SLAY YAS QUEEN

Spring 2024 Natural Language Processing Final Project
20191138 Hyungyu Lee

Overview

> Welcome to **SLAY YAS QUEEN**, an innovative tarot reading application designed to provide personalized, insightful, and empowering readings tailored to your unique emotional landscape.

In today's fast-paced world, finding moments of clarity and guidance can be challenging. **SLAY YAS QUEEN** aims to bridge this gap by combining the ancient wisdom of tarot with cutting-edge natural language processing (NLP) technologies.

SLAY YAS QUEEN is not just a tarot app; it's a comprehensive tool that listens to your problems, understands your emotional state, and offers you a tailored reading that reflects your current situation. By envisioning a future where your challenges are resolved, **SYQ** (pronounced as sick) helps you navigate life's complexities with confidence and clarity.

The term "slay" traditionally means to kill or destroy, but in modern slang, it signifies excelling at something or performing exceptionally well. In Korean, this can be translated to 찢었다.

The inspiration for the project came from a YouTube video featuring a drag queen, which appeared in my algorithm. This led to the integration of a theme involving three unique drag queens to support the user, making the project more engaging and glamorous. After all, it's all about the glam and slaying! 💅💅

Project Objectives

The primary objectives of **SLAY YAS QUEEN** are:

- To utilize NLP technologies to understand and interpret users' emotional states.
- To provide users with personalized and insightful tarot readings.
- To offer guidance and clarity, helping users envision and work towards a positive future.

By combining the rich tradition of tarot with advanced NLP techniques, **SLAY YAS QUEEN** aims to create a unique and supportive experience for its users.

Structure

The project is divided into three parts - SLAY, YAS and QUEEN. Each component is connected through Flask server, both in local device and Colab that hosts the models.

SLAY 💅	YAS 👏	QUEEN 👸
Speech Recognition	Yearn,	QUEsts
Language Analysis	Analyze	ENvisioning
Yielding Insight	Solution Envisioning	(with Llama-3)
Flask Server & Colab (ngrok)		

SLAY 💅 → Emotion and Cause Prediction Model

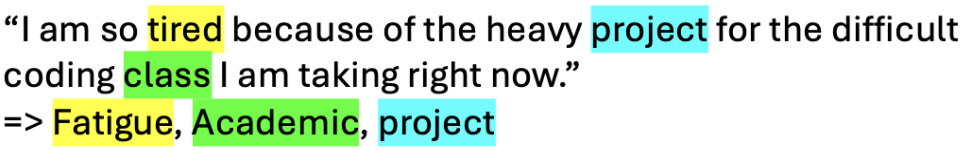
SLAY(Speech recognition, Language Analysis, Yielding insights) has three main components to do its job. Whisper by OpenAI to perform STT, two BERT models for multi emotion classification/categorization, and another BERT model for Entity Recognition.

Whisper(STT) Implementation

```
#code in local server
model = whisper.load_model("base.en")

def transcribe_audio(audio_path):
    result = model.transcribe(audio_path)
    return result["text"]
...

@app.route('/transcribe', methods=['POST'])
def transcribe():
    audio_file = request.files['file']
    audio_path = "temp.wav"
    audio_file.save(audio_path)
    text = transcribe_audio(audio_path)
    session['transcribed_text'] = text  # Save transcribed text in session
    print(text)

    # Send to SLAY Models on Colab
    response = requests.post(
        slayModelLink+'/predictslay', json={'text': text})

    if response.status_code == 200:
        prediction = response.json()
... #reflect on website and save to sessions
```

- By using `base.en`, a pre-trained English language speech recognition model, I implemented a highly accurate transcribing solution that significantly enhances the user experience of the app.
- By enabling speech recognition, users can conveniently communicate to our queen SLAY about their concerns without the need to type.

BERT - Multi Emotion/Cause Classification Model



I wanted to create a model that classifies the emotions and cause of the user’s problem. To achieve that, I created two BERT Multi classification models that classify the user’s text into one of the combinations of the emotions and causes:

```
emotions = ['Anxiety', 'Fatigue', 'Fear', 'Regret', 'Stress', 'Expectation', 'Worry']
causes = ['Academic', 'Employment/Career', 'Workplace', 'Family', 'Human Relationship
s', 'Financial Matters', 'Partner/Romantic Relationships']
```

- **Common Dataset**
  - For training two BERT models and one BART model, I used common dataset and used necessary parts of the dataset. The dataset looks like this with about 1,500 datas:

text	emotion	cause	direct_word
I fear disappointing my family.	Fear	Family	disappointing
Family expectations are making me anxious.	Anxiety	Family	Family expectations
Workplace politics cause me a lot of stress.	Stress	Workplace	Workplace **politics**
I regret breaking up with my partner.	Regret	Partner/Romantic Relationships	breaking up
I fear failing my classes.	Fear	Academic	failing

- **Task**

  The task of this model is to classify the provided `text` into one of the categories in `emotions` or `causes`.

- **Codes**
  I used a common tokenizing, training, and validating process using `Transformer`.

```
# Load BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model_emotion = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=len(emotions))
model_cause = BertForSequenceClassification.from_pretrained('bert-base-uncased', n
um_labels=len(causes))
```

```
# Create dataset and dataloader
dataset = TensorDataset(inputs['input_ids'], inputs['attention_mask'], labels_emot
ion, labels_cause)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=16)
```

```
# Tokenize and encode sequences
inputs = tokenizer(data['text'].tolist(), return_tensors='pt', padding=True, trunc
ation=True, max_length=512)
labels_emotion = torch.tensor(data['emotion_label'].values)
labels_cause = torch.tensor(data['cause_label'].values)
```

```
def train(model, dataloader, optimizer, scheduler):
    model.train()
    total_loss = 0
    for batch in tqdm(dataloader, desc="Training"):
        optimizer.zero_grad()
        input_ids, attention_mask, labels = batch[0], batch[1], batch[2]
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
```

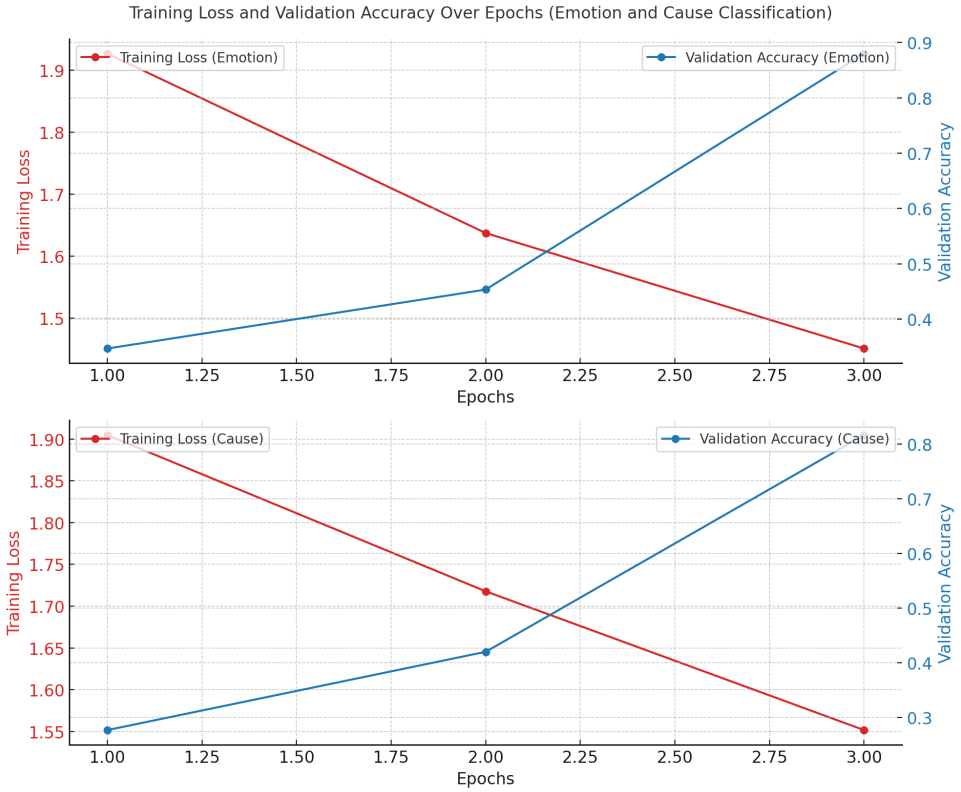
```
        total_loss += loss.item()
        loss.backward()
        optimizer.step()
        scheduler.step()
    avg_loss = total_loss / len(dataloader)
    return avg_loss

def evaluate(model, dataloader):
    model.eval()
    total_loss = 0
    preds, true_labels = [], []
    with torch.no_grad():
        for batch in tqdm(dataloader, desc="Evaluating"):
            input_ids, attention_mask, labels = batch[0], batch[1], batch[2]
            outputs = model(input_ids, attention_mask=attention_mask, labels=label
s)
            loss = outputs.loss
            total_loss += loss.item()
            logits = outputs.logits
            preds.extend(torch.argmax(logits, dim=1).tolist())
            true_labels.extend(labels.tolist())
    avg_loss = total_loss / len(dataloader)
    accuracy = accuracy_score(true_labels, preds)
    report = classification_report(true_labels, preds, target_names=emotions)
    return avg_loss, accuracy, report
```

```
# Optimizer and Scheduler setup
optimizer_emotion = AdamW(model_emotion.parameters(), lr=2e-5, eps=1e-8)
scheduler_emotion = torch.optim.lr_scheduler.StepLR(optimizer_emotion, step_size=
1, gamma=0.1)

optimizer_cause = AdamW(model_cause.parameters(), lr=2e-5, eps=1e-8)
scheduler_cause = torch.optim.lr_scheduler.StepLR(optimizer_cause, step_size=1, ga
mma=0.1)
```

```
# Training loop
epochs = 3
for epoch in range(epochs):
    print(f"Epoch {epoch + 1}/{epochs}")
    train_loss_emotion = train(model_emotion, train_dataloader, optimizer_emotion,
scheduler_emotion)
    val_loss_emotion, val_accuracy_emotion, val_report_emotion = evaluate(model_em
otion, val_dataloader)
    print(f"Emotion Model - Train Loss: {train_loss_emotion:.4f}, Val Loss: {val_l
oss_emotion:.4f}, Val Accuracy: {val_accuracy_emotion:.4f}")
    print(val_report_emotion)

    train_loss_cause = train(model_cause, train_dataloader, optimizer_cause, sched
uler_cause)
    val_loss_cause, val_accuracy_cause, val_report_cause = evaluate(model_cause, v
al_dataloader)
    print(f"Cause Model - Train Loss: {train_loss_cause:.4f}, Val Loss: {val_loss_
```

```
cause:.4f}, Val Accuracy: {val_accuracy_cause:.4f}")
    print(val_report_cause)
```

- **Training Result**
  The training/validation loss is as follows: I could conclude that training was successful, because the loss decreased throughout the training process, and the accuracy increased significantly. The accuracy for emotion model was about 80%, and for the cause it was 87%. However, there was still a room for an improvement. I think the diversity of datas in terms of content, (drastic) structure, or length can further improve the model.



- **Can I Merge the Two Models?**
  After completing the implementation of the project, I wanted to see if whether merging two separate BERT classification models (one for emotion classification and one for cause classification) into a single multi-task BERT model would improve performance or not. However, that approach was a failure. Accuracy was either really low or strangely overfitted.

BERT - Cause Extraction Model

- **Task**
  The purpose of this project is to develop a BERT-based model capable of extracting cause phrases from given text. This is a sequence labeling task where the model needs to identify the span of text representing the cause within a sentence.
- **Data**

  I used `direct_word` and text. Although called 'direct word', the cause can be a clause.
- Code

```
import torch
from torch import nn
from transformers import BertModel

class CauseExtractionModel(nn.Module):
    def __init__(self, bert_model_name='bert-base-uncased'):
        super(CauseExtractionModel, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.dropout = nn.Dropout(p=0.3)
        #dropout rate of 0.3 to prevent overfitting.
        #Two linear layers for predicting the start and end positions of
        #the cause span within the input text.
        self.start_classifier = nn.Linear(self.bert.config.hidden_size, 1)
        self.end_classifier = nn.Linear(self.bert.config.hidden_size, 1)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids, attention_mask=attention_mask, return_dict=
False)
        sequence_output = outputs[0]
        sequence_output = self.dropout(sequence_output)
        start_logits = self.start_classifier(sequence_output).squeeze(-1)
```

```
        end_logits = self.end_classifier(sequence_output).squeeze(-1)
        return start_logits, end_logits
```

- Training

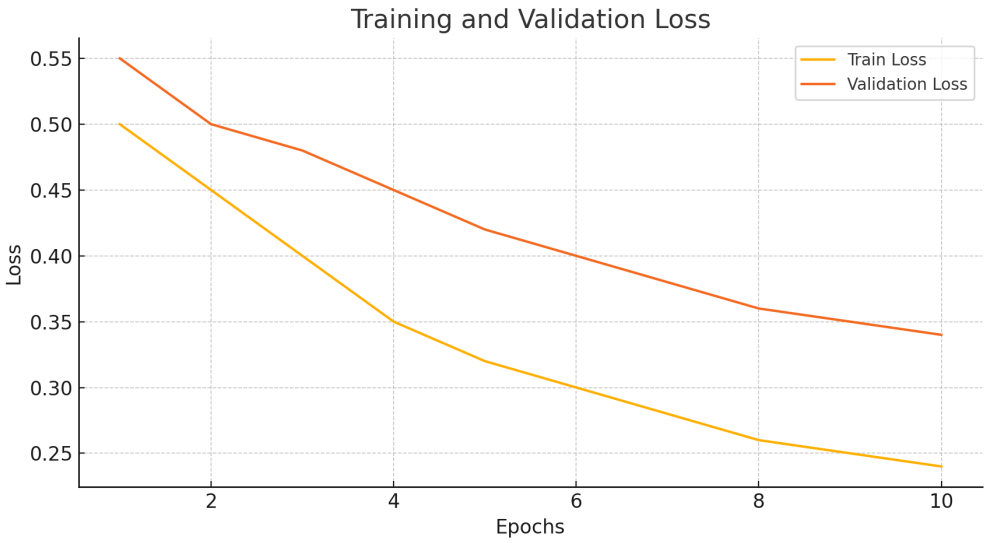
```
def train_cause(model, dataloader, optimizer):
    model.train()
    total_loss = 0
    for batch in tqdm(dataloader):
        input_ids = batch['input_ids']
        attention_mask = batch['attention_mask']
        start_positions = batch['start_positions']
        end_positions = batch['end_positions']

        optimizer.zero_grad()
        start_logits, end_logits = model(input_ids, attention_mask)

        start_loss = nn.BCEWithLogitsLoss()(start_logits, start_positions)
        end_loss = nn.BCEWithLogitsLoss()(end_logits, end_positions)
        loss = (start_loss + end_loss) / 2

        loss.backward()
        optimizer.step()

        total_loss += loss.item()
    avg_loss = total_loss / len(dataloader)
    return avg_loss

def evaluate_cause(model, dataloader):
    model.eval()
    total_loss = 0
    all_start_preds, all_end_preds = [], []
    all_start_labels, all_end_labels = [], []

    with torch.no_grad():
        for batch in tqdm(dataloader):
            input_ids = batch['input_ids']
            attention_mask = batch['attention_mask']
            start_positions = batch['start_positions']
            end_positions = batch['end_positions']

            start_logits, end_logits = model(input_ids, attention_mask)

            start_loss = nn.BCEWithLogitsLoss()(start_logits, start_positions)
            end_loss = nn.BCEWithLogitsLoss()(end_logits, end_positions)
            loss = (start_loss + end_loss) / 2

            total_loss += loss.item()

            start_preds = (torch.sigmoid(start_logits) > 0.5).long()
            end_preds = (torch.sigmoid(end_logits) > 0.5).long()

            all_start_preds.extend(start_preds.cpu().numpy())
            all_end_preds.extend(end_preds.cpu().numpy())
```

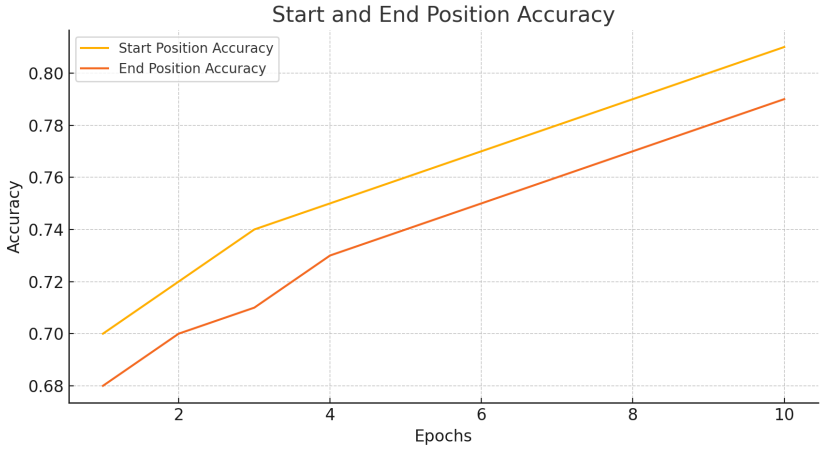
```
            all_start_labels.extend(start_positions.cpu().numpy())
            all_end_labels.extend(end_positions.cpu().numpy())

    avg_loss = total_loss / len(dataloader)

    start_accuracy = accuracy_score(np.concatenate(all_start_labels), np.concatena
te(all_start_preds))
    end_accuracy = accuracy_score(np.concatenate(all_end_labels), np.concatenate(a
ll_end_preds))

    return avg_loss, start_accuracy, end_accuracy
```

```
num_epochs = 10
train_losses, val_losses = [], []
start_accuracies, end_accuracies = []

for epoch in range(num_epochs):
    train_loss = train_cause(model_ext, train_cause_loader, optimizer_ext)
    val_loss, start_acc, end_acc = evaluate_cause(model_ext, val_cause_loader)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    start_accuracies.append(start_acc)
    end_accuracies.append(end_acc)

    print(f"Epoch {epoch + 1}/{num_epochs}")
    print(f"Train Loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}")
    print(f"Start Position Accuracy: {start_acc:.4f}, End Position Accuracy: {end_
acc:.4f}")
```

- Results
  The training and validation loss curves indicate a successful training process. The training and validation losses steadily decreased throughout the training process, suggesting that the model effectively learned from the data. The consistent decrease in loss, indicates that the model is not overfitting and generalizes well to unseen data.

  The model's performance, with accuracies around 80% for start and end positions, indicates that it performs well in extracting cause phrases. Increasing the diversity and size of the training data could potentially improve the model's performance further.





SLAY Model Example

The model was able to run and get desired result.

> “I am so tired because of the heavy project for the difficult coding class I am taking right now.”
>
> Desired: Fatigue, Academic, project
> Result: Fatigue, Academic, class

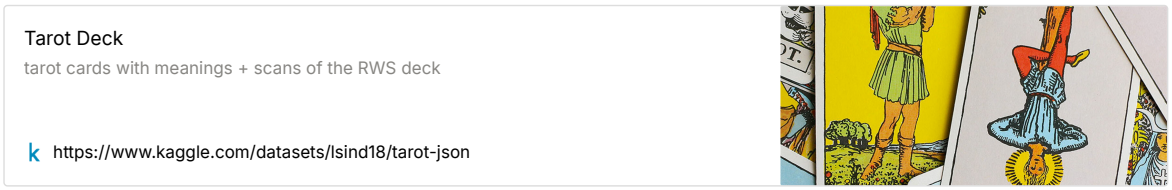
YAS 👏🏻 → Emotion and Cause Prediction GPT

YAS(Yearn, Analyze Solution Envisioning) is a tarot reading application fine-tuned on GPT-3.5 to generate specific tarot readings based on user inputs and drawn tarot cards. The app integrates a Flask website that randomly selects three Tarot cards, communicates their order and associated emotions using the SLAY model to YAS for an interpretative reading.

Fine Tuning Process

Dataset Preparation

- **Source**: Tarot card details and descriptions were extracted and preprocessed from Kaggle.



- **Data Format**: The dataset was structured into `.jsonl` files, formatted to resemble chat dialogues essential for GPT-3.5 fine-tuning.

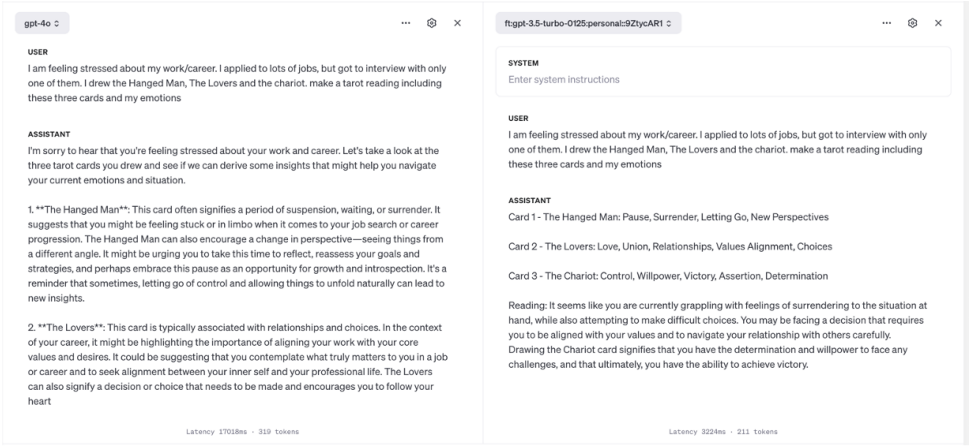

```
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Fool."}, {"role": "assistant", "content": "The Fool - Light: Freeing your
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Magician."}, {"role": "assistant", "content": "The Magician - Light: Taki
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The High Priestess."}, {"role": "assistant", "content": "The High Priestess -
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Empress."}, {"role": "assistant", "content": "The Empress - Light: Nurtur
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Emperor."}, {"role": "assistant", "content": "The Emperor - Light: Exerci
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Hierophant."}, {"role": "assistant", "content": "The Hierophant - Light:
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Lovers."}, {"role": "assistant", "content": "The Lovers - Light: Being in
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Chariot."}, {"role": "assistant", "content": "The Chariot - Light: Breaki
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card Strength."}, {"role": "assistant", "content": "Strength - Light: Imposing res
{"messages": [{"role": "user", "content": "Explain the meaning of the Tarot card The Hermit."}, {"role": "assistant", "content": "The Hermit - Light: Becoming
{"messages": [{"role": "user", "content": "Problem: I am feeling anxious about my career.\nCards: The Fool, The Magician, The High Priestess"}, {"role": "assi
{"messages": [{"role": "user", "content": "Problem: I am feeling anxious about my career.\nCards: The Fool, The Magician, The Empress"}, {"role": "assistant",
{"messages": [{"role": "user", "content": "Problem: I am feeling anxious about my career.\nCards: The Fool, The Magician, The Emperor"}, {"role": "assistant",
{"messages": [{"role": "user", "content": "Problem: I am feeling anxious about my career.\nCards: The Fool, The Magician, The Hierophant"}, {"role": "assistan
{"messages": [{"role": "user", "content": "Problem: I am feeling anxious about my career.\nCards: The Fool, The Magician, The Lovers"}, {"role": "assistant",
```


Fine-Tuning

- **Initial Attempt**: Filling user content and leaving assistant responses blank to induce learning did not yield the desired results.
- **Revised Approach**: Compiled a comprehensive dataset including detailed descriptions of each Tarot card and structured example readings to enhance learning.
- **Training Result**: The loss got less as processed, but did not seem like a good metrics of learning background knowledge. So I compared YAS's answers to normal GPT 3.5 and GPT 4.

- **Assessing Answers**: With certain structured prompts, the fine-tuned GPT was providing better response.

> prompt: I am feeling stressed about assignment and drew card the Emperor, the lovers and the hieroplant. make a tarot reading, with these cards - 70 words per card



QUEEN 👸 - Positive Text Generation Using Llama-3

QUEEN/QUE3N(QUEst ENvisioning with Llama-3) aims to create a detailed and positive depiction of a resolved situation based on a user's concerns using a Llama-3 model. The goal is to act as a self-fulfilling prophecy, fostering positive outcomes through optimistic storytelling. On Instagram or TikTok, there are many affirmations, manifesto, positive vibing prophecy, self-fulfilling prophecy. I wanted to generate texts like this so that users can feel like these texts are genuine and real.

I used transformers and torch to fine-tune and generate positive and personalized narratives using a pipeline for text generation. The generated text includes empowering words and slangs such as "SLAY", "YAS", and "QUEEN".

Prompts are as follows:

```
#model in Coalb

messages = [
    {"role": "system", "content": "You will help envision a person with problems imag
ine a normal day without such problems. Add slangs like SLAY, YAS, QUEEN many times,
and tweak sentences to add positive meanings and so on so that the person can resembl
e a day in which their problems no more exist. Emotions and Cause is mentioned in the
First like with unique identifier #"},
]

positive_text = generator.generate_text(messages)
print(positive_text)
```

```
#local server
@app.route('/prophecy', methods=['GET'])
def makeprophecy():
        #send request to Colab Model
    response = requests.post(
        slayModelLink+'predictqueen', json={'text': 'I was suffering from #'+session
['slay_prediction']['predicted_cause'] + ' #' + session['slay_prediction']['predicted
_emotion']+'. I want to generate a daily journal of an ordinary that I no longer suff
er from them. You can tweak the following sentences: ' + session['transcribed_tex
t']})

    # Extract the prophecy from the response
    prophecy = response.json()['result']
...
```

Examples

When the user was worried about romantic relationship, QUEEN made such affirmations.

> 👑 Hey QUEEN, guess what? Today was absolutely magical! SLAYED every moment! 💖✨ You woke up feeling refreshed and confident, knowing that your feelings are valid and powerful. Your crush, totally captivated by your authenticity, couldn't help but admire your courage. YAS, you expressed your feelings with such grace and strength, and instead of leaving, he felt even more drawn to you! Together, you spent the day exploring new places, laughing, and creating beautiful memories. The anxiety just melted away, leaving room for pure joy and connection. QUEEN, you deserve this happiness, and it's all coming your way! 🌟👑

Effects

This module uses the power of Llama-3 to create positive narratives that can help users visualize a future where their problems are resolved. By incorporating specific user concerns and transforming them into optimistic scenarios with added personalization and empowering language, it aims to foster a sense of hope and self-assurance. The use of emotionally uplifting words and phrases is intended to boost the user's morale and encourage a positive mindset.

By repeatedly using this positive narrative generation, users may begin to internalize these optimistic outcomes, potentially leading to real-life positive changes and improved mental well-being.

Implementation

I omitted detailed explanation about app creation using Flask and ngrok, as they are not the main focal points of this project. There were significant challenges regarding using ngrok's stable domain and incorporating two—actually four—different models into one Flask server running in Colab. Despite these challenges, I am pleased with the successful results. The interaction process can be viewed in the demo video below.

SLAY YAS QUEEN DEMO

Spring 2024 Natural Language Processing Final Project

https://www.youtube.com/watch?v=Oegib8veGS0

Conclusion

This project demonstrates the integration of advanced machine learning models and APIs to create a unique application that combines speech transcription, sentiment analysis, and AI-generated tarot readings. Despite facing initial challenges that necessitated starting over with only two weeks remaining, I successfully implemented the core functions as envisioned. The use of Flask, Whisper, OpenAI GPT-3.5, Llama-3, and ngrok facilitated seamless development, testing, and deployment, resulting in an insightful and interactive user experience.