ứng dụng BERT để tạo vector embedding trong vector database, final hidden state của token nào thường được sử dụng trong output của BERT?

(*a*) `[CLS]` token

(*b*) `[SEP]` token

(*c*) `[EOS]` token

(*d*) Final token

11. Trong các vector database như Faiss, kỹ thuật nào thường được sử dụng để tối ưu hiệu quả tìm kiếm trên dữ liệu đa chiều?

(*a*) Linear Search

(*b*) Indexing

(*c*) Quantization

(*d*) Encryption

12. Trong đoạn code dưới đây:

```
TOP_K = 5
scores, samples = embeddings_dataset.get_nearest_examples(
    EMBEDDING_COLUMN, input_quest_embedding, k=TOP_K
)
```

Biến `TOP_K` còn được hiểu là?

(*a*) Số lượng cluster

(*b*) Số lượng kết quả trả về

(*c*) Số epochs

(*d*) Số chiều trong không gian embedding

- Hết -