Document: chatbot

Bài toán: xây dựng chatbot phục vụ cho công việc để người dùng hỏi về thông tin có trong database

Thành phần:

+Chatbot: mô hình ngôn ngữ lớn (LLM)

+SQL Database: cơ sở dữ liệu SQL có sẵn

Hướng làm chính:

TEXT-TO-SQL Model

***Nội dung chính:** finetuning lại một model LLM có sẵn để thực hiện công việc trả lời câu hỏi người dùng bằng SQL query, SQL query khi model output ra sẽ được query trực tiếp trên server database và trả lại số liệu mong muốn cho người dùng.

***Thực hiện:**

1.Chọn một mô hình ngôn ngữ lớn (LLM model):

*__Mục đích__: một mô hình ngôn ngữ lớn mã nguồn mở có sẵn khả năng trả lời câu hỏi người dùng một cách tự nhiên, (chưa xét đến độ chính xác cũng như chất lượng câu trả lời)

***Yêu cầu:**

a) Chỉ để sử dụng, chưa tính đến việc finetuning

+)một mô hình mã nguồn mở có thể dễ dàng tải về và chạy trên local

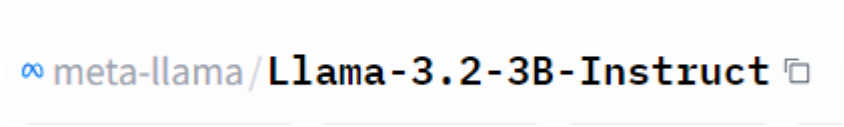
+)căn cứ vào tình trạng của server, chất lượng phần cứng để có thể chọn model phù hợp

Ví dụ: với server chạy thuần trên cpu, không có gpu hay gpu không đủ mạnh (<=3070), nên chọn những model có số lượng tham số nhỏ, 1->3B parameters

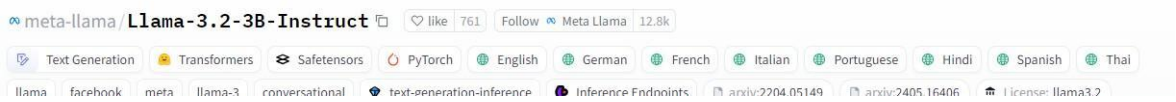
+)thời gian chạy riêng của model loại này:

a) có gpu tương thích + mạnh : dao động tầm 1->3s
b) có gpu tương thích + yếu (vì không phải mọi gpu đều hỗ trợ cuda): dao động tầm 3->15s
c) chỉ dùng cpu: 20->30s

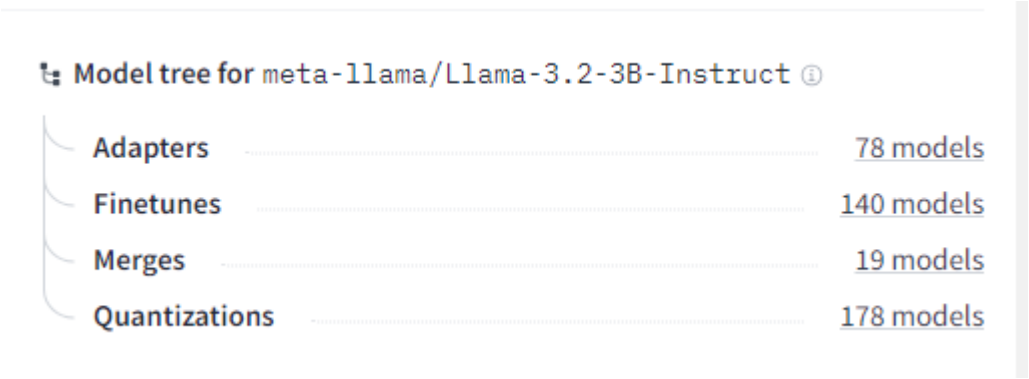
+)model đã được train bằng ngôn ngữ bản địa (ngoài tiếng anh): vì những mô hình phổ biến và nổi tiếng thường trả lời tốt khi người dùng hỏi tiếng anh, còn những ngôn ngữ khác thì rất hên xui, nên cần để ý xem model đó có được train bằng ngôn ngữ mình cần hỏi chưa, không tốn công sức finetuning lại

Ví dụ: 

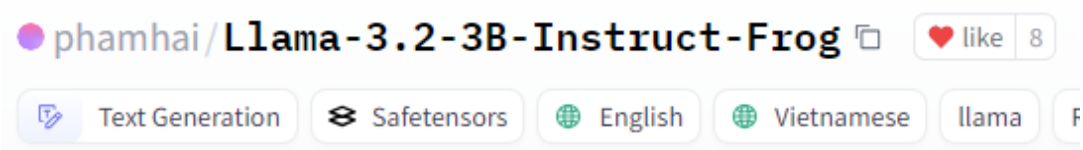
model llama 3B-Instruct có số lượng tham số trong khoảng 1->3B và có đuôi "Instruct"<có nghĩa là model này chuyên dùng cho task QuestionAnswering>. Tuy nhiên model này được nhà phát triển ghi rõ được train cho những ngôn ngữ nào:



Với những model llm được những nhà phát triển lớn tạo ra sẽ có phần model tree, chúng ta có thể tìm những model khác có khả năng tương đương và có thể hỗ trợ những ngôn ngữ khác ở phần finetunes



Ở đây, mình chọn mô hình llama3 của phamhai, đã được finetuned ra tiếng việt:



b) Sử dụng và cần finetuning lại:

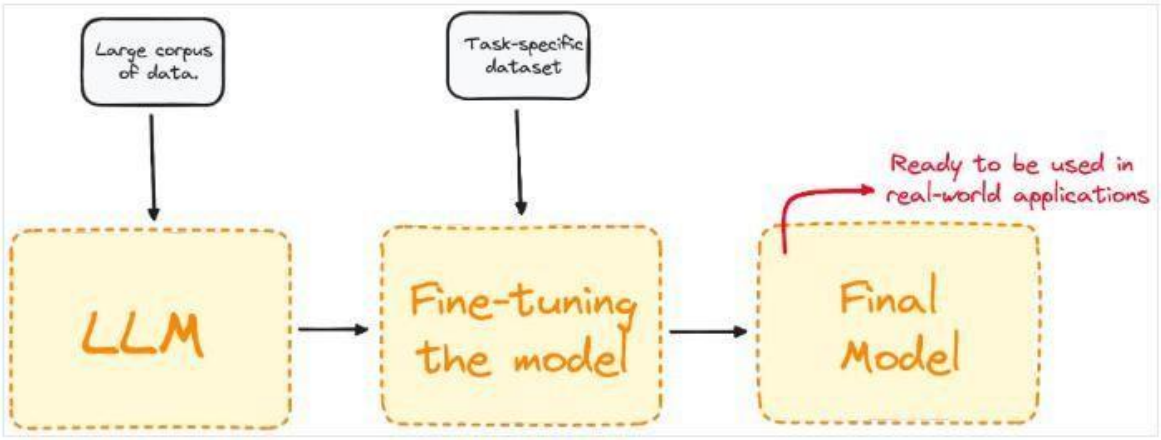
-để finetune được model thì điều kiện tiên quyết là phải có gpu hỗ trợ cuda, gpu càng mạnh thì việc finetuned sẽ càng nhanh hơn <chi tiết ở phần 3.Finetune model>

2. Finetuning model

Finetuning là gì?

-Finetuning là quá trình huấn luyện một mô hình đã được huấn luyện trước đó để chúng có khả năng thực hiện một tác vụ cụ thể.

2.1. Chuẩn bị dữ liệu huấn luyện:



**Điểm lợi của mô hình có nhiều tham số:*

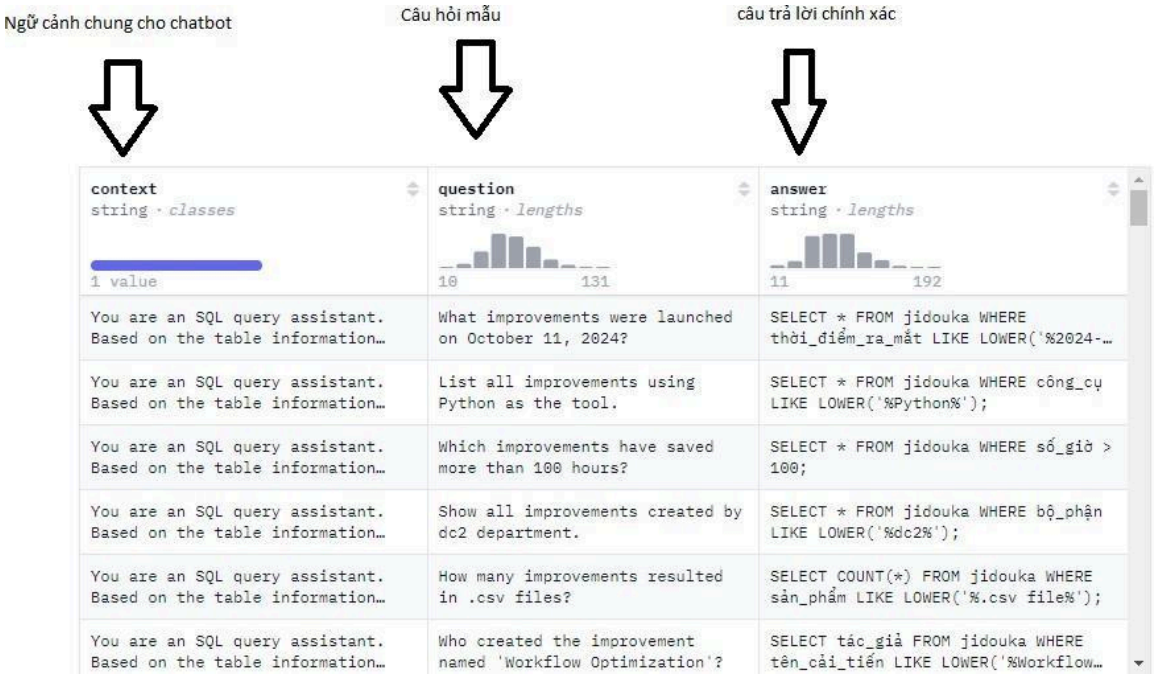
-Số lượng tham số đại diện cho khả năng học hỏi của model, model có số lượng tham số càng lớn sẽ có khả năng cao hơn trong việc học hỏi hiệu quả lượng dữ liệu lớn, đánh đổi với việc đó là lượng dữ liệu cần để model học cũng như thời gian training cũng tăng lên

****các loại dataset:***

-Tùy vào nhiệm vụ mà mình muốn trang bị cho chatbot mà việc chọn dataset để finetuning cũng rất quan trọng.

-Một bộ dataset cơ bản để train mô hình ngôn ngữ:

<Fine tune chatbot để trả lời sql query>



-Hay như khi muốn tăng cường khả năng nhận thức context của câu hỏi đằng trước cho chatbot, mình tạo ra bộ dataset như sau:

#Hướng dẫn tạo dataset tự động bằng ChatGPT:

Bước 1 – Xác định cấu trúc của dataset muốn tạo:

-Ví dụ đơn giản: tạo một bộ dataset cơ bản để train mô hình LLM thành SQL bot, bộ data sẽ có cấu tạo như sau:

+) Context: định nghĩa ngữ cảnh cho Chatbot, ví du: You are an SQL query assistant. Based on the table information below, generate an SQL query to retrieve the relevant information for the user. If the user's question is unrelated to the table, respond naturally in user's language.

+)Question: câu hỏi của người dung

+)Answer: câu trả lời đúng cho câu hỏi đó

Bước 2 – Setup prompt cho chatGPT:

**Define role for chatbot:

Role: Bạn là một chuyên gia về viết các cây truy vấn SQL với cơ sở dữ liệu mySQL.

**Context<yêu cầu mình đặt ra>:

Context: Tôi muốn có một tập dataset để huấn luyện một mô hình LLM từ ngôn ngữ tự nhiên dựa vào cấu trúc sau:

---table Job: ....

---table Department: ....

**Action<Hành động mình muốn nó làm>:

Action: hãy đóng vai người dung muốn truy xuất các thông tin từ bảng trên. Đặt câu hỏi bằng ngôn ngữ tự nhiên và tự trả lời câu hỏi đó bằng SQL querry. Answer: Trả lời theo format python list:

[question (human questions in natural language), answer (SQL querries (viết liền không xuống dòng) or your answer)]

*Note: có thể viết theo nhiều cấu trúc khác nhau, miễn đảm bảo làm rõ các phần: yêu cầu đặt ra, cấu trúc database(vì đây là lĩnh vực truy vấn database) , hành động mình muốn ChatGPT thực hiện, format output cần trả ra.

2.2.Chuyển thành input cho model

Khi có bộ question_answer phù hợp, điều tiếp phải làm là:

+)preprocess data (loại bỏ các kí tự không cần thiết)

+)convert thành dataframe (columns: Context, Question, Answer)

+)export ra csv file

<Notebook tạo dataset cơ bản>

+)Thư viện datasets của Python (thư viện tương thích với những model được xây dựng dựa trên nền tảng transformers)

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("csv", data_files="my_file.csv")
```

+)apply dữ liệu vào chat_template của model

```python
def format_data_template(sample):
    chat = [
          {"role":"system", "content": sample['context']},
          {"role":"user", "content":sample['question']},
          {"role":"assistant","content":sample['answer']}
    ]
    return {
        "messages": tokenizer.apply_chat_template(chat, tokenize=False, add_generation_prompt=True)
    }
```

Giải thích:

- mỗi model ngôn ngữ được tạo ra sẽ đi cùng với 1 tokenizer, dễ hiểu là một phương thức để convert ngôn ngữ con người thành dạng kí tự mà model ngôn ngữ hiểu được.
- Vậy nên trước khi muốn toàn bộ dữ liệu được cho vào model, ta phải dùng phương thức apply_chat_template để convert 1 record context, question và answer thành dạng mà tốt nhất để huấn luyện mô hình

Một record gồm question, answer và context sẽ có dạng:

```
"<|begin_of_text|><|start_header_id|>system<|end_header_id|>\n\nCutting Knowledge Date: December 2023\nToday Date:
06 Dec 2024\n\nYou are an SQL query assistant. Based on schema and history conversation below, generate an SQL que
ry to retrieve the relevant information for the user. If the user's question is unrelated to the table, respond na
turally in user's language.\n\nSchema:\n+Table Author, columns=[AuthorId: int, AuthorName: nvarchar(255), Departme
ntId: int, GroupDCId: int]\n+Table Department, columns=[DepartmentId: int, DepartmentName: nvarchar(255)]\n+Table
GroupDC, columns=[GroupDCId: int, DepartmentId: int, GroupDCName nvarchar(255)]\n+Table Job, columns=[JobId: int,
JobName: nvarchar(255)]\n+Table Tool, columns=[ToolId: int, ToolName: nvarchar(255), ToolDescription: text]\n+Tabl
e Jidouka, columns=[JidoukaId: bigint, ProductApply: nvarchar(255), ImprovementName: nvarchar(255), SoftwareUsing:
nvarchar(255), Description: nvarchar(255), Video: text, DetailDocument: text, TotalJobApplied: int, TotalTimeSave
d: int, DateCreate: datetime, JobId: int, AuthorId: int, DepartmentId: int, GroupDCId: int]\n+Table JidoukaTool, c
olumns=[JidoukaId: bigint, ToolId: int]\n+Primary_keys=[Author.AuthorId, Department.DepartmentId, GroupDC.GroupDCI
d, Job.JobId, Tool.ToolId, Jidouka.JidoukaId]\n+Foreign_keys=[GroupDC.DepartmentId=Department.DepartmentId, Jidouk
a.JobId=Job.JobId, Jidouka.AuthorId=Author.AuthorId, Jidouka.DepartmentId=Department.DepartmentId, Jidouka.GroupDC
Id=GroupDC.GroupDCId, JidoukaTool.JidoukaId=Jidouka.JidoukaId, JidoukaTool.ToolId=Tool.ToolId, Author.DepartmentId
=Department.DepartmentId, Author.GroupDCId=GroupDC.GroupDCId]]\n\nHistory conversation:\nPrevious user question: D
anh sách các nhóm thuộc bộ phận DC5 đã thực hiện cải tiến vào sản phẩm nào?\nPrevious answer: SELECT DISTINCT Grou
pDC.GroupDCName, Jidouka.ProductApply FROM GroupDC JOIN Jidouka ON GroupDC.GroupDCId = Jidouka.GroupDCId JOIN Depa
rtment ON Jidouka.DepartmentId = Department.DepartmentId WHERE Department.DepartmentName LIKE LOWER('%DC5%');\nPre
vious schema linking(Format: [Tables, Columns, Foreign keys, Possible cell values]): {Tables: [GroupDC, Jidouka, D
epartment], Columns: [GroupDC.GroupDCName, Jidouka.ProductApply, GroupDC.GroupDCId, Department.DepartmentId, Depar
tment.DepartmentName], Foreign keys: [Jidouka.GroupDCId = GroupDC.GroupDCId, Jidouka.DepartmentId = Department.Dep
artmentId], Possible cell values: [DC5]}<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCó bao nhiêu cải tiế
n đã được thực hiện vào các sản phẩm này?<|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\nSELECT COUNT(D
ISTINCT Jidouka.ProductApply) FROM Jidouka JOIN GroupDC ON Jidouka.GroupDCId = GroupDC.GroupDCId JOIN Department O
N Jidouka.DepartmentId = Department.DepartmentId WHERE Department.DepartmentName LIKE LOWER('%DC5%');<|eot_id|><|s
tart_header_id|>assistant<|end_header_id|>\n\n"
```

(Chú ý: đây là biểu hiện để mình dễ mường tượng, sau đó còn một bước nữa là chuyển hết đống kí tự này thành dạng số)

2.3 Huấn luyện mô hình

*__Bước 1__: tải mô hình đi kèm với tokenizer mình cần dùng để Train:

```python
# Load model
model = AutoModelForCausalLM.from_pretrained(
    base_model,
    quantization_config=bnb_config,
    device_map="auto",
    torch_dtype=torch.float32,
    #attn_implementation=attn_implementation
)
tokenizer = AutoTokenizer.from_pretrained(base_model,use_fast=True)
tokenizer.padding_side = 'right' # to prevent warnings
```

Note: Đối với những mô hình có số lượng tham số lớn như những mô hình dùng để làm chatbot mà ta finetune trên một phần cứng có giới hạn, có một method để giảm độ lớn của model là **quantizing** chúng về dạng nhỏ hơn sao cho độ thông minh của model không giảm <đọc thêm: quantization >

Function hỗ trợ quantization: ***BitsAndBytesConfig***

```python
# QLoRA config
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True, bnb_4bit_use_double_quant=True, bnb_4bit_quant_type="nf4", bnb_4bit_compute_dtype=torch.float16
)
```

Giải thích sơ qua: model bình thường thường được load ở 8bit hoặc 16bit, khi set load_in_4bit=True, model giảm được số bit phải load. Hơn nữa, model gốc thường có kiểu dữ liệu là torch.float32, set kiểu dữ liệu thành torch.float16 giúp model đỡ phức tạp

Bước 2*: định nghĩa tham số đưa vào mô hình: dùng hàm ***TrainingArguments*:

```python
training_arguments = TrainingArguments(
    output_dir=new_model,
    per_device_train_batch_size=4,    #default: 2
    per_device_eval_batch_size=4,     #default: 2
    gradient_accumulation_steps=4,
    optim="adamw_8bit",
    num_train_epochs=20,
    eval_strategy="epoch",
    eval_steps=0.2,
    save_strategy='epoch',
    logging_steps=1,
    warmup_steps=10,
    logging_strategy="steps",
    learning_rate=2e-4,
    fp16=False,
    bf16=True,
    group_by_length=True,
    report_to="wandb",
    load_best_model_at_end = True
)
```

Các tham số quan trọng là:

- output_dir: định nghĩa đường dẫn để lưu mô hình
- num_train_epochs: số epochs mình dùng để train hết 1 lượt dữ liệu
- load_best_model_at_end = True: để có thể dùng **early stoppin**g- một kĩ thuật giúp ta save được trạng thái model tốt nhất với thời gian train hợp lí một cách tự động

Kỹ thuật Early Stopping < Early Stopping >:

-Sử dụng EarlyStoppingCallback:

```
early_stopping_callback = EarlyStoppingCallback(
    early_stopping_patience=5
)
```

Bước 3: Định nghĩa Trainer

-Có 2 loại trainer phổ biến dùng để huấn luyện mô hình ngôn ngữ là Trainer và SFTTTrainer

a)Trainer:

- Dùng để huấn luyện từ đầu mô hình ngôn ngữ những task như là text classification, question answering hay summarization
- Đi kèm với đó là cần một lượng dữ liệu lớn để có thể huấn luyện mô hình một cách hiệu quả

b)SFTTrainer:

- Tối ưu cho finetuning pretrained model với một lượng datasets nhỏ
- Hiệu quả về việc sử dụng bộ nhớ: hỗ trợ dùng nnhững kĩ thuật optimize tham số như PEFT để giảm memory tiêu thụ trong quá trình training
- Thời gian train ít hơn việc sử dụng Trainer

*Như trong trường hợp finetune LLM để trở thành text2SQL model, mình dùng 1 pretrained model, với lượng dữ liệu và phần cứng vừa phải, mình dùng SFTTrainer để mang lại hiệu quả tốt nhất:

```
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_set,
    eval_dataset = val_set,
    dataset_text_field = 'messages',
    max_seq_length = 2048,
    peft_config = peft_config,
    packing=False,
    args = training_arguments,
    callbacks=[early_stopping_callback]
)
```

SFTTrainer-Các tham số quan trọng:

- Model: mô hình mình đã load trước đó
- Tokenizer: phương thức tokenize được load cùng mô hình trước đó
- train_dataset: dữ liệu dùng để train <bắt buộc>
- eval_dataset: dữ liệu để mô hình đánh giá lúc train <optional>
- args: pass biến mà mình đã dùng TrainingArguments define trước đó
- callbacks: pass biến đã define trước đó để sử dụng early stopping method .

3. Merge và Save model:

-file model sau khi được trainer huấn luyện xong được gọi là **adapter model**

<u>*Giải thích về adapter model:</u>

-Hiểu đơn giản khi mô hình được huấn luyện, thay vì điều chỉnh toàn bộ tham số như Trainer làm, thì SFTTrainer sẽ chỉ căn chỉnh trên một lượng nhỏ tham số hợp lí, và phần căn chỉnh ấy được output ra gọi là adapter model.

-Kích thước của adapter model thường rất nhỏ so với model gốc

-Để sử dụng được thành quả sau khi thực hiện fine tuning, chúng ta phải làm thêm một bước nữa là merge adapter model với model gốc.

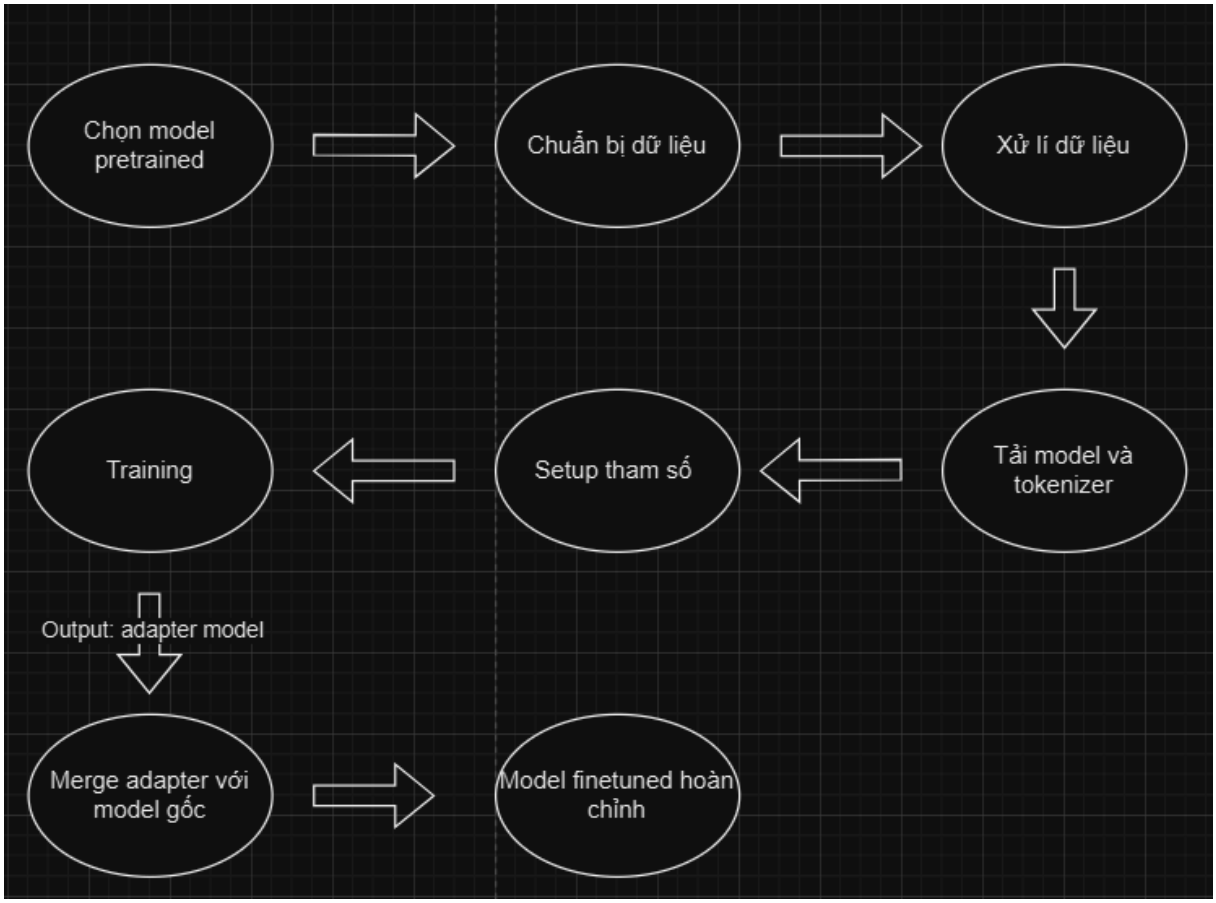
```
tokenizer = AutoTokenizer.from_pretrained(base_model)

base_model_reload = AutoModelForCausalLM.from_pretrained(
        base_model,
        return_dict=True,
        low_cpu_mem_usage=True,
        torch_dtype=torch.float16,
        device_map="auto",
        trust_remote_code=True,
)

# base_model_reload, tokenizer = setup_chat_format(base_model_reload, tokenizer)

# Merge adapter with base model
merge_model = PeftModel.from_pretrained(base_model_reload, adapter_model)

merge_model = merge_model.merge_and_unload()
```

-Bước cuối cùng là lưu model:

```
merged_model = '1b_multiJidouka_contextaware_merged'
merge_model.save_pretrained(merged_model)
tokenizer.save_pretrained(merged_model)
```

Notebook:

Tổng kết:

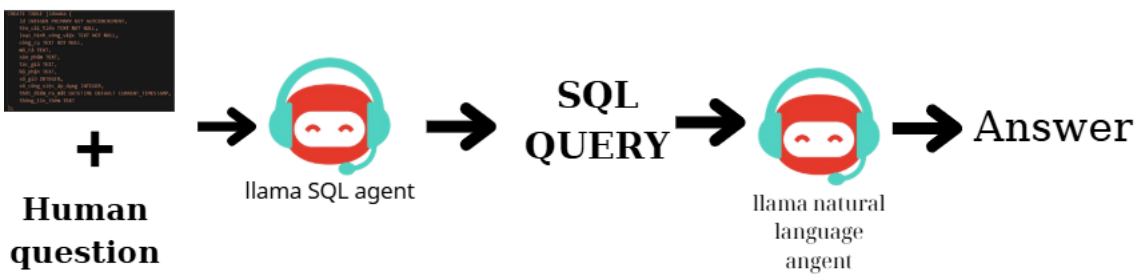


4. Kết hợp multi Agent

-Hiểu đơn giản là một model đơn lẻ sẽ không thể làm tốt mọi việc, việc kết hợp sử dụng những model khác giúp cải thiện độ hiệu quả của hệ thống

<drawbacks: thời gian truy vấn xử lí sẽ lâu hơn do phải dùng nhiều model hợn>

-Ví dụ như trong việc tạo text2SQL model:



-Ở đây mình dùng môt con finetuned SQL để có thể tạo ra SQL để truy vấn câu hỏi người dùng, sau đó , kết quả truy vấn SQL sẽ được một model ngôn ngữ tự nhiên khác đọc hiểu và truyền đạt lại thành answer dễ hiểu.

5. Triển khai chatbot trên docker

-Để thuận tiện cho việc export API sử dụng cho đa nền tảng, đặc biệt là nền tảng web, cách tốt nhất là đóng gói tất cả những gì cần thiết để model chạy vào một docker image.

5.1.Cấu trúc của 1 image chatbot

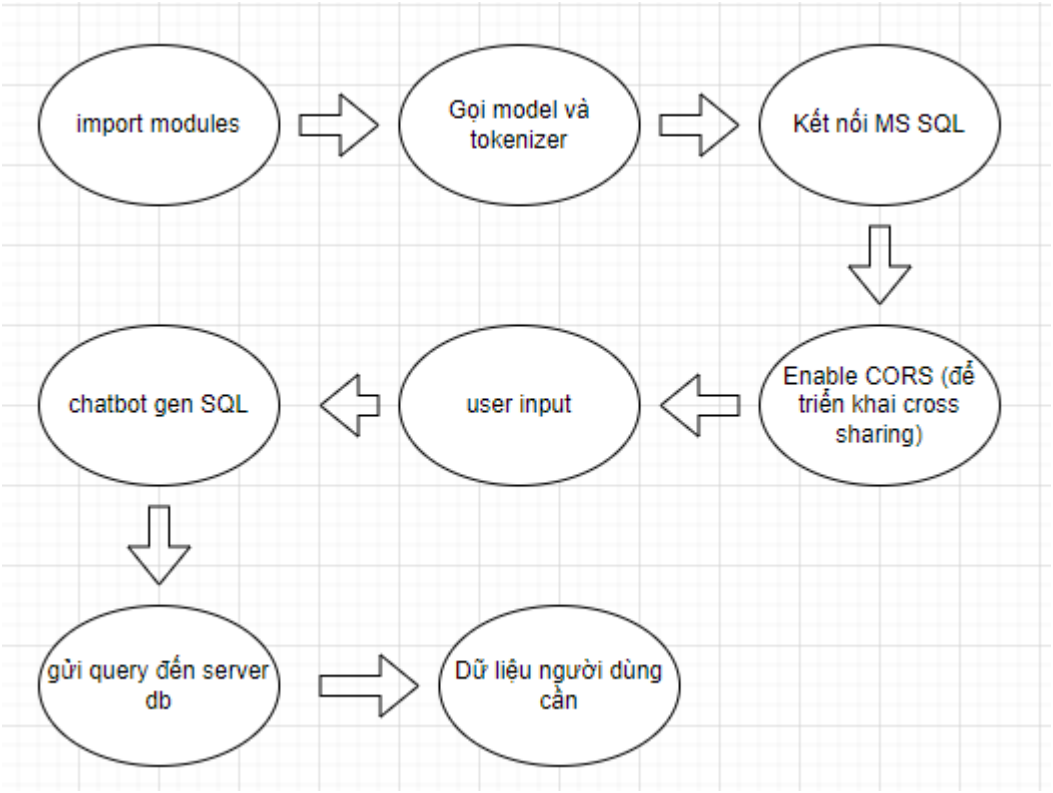
1. **llama3.2_1b** (tên model): folder chứa model và tokenizer
2. **static** (optional): giao diện web, bao gồm index.html, style.css or script.js
3. **Dockerfile**: file config phục vụ để build docker image

**Giải thích:*

```
FROM python:3.9  #tải môi trường python 3.9

RUN useradd -m -u 1000 user     #tạo user id (optional)
WORKDIR /app    #định nghĩa directory làm việc chính là /app

COPY --chown=user ./requirements.txt requirements.txt  #copy file requirements.txt

RUN apt-get update \                    #tải những thư viện cần thiết
 && apt-get install unixodbc -y \       #thư viện cần thiết để kết nối python với MS SQL server
 && apt-get install unixodbc-dev -y \
 && apt-get install freetds-dev -y \
 && apt-get install freetds-bin -y \
 && apt-get install tdsodbc -y \
 && apt-get install --reinstall build-essential -y

RUN echo "[FreeTDS]\n\                  #cấu hình driver để kết nối python với MS SQL server
Description = FreeTDS Driver\n\
Driver = /usr/lib/x86_64-linux-gnu/odbc/libtdsodbc.so\n\
Setup = /usr/lib/x86_64-linux-gnu/odbc/libtdsS.so" >> /etc/odbcinst.ini

RUN pip install numpy==1.26.3           #tải phiên bản numpy tương thích với môi trường
RUN pip install --no-cache-dir --upgrade -r requirements.txt #tải module cần thiết

RUN pip install Jinja2

COPY --chown=user . /app        #copy những file còn lại vào /app
USER user
ENV HOME=/home/user \
        PATH=/home/user/.local/bin:$PATH                #set path môi trường cho user

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "7860"]
```

4.**main.py**: file backend xử lí logic cho chatbot

*__**sơ đồ tổ chức file main.py:__*



5. **requirements.txt:** định nghĩa các module cần thiết để model hoạt động:

```
fastapi==0.74.*
requests==2.27.*
sentencepiece==0.1.*
torch==1.11.*
transformers==4.*
uvicorn[standard]==0.17.*
accelerate
pandas
pyodbc
pydantic
```

**_Giải thích:_

- fastapi (recommended): framework xử lí backend, nhẹ và nhanh hơn Flask, thích hợp để triển khai khi dùng mô hình ngôn ngữ lớn
- torch, transformers: frameworks hỗ trợ để chạy model chatbot
- uvicorn: hỗ trợ expose API cho môi trường product
- pyodbc: giúp connect môi trường python với MS SQL server

5.2.Các lệnh build cơ bản

-Đầu tiên trỏ terminal đến thư mục cần để build

```
PS C:¥Users¥dmvns00008> cd D:/api
PS D:¥api>
```

-Build Image:

+)**chatbot**: tên đặt cho image

+)sử dụng hết các thư mục trong api folder để build image, bao gồm Dockerfile, requirements.txt, folder của model, main,py

-Run Image:

+)**7860:7860** : định nghĩa cổng port khi export api

+)**chatbot**: tên image vừa build

5.3.Triển khai docker image trên centos

-Trước khi có thể chạy docker image trên server CentOs, gần save image về thành file .docker:

+)**chatbot:latest**: tên của image với latest là dùng phiên bản mới nhất

+)**chatbot.docker**: tên file .docker người dùng muốn extract

-Câu lệnh trên extract image ra thành một file .docker, việc cần làm tiếp theo là copy file .docker ấy lên server CentOs

-Khi ở môi trường centos, trỏ tới thư mực chứa file .docker

-Load image từ docker file:

-Bước tiếp theo: sử dụng câu lệnh docker run như trên.

5.4.Sửa đổi file khi docker container đang chạy

-Dừng docker container đang chạy:

```
$ docker stop my_container
```

+) my_container: tên hoặc id của container đang chạy

-Copy file đã chỉnh sửa ở máy tính local lên container trên server:

```
$ docker cp ./some_file CONTAINER:/work
```

+)./some_file: file đã chỉnh sửa cần cop vào trong container

+)CONTAINER: tên hoặc id của container

+)/work: thư mực chứa file cần sửa trên môi trường container

-restart lại container:

```
$ docker restart my_container
```

+)my_container: tên hoặc id của container

5.5.Tại sao không tải model sau khi docker container đã run

-Việc build image khi không có file model tải sẵn có thể giúp cho lúc build tốn ít thời gian hơn nhưng khi container run, docker container lấy lệnh get model từ trang chủ hugging face sẽ bị chặn và không tải model hoàn thiện về docker.

6.Một số lưu ý khác

6.1. Chọn batch size phù hợp khi training

-**Batch size** hiểu đơn giản là số lượng sample được dùng cho mỗi lần huấn luyện

-Đây là một tham số giúp ta tận dụng được **tối đa** sức mạnh của phần cứng khi training

-Batch size cao có thể giúp model được huấn luyện nhanh hơn, thường set là 2^N . Đổi lại sẽ tiêu tốn nhiều VRAM của GPU, và có thể dẫn đến **CUDAOutOfMemory.**

-Việc lựa chọn batch size hợp lí để training là rất quan trọng để vừa đạt được thời gian train tốt nhất với điều kiện không bị **OutOfMemory.**

-Như trong **TrainingArguments** có 2 tham số ta sẽ cần định nghĩa:

```
per_device_train_batch_size=2,
per_device_eval_batch_size=2,
```

+)**per_device_train_batch_size**: batch size của mỗi thiết bị **GPU/TPU/CPU/...** cho việc training

+)**per_device_eval_batch_size**: batch size của mỗi thiết bị **GPU/TPU/CPU/...** cho việc evaluation

6.2. PEFT và LoraConfig

-PEFT (Parameter-efficient fine-tuning) là nhóm các phương pháp cải thiện hiệu suất cho việc finetuning các mô hình ngôn ngữ lớn dùng cho các task cụ thể.

-Cách hoạt động tổng quát là việc giữ lại phần lớn cấu trúc của pretrained model gốc, và thực hiện việc tinh chỉnh trên số nhỏ còn lại giúp giảm computation cost và memory

-LORA(Low-rank Adaptation for Large Language Model): một phương pháp thuộc nhóm Peft giúp cải thiện hiệu suất training bằng việc chỉ insert 1 lượng nhỏ weight vào trong cấu trúc model và thực hiện tinh chỉnh trên đó