Step-by-Step Approach: Fine-Tuning and Using T5 Model for Meeting Transcript Summarization

Introduction

This document outlines a step-by-step approach to fine-tune and use a T5 (Text-to-Text Transfer Transformer) model for the summarization of meeting transcripts. The objective is to generate structured summaries that include questions, notes, action items for both participants and selected participants, and a summary of the meeting.

Prerequisites

Before starting, ensure you have the following:

- Python environment with necessary libraries (e.g., Transformers, Pandas, PyTorch)
- Access to a dataset of meeting transcripts in an Excel file format
- Adequate GPU resources for fine-tuning (optional, but recommended)

Here's a detailed step-by-step approach for the provided code:

Step 1: Initializing the Tokenizer and Model:

```
# Import required libraries
from transformers import T5Tokenizer, T5ForConditionalGeneration

# Initialize the tokenizer and model
model_name = "t5-small"  # You can use "t5-small" or other variants
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)
```

Step 2: Loading the Dataset

```
# Import pandas to work with Excel files
import pandas as pd

# Load the meeting transcripts dataset from an Excel file
df = pd.read_excel('/content/Copy of Sample_Insights_from_MeetMinutes(1242).xlsx')
```

Step 3: Formatting the Data

```
# Import required libraries
import json
from transformers import T5Tokenizer

# Initialize the T5 tokenizer
model_name = "t5-small"  # You can use "t5-small" or other variants
tokenizer = T5Tokenizer.from_pretrained(model_name)

# Define a function to format data
def format_data(text, summary, tokenizer, max_input_length=512, max_target_length=150):
    inputs = tokenizer.encode("summarize: " + text, max_length=max_input_length, truncation=True,
padding='max_length')
    targets = tokenizer.encode(summary, max_length=max_target_length, truncation=True,
padding='max_length')

    return {
        "input_ids": inputs,
        "attention_mask": [1] * len(inputs),
```

```
        "decoder_input_ids": targets[:-1],
        "labels": targets[1:]
    }

# Format the entire dataset
formatted_data = []

for index, row in df.iterrows():
    input_text = row['og_transcript']
    output_json = json.loads(row['output'])
    target_summary = output_json['summary']['S']  # Assuming 'summary' is the key in your target JSON
    formatted_instance = format_data(input_text, target_summary, tokenizer)
    formatted_data.append(formatted_instance)
```

Step 4: Preparing the Data for Fine-Tuning

```
# Import required libraries
import torch
from torch.utils.data import DataLoader, TensorDataset, random_split
from transformers import T5ForConditionalGeneration, T5Tokenizer, AdamW

# Initialize the T5 model
model_name = "t5-small"  # You can use "t5-small" or other variants
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Convert the formatted data into PyTorch tensors
input_ids = torch.tensor([example["input_ids"] for example in formatted_data])
```

```
attention_mask = torch.tensor([example["attention_mask"] for example in formatted_data])
decoder_input_ids = torch.tensor([example["decoder_input_ids"] for example in formatted_data])
labels = torch.tensor([example["labels"] for example in formatted_data])

# Create a TensorDataset
dataset = TensorDataset(input_ids, attention_mask, decoder_input_ids, labels)

# Split the dataset into training and validation sets
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# Define batch size and create data loaders
batch_size = 4  # Adjust this based on your available GPU memory
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)

# Define optimizer and training parameters
optimizer = AdamW(model.parameters(), lr=1e-5)
epochs = 3  # Adjust the number of training epochs as needed
```

Step 5: Fine-Tuning the Model

```
# Fine-tuning loop
for epoch in range(epochs):
    model.train()
    for batch in train_loader:
```

```
        input_ids, attention_mask, decoder_input_ids, labels = batch
        optimizer.zero_grad()
        outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            decoder_input_ids=decoder_input_ids,
            labels=labels
        )
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for batch in val_loader:
            input_ids, attention_mask, decoder_input_ids, labels = batch
            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask,
                decoder_input_ids=decoder_input_ids,
                labels=labels
            )
            total_val_loss += outputs.loss.item()

    average_val_loss = total_val_loss / len(val_loader)
    print(f"Epoch {epoch + 1}/{epochs}, Validation Loss: {average_val_loss:.4f}")
```

```
# Save the fine-tuned model and tokenizer
model.save_pretrained("fine-tuned-t5")
tokenizer.save_pretrained("fine-tuned-t5")
```

Step 7: Generate Summaries, Questions, and Action Items

```
from transformers import T5ForConditionalGeneration, T5Tokenizer

# Load the fine-tuned model and tokenizer
model = T5ForConditionalGeneration.from_pretrained("fine-tuned-t5")
tokenizer = T5Tokenizer.from_pretrained("fine-tuned-t5")

# Define prompt templates for questions, notes, and action items
question_prompt = "Generate a question from the following transcript: "
note_prompt = "Generate a note from the following transcript: "
other_action_item_prompt = "Generate an action item for others from the following transcript: "
user_action_item_prompt = "Generate an action item for the selected participants from the following transcript: "

# Function to generate content based on prompts
def generate_content(prompt, transcript_chunk):
    input_text = transcript_chunk
    input_prompt = input_text + " " + prompt

    # Tokenize and generate content based on the prompt
    input_ids = tokenizer.encode(input_prompt, return_tensors="pt", max_length=512, truncation=True, padding=True)
    generated_ids = model.generate(input_ids, max_length=150, min_length=30, num_beams=4, length_penalty=2.0, num_return_sequences=1)
    generated_content = tokenizer.decode(generated_ids[0], skip_special_tokens=True)
```

```
    return generated_content

# Initialize lists to store extracted information
questions = []
notes = []
other_action_items = []
user_action_items = []

# Process each transcript chunk
for chunk in speaker_turns:
    # Extract questions
    question = generate_content(question_prompt, chunk)
    questions.append(question)

    # Extract notes
    note = generate_content(note_prompt, chunk)
    notes.append(note)

    # Extract other action items
    other_action_item = generate_content(other_action_item_prompt, chunk)
    other_action_items.append(other_action_item)

    # Extract user action items
    user_action_item = generate_content
```