Documentation

Building a Multilingual Speech Recognition Model for RAG Without Training

This document details the functionalities, code structure, implementation, and usage guidelines for the for the pre-trained multilingual speech recognition model, Multilingual Whisper, to enable RAG to perform tasks in multiple languages.

Setup

Environment Setup

Python 3.9.9 and PyTorch 1.10.1 was used to train and test the models, but the codebase is expected to be compatible with Python 3.8-3.11 and recent PyTorch versions.

For the entire project, a conda environment was used, which was created using:

```
conda create -n venv python=3.9.18 anaconda
```

The PyTorch version used is PyTorch 1.10.1, which can be installed using:

```
conda install pytorch==1.10.1 torchvision==0.11.2 torchaudio==0.10.1 -c pytorch
```

Whisper dependencies

You can download and install (or update to) the latest release of Whisper with the following command:

```
pip install -U openai-whisper
```

It also requires the command-line tool ffmpeg to be installed on your system, which is available from most package managers:

```
# on Ubuntu or Debian

sudo apt update && sudo apt install ffmpeg

# on Arch Linux

sudo pacman -S ffmpeg

# on MacOS using Homebrew (https://brew.sh/)

brew install ffmpeg

# on Windows using Chocolatey (https://chocolatey.org/)

choco install ffmpeg

# on Windows using Scoop (https://scoop.sh/)

scoop install ffmpeg
```

The codebase also depends on a few Python packages, most notably OpenAI's tiktoken for their fast tokenizer implementation. The open source version of tiktoken can be installed from PyPI:

```
pip install tiktoken
```

You may need rust installed as well, in case tiktoken does not provide a pre-built wheel for your platform. If you see installation errors during the pip install command above, please follow the Getting started page to install Rust development environment. Additionally, you may need to configure the PATH environment variable, e.g. export PATH="$HOME/.cargo/bin:$PATH". If the installation fails with No module named 'setuptools_rust', you need to install setuptools_rust, e.g. by running:

```
pip install setuptools-rust
```

Available models and languages

There are five model sizes, four with English-only versions, offering speed and accuracy tradeoffs. Below are the names of the available models and their approximate memory requirements and inference speed relative to the large model; actual speed may vary depending on many factors including the available hardware.

The *.en* models for English-only applications tend to perform better, especially for the *tiny.en* and *base.en* models. We observed that the difference becomes less significant for the *small.en* and *medium.en* models.

For our purpose, we use the *medium* model, which was observed to be the most suitable model, because it is the lightest model with accurate results for transcription as well as translation.

Chatbot Dependencies

Before we start building our chatbot, we need to install some Python libraries. Here's a brief overview of what each library does:

- langchain: This is a library for GenAI. We'll use it to chain together different language models and components for our chatbot.
- openai: This is the official OpenAI Python client. We'll use it to interact with the OpenAI API and generate responses for our chatbot.
- datasets: This library provides a vast array of datasets for machine learning. We'll use it to load our knowledge base for the chatbot.
- pinecone-client: This is the official Pinecone Python client. We'll use it to interact with the Pinecone API and store our chatbot's knowledge base in a vector database.

You can install these libraries using pip like so:

```
pip install langchain==0.0.292 openai==0.28.0 datasets==2.10.1 pinecone-client==2.2.4
tiktoken==0.5.1
```

You'll need to get an OpenAI API key and Pinecone API key to run the chatbot and use a vector database respectively.

Dependencies for Transformer-based models for Text Summarization

The command *pip install transformers* is used to install the transformers package, which provides access to state-of-the-art Transformer-based models for NLP tasks, including Text Summarization.

# install transformers

!pip install transformers

Once the transformers package is installed, you can import and use the Transformer-based models in your own projects.

Program Code

```
import whisper
import os
import pinecone
import time

from langchain.chat_models import ChatOpenAI
from datasets import load_dataset
from langchain.embeddings.openai import OpenAIEmbeddings
from tqdm.auto import tqdm  # for progress bar
from langchain.vectorstores import Pinecone

from transformers import pipeline


model = whisper.load_model("medium")  # Load whisper model

# Input Query audio/video file
result_1 = model.transcribe("content/vid_1.mp4", fp16 = False)  # 
Transcription result
result_2 = model.transcribe("content/vid_1.mp4", task = 'translate', fp16 = 
False)  # Translation result

print(f"Model is {'multilingual' if model.is_multilingual else 'English-
only'} ")
print("Language : ", result_1['language'])
print("Transcription : ", result_1['text'])
print("Translation : ", result_2['text'])


os.environ["OPENAI_API_KEY"] = "YOUR_OPENAI_API_KEY" # Keep your OpenAi key 
here

chat = ChatOpenAI(
    openai_api_key = "YOUR_OPENAI_API_KEY", # Keep your OpenAi key here
    model = 'gpt-3.5-turbo'
)
embed_model = OpenAIEmbeddings(model="text-embedding-ada-002")

from langchain.schema import (
    SystemMessage,
    HumanMessage,
    AIMessage
)

messages = [
    SystemMessage(content="You are a helpful assistant."),
    HumanMessage(content="Hi AI, how are you today?"),
    AIMessage(content="I'm great thank you. How can I help you?")
]

messages_1 = [
    SystemMessage(content="You are a helpful assistant."),
    HumanMessage(content="Hi AI, how are you today?"),
    AIMessage(content="I'm great thank you. How can I help you?")
]

# now create a new user prompt
prompt = HumanMessage(
    content="What is so special about Llama 2?"  # Unknown topic for chatbot,
to be stored in vector database
)
# add to messages
messages_1.append(prompt)

res = chat(messages_1)
print(res.content)  # Output for unknown topics (Usually Hallucinations)

dataset = load_dataset(
    "jamescalam/llama-2-arxiv-papers-chunked",
    split="train"
)

pinecone.init(
    api_key='YOUR_PINECONE_API_KEY',  # Keep your Pinecone API key here
    environment="YOUR_PINECONE_ENVIRONMENT"  # Keep your Pinecone Environment
here
)

index_name = 'YOUR_PINECONE_INDEX_NAME'  # Keep your pinecone index name here

if index_name not in pinecone.list_indexes():
    pinecone.create_index(
        index_name,
        dimension=1536,
        metric='cosine'
    )
    # wait for index to finish initialization
    while not pinecone.describe_index(index_name).status['ready']:
        time.sleep(1)

index = pinecone.Index(index_name)
index.describe_index_stats()
```

```python
data = dataset.to_pandas()  # this makes it easier to iterate over the
dataset

batch_size = 100

for i in tqdm(range(0, len(data), batch_size)):
    i_end = min(len(data), i+batch_size)
    # get batch of data
    batch = data.iloc[i:i_end]
    # generate unique ids for each chunk
    ids = [f"{x['doi']}-{x['chunk-id']}" for i, x in batch.iterrows()]
    # get text to embed
    texts = [x['chunk'] for _, x in batch.iterrows()]
    # embed text
    embeds = embed_model.embed_documents(texts)
    # get metadata to store in Pinecone
    metadata = [
        {'text': x['chunk'],
         'source': x['source'],
         'title': x['title']} for i, x in batch.iterrows()
    ]
    # add to Pinecone
    index.upsert(vectors=zip(ids, embeds, metadata))

index.describe_index_stats()

text_field = "text"  # the metadata field that contains our text

# initialize the vector store object
vectorstore = Pinecone(
    index, embed_model.embed_query, text_field
)

query = "What is so special about Llama 2?"  # The Translated output is input
as Query to the RAG implemented chatbot

vectorstore.similarity_search(query, k=3)

def augment_prompt(query: str):
    # get top 3 results from knowledge base
    results = vectorstore.similarity_search(query, k=3)
    # get the text from the results
    source_knowledge = "\n".join([x.page_content for x in results])
    # feed into an augmented prompt
    augmented_prompt = f"""Using the contexts below, answer the query.

    Contexts:
    {source_knowledge}

    Query: {query}"""
    return augmented_prompt

# With RAG for previous question
prompt = HumanMessage(
    content=augment_prompt(query)  # Implementing RAG
)
```

```
messages_1.append(prompt)

res = chat(messages_1)
print(res.content)    # New answer using RAG


# Custom Query from Input Multilingual Whisper
query = result_2['text']  # The Translated output is input as Query to the
RAG implemented chatbot
vectorstore.similarity_search(query, k=3)

prompt = HumanMessage(
    content=augment_prompt(query)  # Implementing RAG
)
messages.append(prompt)

res = chat(messages)
print(res.content)    # New answer using RAG


# Summarizing RAG Chatbot output
summarizer = pipeline("summarization",
model="stevhliu/my_awesome_billsum_model")    # Load Summarization model
article = res.content  # Chatbot output as input to summarizer

summary = summarizer(article, max_length=130, min_length=30, do_sample=False)
print(summary[0]['summary_text'])    # Summary of chatbot output
```

Brief Explanation of Code:

- Loading Models and Libraries
    - Imports necessary libraries and models: whisper, os, pinecone, time, various modules from langchain, datasets, and transformers.
- Model Initialization and Usage
    - Loads the whisper model and performs transcriptions and translations of an audio/video file.
    - Whisper model output taken as a query for RAG implemented chatbot.
    - Initializes a chat model (ChatOpenAI) for interaction and creates messages for the chatbot.
    - Utilizes the chat model to get responses and prints the content.
- Dataset and Vector Storage
    - Loads a dataset related to "llama-2-arxiv-papers-chunked".
    - Initializes Pinecone for vector storage, creates an index, and updates it with embeddings and metadata.
- Vector Search and Augmented Prompt

  - Performs similarity searches in the vector store based on a given query.
  - Defines a function to create an augmented prompt using retrieved knowledge to assist in answering queries.
- RAG Implementation
  - Constructs prompts for the RAG (Retrieval-Augmented Generation) model, appending queries to messages.
  - Uses the chat model for RAG implementation and prints the new answer using RAG.
- Summarization
  - Loads a summarization model.
  - Summarizes the chatbot output from the RAG using the summarization model and prints the summary.

Usage Guide

The code can be run using python in the environment previously created, after setting few important parameters, which are:

- audio/video file path
- API keys for OpenAI and Pinecone vector db

Upon running the code, the outputs of transcription, translation, output for the default query, output for the audio/video transcription, translated and taken as a query and also the summarization for the output of the query, will be output by the code.

Summary of the Code Documentation

- The code incorporates Whisper for speech recognition and OpenAI models for chatbot interactions.
- Utilizes Pinecone for vector storage and retrieval.
- Offers functionality for transcription, translation, and chatbot interactions.
- The code showcases how pre-trained models can be leveraged for various language-related tasks without retraining.

References

- https://github.com/openai/whisper
- https://github.com/pinecone-io
- https://medium.com/@lokaregns/text-summarization-with-hugging-face-transformers-a-beginners-guide-9e6c319bb5ed