WIMCHAT APP

COMPLETE GUIDE (HOW TO USE?)

Achraf Diani – Lahcen Ezzara

ENSIAS – ENSAMC

achraf_diani@um5.ac.ma – ezzara.lahcen@ensam-casa.ma

End-of-Year Internship (PFA) at WIMTECH
June, July 2024

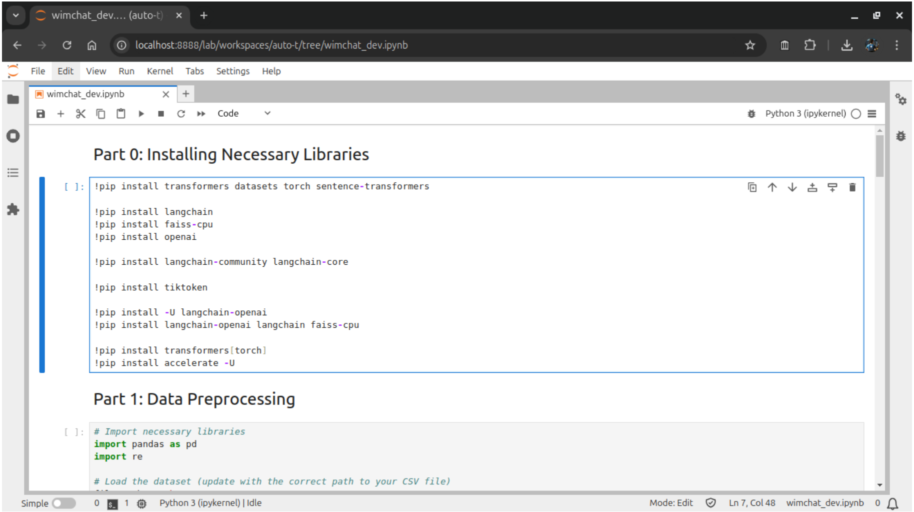
Guide Overview

1. AI Model
   - Run the Notebook
   - Data Preprocessing
   - Bert Training
   - GPT-3.5 Turbo
   - Query Example
2. Django Project
   - Project Structure
   - Model Folder
   - Run the Django Project Locally
   - Starting the Project
   - Start Chatting

AI Model

Run the Notebook

Execute cell by cell: start with libraries cell.



Part 0: Installing Necessary Libraries

```
[ ]: !pip install transformers datasets torch sentence-transformers

     !pip install langchain
     !pip install faiss-cpu
     !pip install openai

     !pip install langchain-community langchain-core

     !pip install tiktoken

     !pip install -U langchain-openai
     !pip install langchain-openai langchain faiss-cpu

     !pip install transformers[torch]
     !pip install accelerate -U
```

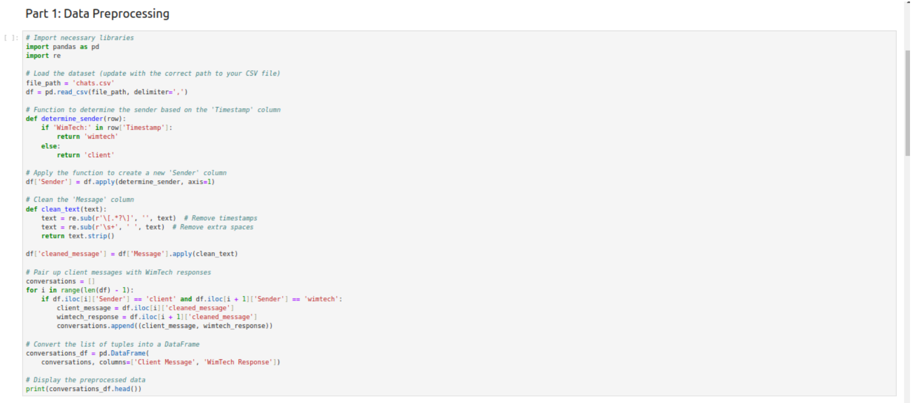
Part 1: Data Preprocessing

```
[ ]: # Import necessary libraries
     import pandas as pd
     import re

     # Load the dataset (update with the correct path to your CSV file)
```

AI Model

Data Preprocessing

The second cell is for data preprocessing change the path of the data with yours and run the cell.



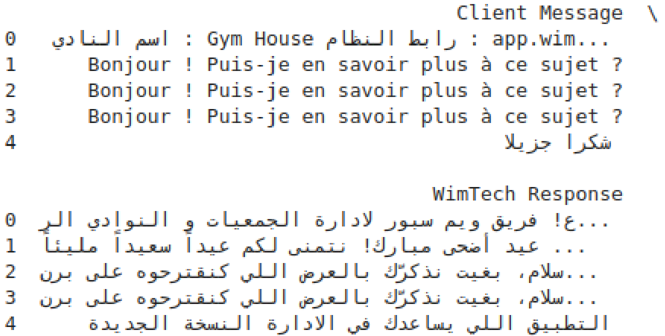
Part 1: Data Preprocessing

```python
# Import necessary libraries
import pandas as pd
import re

# Load the dataset (update with the correct path to your CSV file)
file_path = 'chats.csv'
df = pd.read_csv(file_path, delimiter=',')

# Function to determine the sender based on the 'Timestamp' column
def determine_sender(row):
    if 'WimTech:' in row['Timestamp']:
        return 'wimtech'
    else:
        return 'client'

# Apply the function to create a new 'Sender' column
df['Sender'] = df.apply(determine_sender, axis=1)

# Clean the 'Message' column
def clean_text(text):
    text = re.sub(r'\[.*?\]', '', text)  # Remove timestamps
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    return text.strip()

df['cleaned_message'] = df['Message'].apply(clean_text)

# Pair up client messages with WimTech responses
conversations = []
for i in range(len(df) - 1):
    if df.iloc[i]['Sender'] == 'client' and df.iloc[i + 1]['Sender'] == 'wimtech':
        client_message = df.iloc[i]['cleaned_message']
        wimtech_response = df.iloc[i + 1]['cleaned_message']
        conversations.append((client_message, wimtech_response))

# Convert the list of tuples into a DataFrame
conversations_df = pd.DataFrame(
    conversations, columns=['Client Message', 'WimTech Response'])

# Display the preprocessed data
print(conversations_df.head())
```

AI Model

Data Preprocessing

The second cell changes the data from the first format (wimtech - name of client) to the second format (wimtech - client).

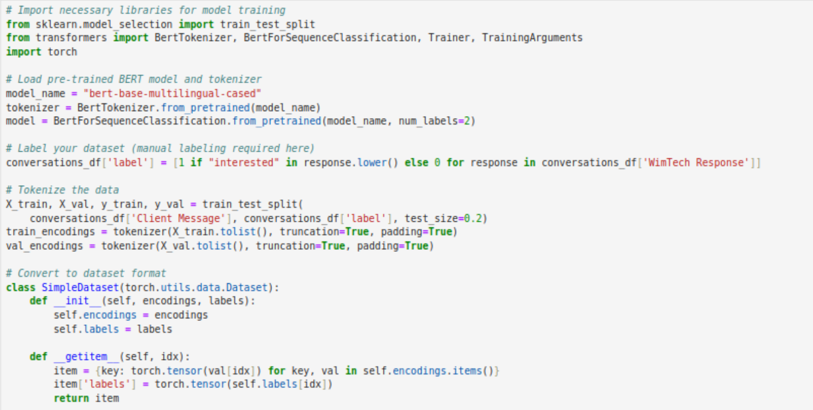


```
                                              Client Message  \
0  اسم النادي : Gym House رابط النظام : app.wim...
1     Bonjour ! Puis-je en savoir plus à ce sujet ?
2     Bonjour ! Puis-je en savoir plus à ce sujet ?
3     Bonjour ! Puis-je en savoir plus à ce sujet ?
4                                        شكرا جزيلا

                                            WimTech Response
0  ع! فريق ويم سبور لادارة الجمعيات و النوادي الر...
1  عيد أضحى مبارك! نتمنى لكم عيداً سعيداً مليئا ...
2  سلام، بغيت نذكرّك بالعرض اللي كنقترحوه على برن...
3  سلام، بغيت نذكرّك بالعرض اللي كنقترحوه على برن...
4     التطبيق اللي يساعدك في الادارة النسخة الجديدة
```

AI Model

Bert Training

The third cell is to train our model: Bert is trained from this conversations to learn how to respond (run it).

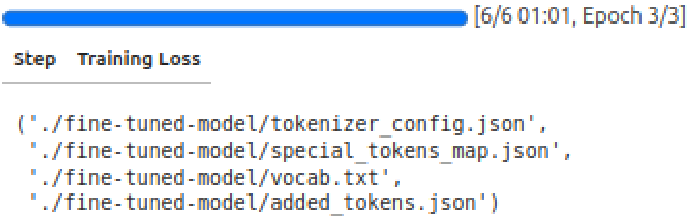


```python
# Import necessary libraries for model training
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
import torch

# Load pre-trained BERT model and tokenizer
model_name = "bert-base-multilingual-cased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Label your dataset (manual labeling required here)
conversations_df['label'] = [1 if "interested" in response.lower() else 0 for response in conversations_df['WimTech Response']]

# Tokenize the data
X_train, X_val, y_train, y_val = train_test_split(
    conversations_df['Client Message'], conversations_df['label'], test_size=0.2)
train_encodings = tokenizer(X_train.tolist(), truncation=True, padding=True)
val_encodings = tokenizer(X_val.tolist(), truncation=True, padding=True)

# Convert to dataset format
class SimpleDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
```

AI Model

Bert Training

Once runned you will get those json files you can find them on files download them (we will use them later on our app).



[6/6 01:01, Epoch 3/3]

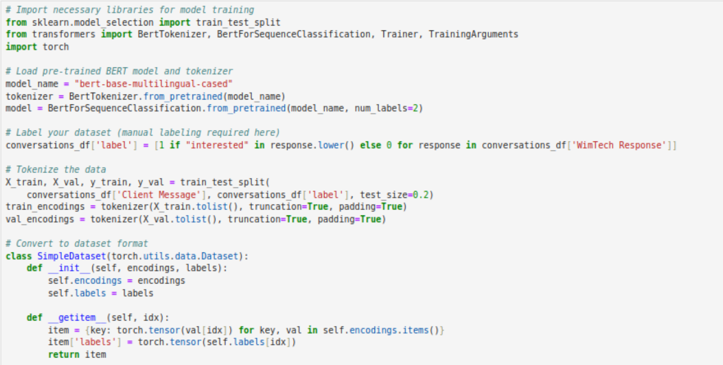
Step **Training Loss**

```
('./fine-tuned-model/tokenizer_config.json',
 './fine-tuned-model/special_tokens_map.json',
 './fine-tuned-model/vocab.txt',
 './fine-tuned-model/added_tokens.json')
```

AI Model

GPT-3.5 Turbo

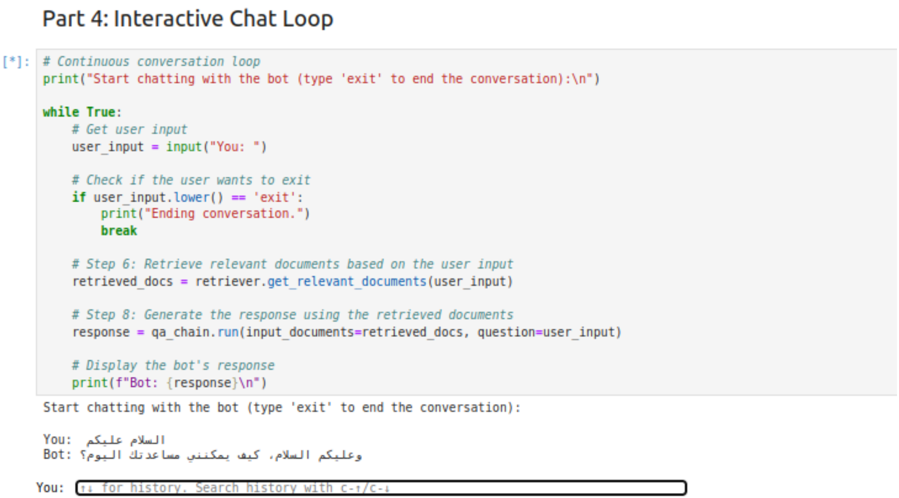
The last cell is for our Generative AI model (the model who will give the response) in this case its GPT-3.5 Turbo, we used its api to generate responses you can run this cell to see a demo of how it will respond you will find where to add your question and you will get your response.



```python
# Import necessary libraries for model training
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
import torch

# Load pre-trained BERT model and tokenizer
model_name = "bert-base-multilingual-cased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Label your dataset (manual labeling required here)
conversations_df['label'] = [1 if "interested" in response.lower() else 0 for response in conversations_df['WimTech Response']]

# Tokenize the data
X_train, X_val, y_train, y_val = train_test_split(
    conversations_df['Client Message'], conversations_df['label'], test_size=0.2)
train_encodings = tokenizer(X_train.tolist(), truncation=True, padding=True)
val_encodings = tokenizer(X_val.tolist(), truncation=True, padding=True)

# Convert to dataset format
class SimpleDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
```

AI Model

Query Example



Part 4: Interactive Chat Loop

[*]:

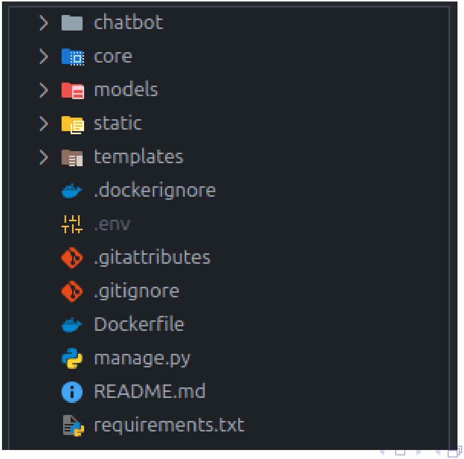
```
# Continuous conversation loop
print("Start chatting with the bot (type 'exit' to end the conversation):\n")

while True:
    # Get user input
    user_input = input("You: ")

    # Check if the user wants to exit
    if user_input.lower() == 'exit':
        print("Ending conversation.")
        break

    # Step 6: Retrieve relevant documents based on the user input
    retrieved_docs = retriever.get_relevant_documents(user_input)

    # Step 8: Generate the response using the retrieved documents
    response = qa_chain.run(input_documents=retrieved_docs, question=user_input)

    # Display the bot's response
    print(f"Bot: {response}\n")
```

```
Start chatting with the bot (type 'exit' to end the conversation):

You:  السلام عليكم
Bot: وعليكم السلام، كيف يمكنني مساعدتك اليوم؟

You: ↑↓ for history. Search history with c-↑/c-↓
```

Django Project

Project Structure

The directory structure is likely for a chatbot project, with directories for core logic, models, static assets, templates, and configuration files.

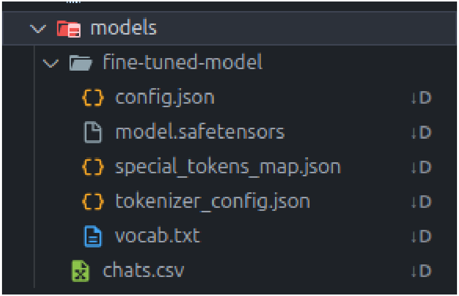


```
> chatbot
> core
> models
> static
> templates
  .dockerignore
  .env
  .gitattributes
  .gitignore
  Dockerfile
  manage.py
  README.md
  requirements.txt
```

Django Project

Model Folder

Just place all exported files (including the **fine-tuned-model** folder) and the **chats.csv** file in the ”**model**” folder. This ensures they’re easily accessible for the *Django view*.

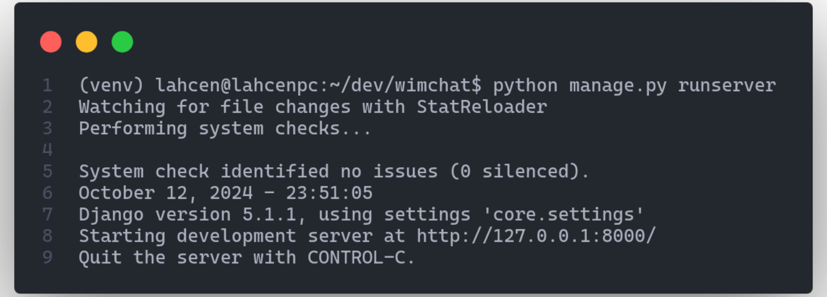


```
models
  fine-tuned-model
    config.json              ↓D
    model.safetensors        ↓D
    special_tokens_map.json  ↓D
    tokenizer_config.json    ↓D
    vocab.txt                ↓D
  chats.csv                  ↓D
```

Django Project

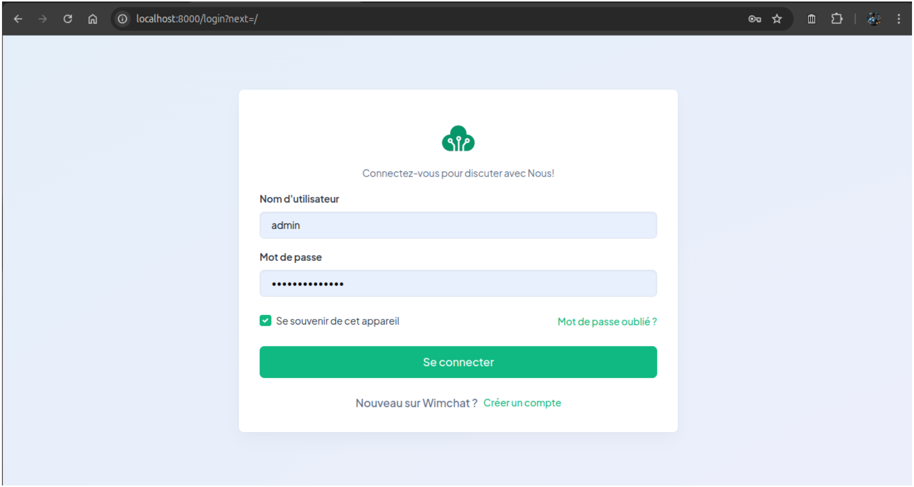
Run the Django Project Locally

To run the project locally, use the following command:
python manage.py runserver



```
(venv) lahcen@lahcenpc:~/dev/wimchat$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 12, 2024 - 23:51:05
Django version 5.1.1, using settings 'core.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Navigate to *http://localhost:8000* to find the login page.



localhost:8000/login?next=/

Connectez-vous pour discuter avec Nous!

Nom d'utilisateur

admin

Mot de passe

•••••••••••••••

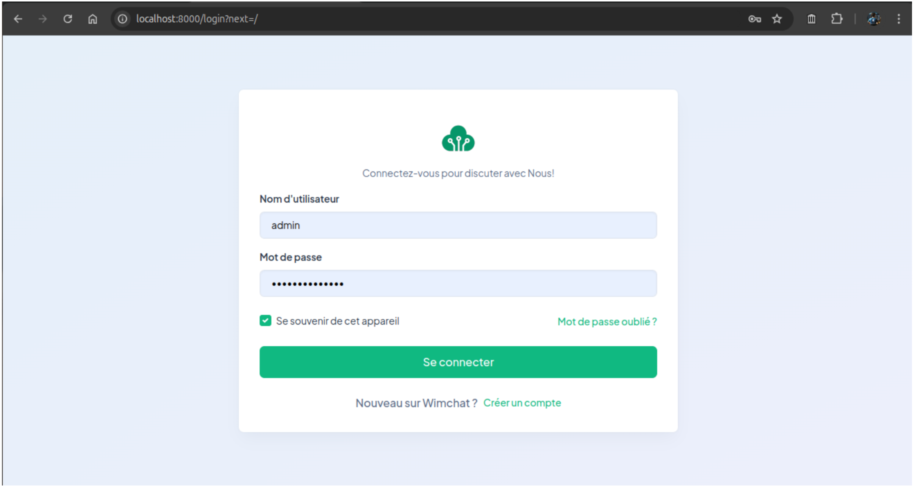
☑ Se souvenir de cet appareil

Nouveau sur Wimchat ? Créer un compte

Django Project

Start Chatting

After logging into the app, you can immediately begin chatting.



Connectez-vous pour discuter avec Nous!

Nom d'utilisateur

admin

Mot de passe

••••••••••••••

☑ Se souvenir de cet appareil

Nouveau sur Wimchat ? Créer un compte

THE END

WIMCHAT APP COMPLETE GUIDE