Tech-Enhanced AI Interview Learning Platform

Agam Pandey, Akshat Shaw, Aman Behera, Kushagra Singh, Vaibhav Prajapati, Vishnu Jain

April 11, 2024

Abstract

Developed a sophisticated machine learning model capable of generating diverse interview questions aligned with specific topics, ensuring depth of conversation. Integrated advanced Natural Language Processing (NLP) algorithms to analyse spoken responses, identifying grammatical errors, and offering accurate corrections after the interview. Implemented state-of-the art, speech processing techniques to assess learners speaking pace, detecting variations, and providing timely feedback for improvement. Also, curated comprehensive datasets and conducted rigorous training to enhance the AI model's adaptability and performance.

1 Introduction

The AI interview learning platform represents a pioneering approach in job interview preparation, harnessing advanced techniques in natural language processing and large language models. Built using PyTorch and the Transformers library from Hugging Face, it offers a dynamic learning environment tailored to individual needs. At its heart lies the Mistral 7B LLM, finely tuned on a rich dataset of interview questions and responses, capable of generating diverse questions aligned with specific topics and difficulty levels.

An essential feature is its ability to simulate real-life interviews through coherent conversations, facilitated by advanced conversational AI algorithms. These algorithms analyze spoken responses, identifying errors and nuances in language use, and generate follow-up questions to maintain the flow of dialogue. By integrating cutting-edge language models, NLP techniques, and speech processing capabilities, the platform delivers a personalized learning experience, empowering users to refine their interview skills and excel in a competitive job market.

2 Table of Contents

The Pipeline of the platform can be divided into following parts:

3 Requirements

```
os
torch
trl
fast wishper
huggingface_hub
datasets
accelerate
datasets scipy
bitsandbytes
git+https://github.com/huggingface/accelerate.git
git+https://github.com/huggingface/peft.git
git+https://github.com/huggingface/transformers.git
```

4 Automatic Speech Recognisation (ASR)

We used Faster Whisper transcription with CTranslate2 , as the backbone for our Automatic Speech to text conversion. This implementation is up to 4 times faster than openai/whisper for the same accuracy while using less memory.

4.1 Calculate Speaking Pace

We then wrote a function to calculate the speaking pace, which would return a value which would then passed on to pace checker, which would then give feedback to the user.

```
def calculate_speaking_pace(transcription, chunk_length):
words = transcription.split()
num_words = len(words)
speaking_rate = num_words / chunk_length  # Words per second
return speaking_rate
```

4.2 Pace Checker

We set a optimal pace range, within which answers would be acceptable, else the user would be given an prompt to ”Too Fast” or ”Too Slow”.

```
def pace_checker(pace):
optimal_pace_range = (1, 3)
if optimal_pace_range[0] <= pace <= optimal_pace_range[1]:
    print("Good pace")
elif pace < optimal_pace_range[0]:
    print("very slow")
elif pace > optimal_pace_range[1]:
    print("Too fast")
```

4.3 Speech To Text

Then we converted the corrected audio data captured from the user into text.

```
def get_text():
audio_path = "audio.wav"
result = model_audio.transcribe(audio_path)
segments, info = result
print("Detected language '%s' with probability %f" % (info.language, info.
                                            language_probability))
text = ""
for segment in segments:
    text += segment.text
pace = calculate_speaking_pace(text, chunk_length=DEFAULT_CHUNK_LENGTH)
print(pace)
pace_checker(pace)
return text
```

5 Model Training & Fine Tuning

5.1 Finetuning of Mistral 7B LLM using QLoRA

5.1.1 Preparing Data for Fine tuning

- **Data Collection:** Collected various datasets (namely SDE-Data, DS-Data, PMConsult data, Combined data)
- **Data Cleaning:** Removed noise, corrected errors, and ensuring a uniform format, then combined all these datasets and used it for fine tuning purposes of the LLM.
- **Loading the Datasets:**

```
import pandas as pd
from transformers import DataCollatorForLanguageModeling

df1 = pd.read_excel('/kaggle/input/dataset-qa/combined_dataset.xlsx')
df2=pd.read_excel('/kaggle/input/dataset-qa/Sde_data.xlsx')
df3=pd.read_excel('/kaggle/input/dataset-qa/DS_data.xlsx')
df4=pd.read_excel('/kaggle/input/dataset-qa/PMConsult_data.xlsx')
df1=df1.rename(columns={'Position/Role':'Job_Position'})

df = pd.concat([df1, df2, df3, df4], ignore_index=True)
df.drop(columns=['Column1.6','Column1.7'],inplace=True)
train_df = df.sample(frac=0.8, random_state=42)
val_df = df.drop(train_df.index)
df.to_csv('data.csv',index=False)
```

- **Data Splitting:** Splitted the dataset into two parts train and test with a split percentage of 80% for training, 10% for validation, and 10% for testing.

5.1.2 Fine-Tuning the Model — The Crux

We did fine-tuning the Mistral 7B model using the QLoRA (Quantization and LoRA) method. This approach combines quantization and LoRA adapters to improve the model's performance. We used the PEFT library from Hugging Face to facilitate the fine-tuning process.

- **Loading the dependencies:** You only have to run this once per machine

```
!pip install -q -U bitsandbytes
!pip install -q -U git+https://github.com/huggingface/transformers.git
!pip install -q -U git+https://github.com/huggingface/peft.git
!pip install -q -U git+https://github.com/huggingface/accelerate.git
!pip install -q -U datasets scipy
```

```
from accelerate import FullyShardedDataParallelPlugin, Accelerator
from torch.distributed.fsdp.fully_sharded_data_parallel import
                                            FullOptimStateDictConfig,
                                            FullStateDictConfig
fsdp_plugin = FullyShardedDataParallelPlugin(
    state_dict_config=FullStateDictConfig(offload_to_cpu=True, rank0_only=False),
    optim_state_dict_config=FullOptimStateDictConfig(offload_to_cpu=True,
                                                rank0_only=False),
)
accelerator = Accelerator(fsdp_plugin=fsdp_plugin)
```

- **Loading the Pre-trained Model:** The Mistral 7B model is loaded into the deep learning framework & we set up the accelerator using the FullyShardedDataParallelPlugin & Accelerator.

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig,
                                            AutoModelForSeq2SeqLM
# We would be fine tuning Mistral-7B LLM
base_model_id = "mistralai/Mistral-7B-Instruct-v0.2"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True, bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)model = AutoModelForCausalLM.from_pretrained(base_model_id, quantization_config=
                                             bnb_config)
```

- **Tokenization:** Tokenized the text data into the format suitable for the model to ensures compatibility with the pre-trained architecture. We use self-supervised fine-tuning to align the label & input ids.

```
tokenizer = AutoTokenizer.from_pretrained(
base_model_id,
model_max_length=512,
padding_side="left",
add_eos_token=True)
tokenizer.pad_token = tokenizer.eos_token
```

```
def tokenize(prompt):
    result = tokenizer(
        prompt,
        truncation=True,
        max_length=512,
        padding="max_length",
    )
result["labels"] = result["input_ids"].copy()
return result
tokenizer.pad_token = tokenizer.eos_token
```

- **Set up LoRA:** Now, we prepare the model for fine-tuning by applying LoRA adapters to the linear layers of the model.
- **Custom Data Collator & Question Generator:** Created custom data loaders for training, validation, & testing. These loaders facilitate efficient model training by feeding data in batches, enabling the model to learn from the dataset effectively. Provided custom prompts based on the job position, to generate questions.

```
def generate_and_tokenize_prompt(df):
full_prompt = f"""
  You are now conducting an interview for the {df['Job Position']}\ role.
  You have asked the candidate the following question:
  {df['Question']}\
  The candidate has responded as follows:
  {df['Answer']}\

  Please formulate a thoughtful follow-up question to probe deeper into the
                                              candidate's understanding and
                                              experience of the candidate's
                                              response in relation to the desired
                                              skills and knowledge for the {df["
                                              Job Position"]}\ role."""
  return tokenize(full_prompt)
```

After this we splitted the combined dataset and the uploaded it on hugging-face, and then mapped our prompt to the dataset for the fine tuning of the model, so as to increase the efficiency.

```
from datasets import load_dataset
# If the dataset is gated/private, make sure you have run huggingface-cli login
dataset = load_dataset("akshatshaw/eddcv")
```

```
tokenized_train_dataset = dataset['train'].map(generate_and_tokenize_prompt)
tokenized_val_dataset = dataset['test'].map(generate_and_tokenize_prompt)
```

- **Trainable parameters & Fine Tuning Loop:** Set the training loop for the model to fine tune, after which the model would be outputting the evaluation prompt.

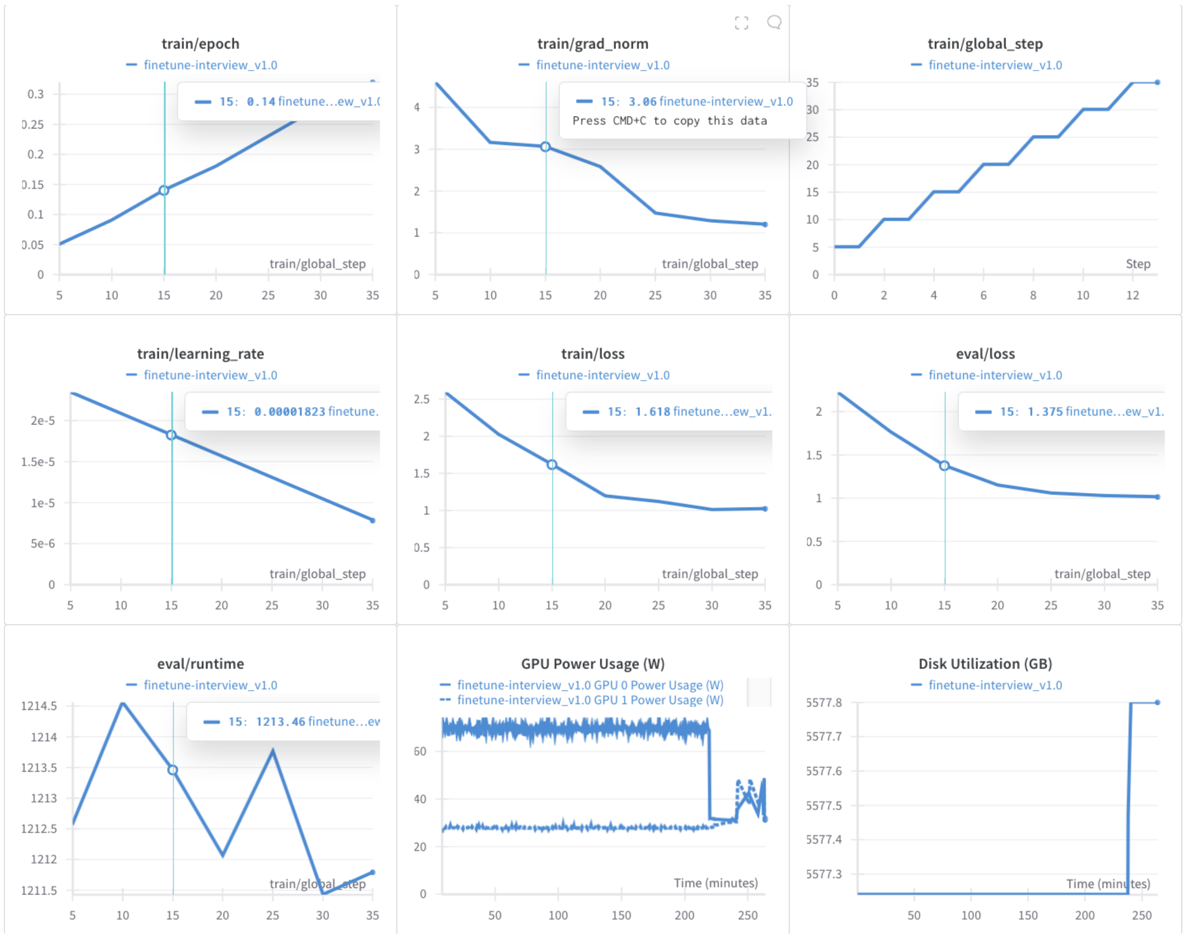

Figure 1: Model Training Graphs

```
# Now, we prepare the model for fine-tuning by applying LoRA adapters to the model
def print_trainable_parameters(model):
    """Prints the number of trainable parameters in the model."""
    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():
        all_param += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    print(
        f"trainable params: {trainable_params} || all params: {all_param} ||
                                             trainable%: {100 * (
                                             trainable_params / all_param)}
                                             ")
```

Reference about usage of peft from LoRA's paper here here After training, you can use the fine-tuned model for inference. You'll need to load the base Mistral model from the Huggingface Hub and then load the QLoRA adapters from the best-performing checkpoint directory.

5.1.3 Model Evaluation

BLeU score (Bi-Lingual Evaluation Understudy) is a metric for automatically evaluating machine-translated text. The BLEU score is a number between zero and one that measures the similarity of the machine-translated text to a set of high quality reference translations.

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \qquad p_n = \frac{\sum_{\text{n-grams}\in\text{Candidate}} \text{Count}_{\text{n-grams}}}{\sum_{\text{n-grams}\in\text{Candidate}} \text{Total}_{\text{n-grams}}}$$

where:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp\left(1 - \frac{r}{c}\right) & \text{if } c \leq r \end{cases} \qquad \begin{aligned} \text{Count}_{\text{n-grams}} &= \text{number of n-grams in candidate translation appearing} \\ \text{Total}_{\text{n-grams}} &= \text{total number of n-grams in candidate translation} \end{aligned}$$

6 Deployment

We uploaded our training datasets and **Fine Tuned Mistral-LLM** (mistralai/Mistral-7B-Instruct-v0.2) using **Quantized-LoRA** to hugging-face.

6.1 Frontend

Tech-Stack used - **HTML/ CSS**

- We created a website interface page where the user would select the interview domain according to which he would be questioned.
- This page would lead to the chatbot page, where the user can interact with our model by means of audio and textual data.

The user can select from the following domains:

- Call center agent
- Consultancy
- Customer Service
- Customer Service
- Agent Customer Service Manager
- Customer Service Representative
- Data Scientist
- Marketing Manager

6.2 Backend

Tech-Stack used - **Javascript / Flask**

6.2.1 Flask

- In the flask file, two models are imported - first one is faster-wisper, second one is Mistral.
- After receiving the audio files from the user, it is saved in the device locally in .wav format.
- Then we read this file using faster-wisper and then convert it into text.
- We then feed this text into our fine tuned mistral model, which then asks the user questions based on the domain selected by the user.
- Our model also corrects the grammatical errors present in the text prompted by the user.
- Model then outputs a desired answer based on the previous inputs given by the user and the context of the discussion.

7 Hyperparameters

These are the list of Hyperparameters involved and how they were tuned:

1. **n-bits quantisation**: we used 4-bit quantisation to speed up the process, and make our model light as it reduces the precision of the weights and biases involved in the model. For more info click here

2. **Optimal Speech Range:** we set the range to be (1-3) words/sec, as the average speaking rate of a human is 2.5 words/sec, if the input is not within this range then the user is prompted to adjust.
3. **peft LoRA configuration:** **"lora-alpha"** this sets the value of the alpha parameter used in LoRA. It controls the strength of the linearity assumption in LoRA & **"r"** this parameter controls the number of LoRA blocks in the model.
4. **Number of Epochs for training of model**: We set the number of epochs to be 50, but the eval/loss was plateauing near about 35 epochs. The training paramters graph can be seen here

8 Challenges faced

- Dataset finding and creation
- Integrating different parts of the pipelines
- Finalising the model architecture
- GPU & Computational power limitations
- Tighter time constraint

9 Limitations and Improvement

- Customized report of the candidate as per industry requirement
- Expanding the range of interview domains covered
- Refined Speech Recognition and Model training

10 References

- Github Link of the project
- Model stored in Hugging-Face
- LoRA paper used for quantisation reference
- Dataset Link
- Dataset2 Link