Meeting Bank Summarization

1: Setup and Dependencies

First, we clone the utility repository and install all necessary libraries, including the transformers library for BART and the SummerTime library required by the ResultsEval.py script.

```
# ============================================================================
## 1. Setup, Dependencies, and Directory Change
# ============================================================================
# Setup Dependencies
!pip install transformers
!pip install datasets
!pip install rouge_score
!pip install evaluate
!pip install accelerate
!pip install bert_score
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.36.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2025.11.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.0->transformers) (2025.3.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.0->transformers) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.0->transformers) (1.2.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2025.11.12)
Requirement already satisfied: datasets in /usr/local/lib/python3.12/dist-packages (4.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets) (3.20.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.32.4)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.12/dist-packages (from datasets) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets) (3.6.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2025.3.0)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.36.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from datasets) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets) (6.0.3)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (3.13.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (1.2.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (2025.11.12)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.8.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (6.7.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (0.4.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.22.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.17.0)
Requirement already satisfied: rouge_score in /usr/local/lib/python3.12/dist-packages (0.1.2)
Requirement already satisfied: absl-py in /usr/local/lib/python3.12/dist-packages (from rouge_score) (1.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (from rouge_score) (3.9.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from rouge_score) (2.0.2)
Requirement already satisfied: six>=1.14.0 in /usr/local/lib/python3.12/dist-packages (from rouge_score) (1.17.0)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk->rouge_score) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk->rouge_score) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk->rouge_score) (2025.11.3)
```

```
import torch
import numpy as np
import evaluate
import pandas as pd
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSeq2SeqLM,
    Seq2SeqTrainingArguments,
    Seq2SeqTrainer,
    DataCollatorForSeq2Seq
)
import os

# --- Configuration ---
MODEL_CHECKPOINT = "facebook/bart-base"
MAX_INPUT_LENGTH = 1024
MAX_TARGET_LENGTH = 128
OUTPUT_DIR = "./models/meetingbank"
EVAL_BATCH_SIZE = 8
NUM_TRAIN_EPOCHS = 1 # Reduced for quick run. Increase for better results.
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Change directory to a clean output location
GDRIVE_ROOT =  "/content/drive/MyDrive/CS_685/youtube-video-summarization/"

%cd $GDRIVE_ROOT
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/CS_685/youtube-video-summarization
```

2: Data Loading and Preprocessing for BART

We will use the Hugging Face datasets library for easy access to the MeetingBank data and the transformers library for preprocessing (tokenization) which is the core preprocessing step for models like BART.

```
# ==============================================================================
## 2. Configuration and Preprocessing
# ==============================================================================
print("\n--- Section 2: Data Loading and Preprocessing ---")

# 1. Load Data and Tokenizer
print("Loading dataset and tokenizer...")
raw_datasets = load_dataset("huuuyeah/meetingbank")
tokenizer = AutoTokenizer.from_pretrained(MODEL_CHECKPOINT)

# 2. Define Preprocessing Function (Tokenization)
def preprocess_function(examples):
    # Tokenize input (transcript)
    model_inputs = tokenizer(
        examples["transcript"],
        max_length=MAX_INPUT_LENGTH,
        truncation=True,
        padding="max_length"
    )

    # Tokenize label (summary)
    # The tokenizer's `as_target_tokenizer` context manager is used for label tokenization
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["summary"],
            max_length=MAX_TARGET_LENGTH,
            truncation=True,
            padding="max_length"
        )

    # Replace padding token id with -100 for loss ignoring
    labels["input_ids"] = [
        [(l if l != tokenizer.pad_token_id else -100) for l in label]
        for label in labels["input_ids"]
    ]

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

# 3. Apply Preprocessing and Data Type Fix
print("Applying tokenization and data type fixes...")
tokenized_datasets = raw_datasets.map(
    preprocess_function,
    batched=True,
    remove_columns=raw_datasets["train"].column_names
)

# === Ensure all token ID arrays are standard integer types (int32) ===
# This prevents the OverflowError during prediction/decoding.
def ensure_int32(example):
    example["input_ids"] = np.array(example["input_ids"], dtype=np.int32)
    example["labels"] = np.array(example["labels"], dtype=np.int32)
    example["attention_mask"] = np.array(example["attention_mask"], dtype=np.int32)
    return example

tokenized_datasets = tokenized_datasets.map(ensure_int32)
tokenized_datasets.set_format("torch")
print("Preprocessing complete. Data types secured.")
```

```
--- Section 2: Data Loading and Preprocessing ---
Loading dataset and tokenizer...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Applying tokenization and data type fixes...
Map: 100%  861/861 [00:02<00:00, 376.35 examples/s]
/usr/local/lib/python3.12/dist-packages/transformers/tokenization_utils_base.py:4169: UserWarning: `as_target_tokenizer` is deprecated and will be removed in v5 of Transformers. You can tokenize your labels by using the argument `text_targ
  warnings.warn(
Map: 100%  861/861 [00:01<00:00, 810.98 examples/s]
Preprocessing complete. Data types secured.
```

3: Training Setup and Fine-Tuning (Required to get a Fine-Tuned Model)

```
# ==============================================================================
## 3. Training Setup and Fine-Tuning
# ==============================================================================
print("\n--- Section 3: BART Fine-Tuning Setup ---")

# 1. Load the BART Model and Data Collator
model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_CHECKPOINT)
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

# 2. Define Training Arguments
training_args = Seq2SeqTrainingArguments(
    output_dir=OUTPUT_DIR,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=EVAL_BATCH_SIZE,
    num_train_epochs=NUM_TRAIN_EPOCHS,
    learning_rate=2e-5,
    eval_strategy="epoch",
    save_strategy="epoch",
    fp16=torch.cuda.is_available(),
    predict_with_generate=True,
    load_best_model_at_end=True,
    report_to="none"
)

# 3. Initialize Trainer (FutureWarning note: using processing_class for future compatibility)
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    processing_class=tokenizer,
)

# 4. Start Fine-Tuning
print(f"Starting BART fine-tuning ({NUM_TRAIN_EPOCHS} epoch)...")
trainer.train()

# Find the path to the best saved model checkpoint
best_checkpoint_path = trainer.state.best_model_checkpoint if trainer.state.best_model_checkpoint else OUTPUT_DIR

print(f"\nBest/Final checkpoint path: {best_checkpoint_path}")
```

```
--- Section 3: BART Fine-Tuning Setup ---
Starting BART fine-tuning (1 epoch)...
/usr/local/lib/python3.12/dist-packages/transformers/data/data_collator.py:740: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please consider converting the list to a single numpy.ndarray with numpy.array
  batch["labels"] = torch.tensor(batch["labels"], dtype=torch.int64)
 [1293/1293 03:00, Epoch 1/1]
```

Epoch	Training Loss	Validation Loss
1	1.860000	1.476765

```
/usr/local/lib/python3.12/dist-packages/transformers/modeling_utils.py:3918: UserWarning: Moving the following attributes in the config to the generation config: {'early_stopping': True, 'num_beams': 4, 'no_repeat_ngram_size': 3, 'forced_
  warnings.warn(
There were missing keys in the checkpoint model loaded: ['model.encoder.embed_tokens.weight', 'model.decoder.embed_tokens.weight', 'lm_head.weight'].

Best/Final checkpoint path: ./models/meetingbank/checkpoint-1293
```

4: Evaluation Function

```
# Load the ROUGE metric
rouge_metric = evaluate.load("rouge")

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    if isinstance(preds, tuple):
        preds = preds[0]

    # Ensure predictions are within valid token ID range and are integers
    max_vocab_id = tokenizer.vocab_size - 1
    preds = np.clip(preds, 0, max_vocab_id).astype(np.int32)

    # Decode predictions and labels
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id).astype(np.int32)
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    # Compute ROUGE
    result = rouge_metric.compute(
        predictions=decoded_preds, references=decoded_labels, use_stemmer=True
    )

    # Format score
    result = {f"rouge_{k}": round(v * 100, 4) for k, v in result.items()}

    return result
```

5: Running Both Evaluations

```
# ==============================================================================
## 5. Running Both Evaluations
# ==============================================================================
print("\n--- Section 5: Running Evaluations ---")

# Define common evaluation arguments
eval_args = Seq2SeqTrainingArguments(
    output_dir="./evaluation_output",
    per_device_eval_batch_size=EVAL_BATCH_SIZE,
    predict_with_generate=True,
    disable_tqdm=False,
    report_to="none",
)
```

```
# --- A. Evaluation on the Pre-trained Model (Baseline) ---
print("\n" + "="*50)
print("Evaluating Pre-trained BART Model (Baseline)")
print("="*50)
pre_trained_model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_CHECKPOINT)

pre_trained_trainer = Seq2SeqTrainer(
    model=pre_trained_model,
    args=eval_args,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    processing_class=tokenizer,
)

pre_trained_results = pre_trained_trainer.predict(
    test_dataset=tokenized_datasets["test"],
    # FIX: Pass generation kwargs directly (num_beams, max_length)
    num_beams=4,
    max_length=MAX_TARGET_LENGTH
)
print("Pre-trained Model ROUGE Results:")
pretrained_results = compute_metrics((pre_trained_results.predictions, pre_trained_results.label_ids))
# print(pretrained_results)

# Turn pretrained_results into dataframe
pretrained_results_df = pd.DataFrame(pretrained_results, index=[0])
print(pretrained_results_df)
```

```
--- Section 5: Running Evaluations ---

==================================================
Evaluating Pre-trained BART Model (Baseline)
==================================================
Pre-trained Model ROUGE Results:
   rouge_rouge1  rouge_rouge2  rouge_rougeL  rouge_rougeLsum
0       35.8944       24.4663       31.0136          30.9908
```

```
# --- B. Evaluation on the Fine-Tuned Model ---
print("\n" + "="*50)
print("Evaluating Fine-Tuned BART Model")
print("="*50)
# Load the best model found during training
fine_tuned_model = AutoModelForSeq2SeqLM.from_pretrained(best_checkpoint_path)

fine_tuned_trainer = Seq2SeqTrainer(
    model=fine_tuned_model,
    args=eval_args,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    processing_class=tokenizer,
)

fine_tuned_results = fine_tuned_trainer.predict(
    test_dataset=tokenized_datasets["test"],
    num_beams=4,
    max_length=MAX_TARGET_LENGTH
)
print("Fine-Tuned Model ROUGE Results:")
finetuned_results = compute_metrics((fine_tuned_results.predictions, fine_tuned_results.label_ids))
# print(finetuned_results)

# Turn fine-tuned results into dataframe
finetuned_results_df = pd.DataFrame(finetuned_results, index=[0])
print(finetuned_results_df)
```

```
==================================================
Evaluating Fine-Tuned BART Model
==================================================
Fine-Tuned Model ROUGE Results:
   rouge_rouge1  rouge_rouge2  rouge_rougeL  rouge_rougeLsum
0        64.031       54.1433        61.462          61.4351
```

6: Save the Evaluation Results

```
# ==============================================================================
## 6. Save Both Evaluation Results to CSV
# ==============================================================================
print("\n--- Section 6: Saving Predictions to CSV ---")

def save_predictions_to_csv(prediction_output, model_name, raw_data, tokenizer):
    """
    Decodes predictions, aligns them with inputs/references, and saves to CSV.
    """
    print(f"Saving predictions for {model_name}...")

    # 1. Decode Predictions (Generated Summaries)
    pred_ids = prediction_output.predictions
    if isinstance(pred_ids, tuple):
        pred_ids = pred_ids[0]

    # Ensure predictions are within valid token ID range and are integers
    # Add clipping before casting to int32 to prevent OverflowError
    max_vocab_id = tokenizer.vocab_size - 1
    pred_ids = np.clip(pred_ids, 0, max_vocab_id).astype(np.int32)
    decoded_preds = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)

    # 2. Decode Labels (Reference Summaries)
    label_ids = prediction_output.label_ids
    # Replace -100 padding with pad_token_id before decoding
    label_ids = np.where(label_ids != -100, label_ids, tokenizer.pad_token_id)

    # Cast to int32 before decoding
    label_ids = label_ids.astype(np.int32)
    decoded_labels = tokenizer.batch_decode(label_ids, skip_special_tokens=True)

    # 3. Get Original Inputs
    original_inputs = raw_datasets["test"]["transcript"]

    # 4. Create DataFrame and Save
    results_df = pd.DataFrame({
        'Input_Transcript': original_inputs,
        'Reference_Summary': decoded_labels,
        'Predicted_Summary': decoded_preds
    })

    RESULTS_FILE_PATH = f"{model_name.lower().replace(' ', '_')}_predictions.csv"
    DIR_PATH = os.path.dirname('./data/meetingbank/results/')
    SAVE_PATH = os.path.join(DIR_PATH, RESULTS_FILE_PATH)
    results_df.to_csv(SAVE_PATH, index=False)

    print(f"Successfully saved to: {SAVE_PATH}")
    return SAVE_PATH

def save_evaluations_to_csv(results_df, model_name, type, base_dir='./data/meetingbank/eval/'):
    """
    Saves a ROUGE score dataframe to a CSV file.
    """
    # Create directory if it doesn't exist
    os.makedirs(base_dir, exist_ok=True)

    # Sanitize model_name for filename
    sanitized_model_name = model_name.lower().replace(' ', '_').replace('-', '_')
    eval_path = os.path.join(base_dir, f"{sanitized_model_name}_rouge_scores_{type}.csv")
    results_df.to_csv(eval_path, index=False)
    print(f"Successfully saved {model_name} ROUGE scores to: {eval_path}")

# --- Execute Saving ---

# 1. Save Pre-trained Model Results
save_predictions_to_csv(
    pre_trained_results,
    "Pre-trained BART",
    raw_datasets,
    tokenizer
)

# 2. Save Fine-Tuned Model Results
save_predictions_to_csv(
    fine_tuned_results,
    "Fine-Tuned BART",
    raw_datasets,
    tokenizer
)

# 3. Save Pre-trained Evaluation Results (ROUGE scores)
save_evaluations_to_csv(pretrained_results_df, "Pre-trained BART", "segment")

# 4. Save Fine-tuned Evaluation Results (ROUGE scores)
save_evaluations_to_csv(finetuned_results_df, "Fine-Tuned BART", "segment")

print("\nAll prediction and evaluation files have been generated in the current directory.")
```

```
--- Section 6: Saving Predictions to CSV ---
Saving predictions for Pre-trained BART...
Successfully saved to: ./data/meetingbank/results/pre-trained_bart_predictions.csv
Saving predictions for Fine-Tuned BART...
Successfully saved to: ./data/meetingbank/results/fine-tuned_bart_predictions.csv
Successfully saved Pre-trained BART ROUGE scores to: ./data/meetingbank/eval/pre_trained_bart_rouge_scores_segment.csv
Successfully saved Fine-Tuned BART ROUGE scores to: ./data/meetingbank/eval/fine_tuned_bart_rouge_scores_segment.csv

All prediction and evaluation files have been generated in the current directory.
```

7: Combine Segment Summaries (The "Combine" Step)

This step decodes the model's predictions, sorts them by uid, and groups them by source to create the full predicted summary for each meeting.

```
import pandas as pd
import numpy as np

# Assuming the following objects are defined and available in your Notebook:
# fine_tuned_results: The prediction output from trainer.predict()
# raw_datasets: The loaded segmented dataset (must contain 'uid', 'summary', 'transcript')
# tokenizer: The BART Tokenizer

def combine_segments_to_full_summaries(prediction_output, raw_datasets, tokenizer):
    """
    Decodes predictions, extracts the Meeting ID from 'uid', sorts by 'uid',
    and concatenates segment summaries into full meeting summaries.
    """
    # 1. Decode Predictions
    pred_ids = prediction_output.predictions
    if isinstance(pred_ids, tuple):
        pred_ids = pred_ids[0]

    # Explicitly cast to np.int32 to prevent OverflowError
    # Add clipping to ensure values are within valid token ID range
    max_vocab_id = tokenizer.vocab_size - 1
    pred_ids = np.clip(pred_ids, 0, max_vocab_id).astype(np.int32)
    decoded_preds = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)

    # 2. Prepare Metadata DataFrame
    test_metadata = pd.DataFrame(raw_datasets["test"])
    test_metadata['generated_summary_segment'] = decoded_preds

    if 'uid' not in test_metadata.columns:
        raise ValueError("DataFrame must contain the 'uid' column.")

    # Explicitly ensure the 'uid' column is of string type to prevent potential TypeError
    test_metadata['uid'] = test_metadata['uid'].astype(str)

    # 3. CRITICAL: Extract Meeting ID (location_date) from 'uid' (location_date_item)
    # rsplit('_', 1) splits once from the right (separating the segment index)
    # Changed to .apply() to avoid a mysterious TypeError with .str.rsplit()
    test_metadata['meeting_id'] = test_metadata['uid'].apply(lambda x: x.rsplit('_', 1)[0])

    # 4. Sort by 'uid' to ensure correct segment order (e.g., _001, _002...)
    test_metadata = test_metadata.sort_values(by=['uid'])

    # 5. Group by 'meeting_id' and concatenate the predicted summaries
    grouped_summaries = test_metadata.groupby('meeting_id')['generated_summary_segment'].apply(
        lambda x: ' '.join(x.tolist())
    ).reset_index(name='concatenated_full_summary')

    return grouped_summaries
```

```
# Execution Example:
full_generated_summaries_df = combine_segments_to_full_summaries(
    fine_tuned_results,
    raw_datasets,
    tokenizer
)

print("\nFull Generated Meeting Summaries (Combined):")
print(full_generated_summaries_df.head())
```

```
Full Generated Meeting Summaries (Combined):
           meeting_id                         concatenated_full_summary
0  AlamedaCC_01052021  A bill for an ordinance changing the zoning cl...
1  AlamedaCC_01062015  Recommendation to authorize City Manager or de...
2  AlamedaCC_01072020  Recommendation to receive CDBG funding for the...
3  AlamedaCC_01162018  A MOTION approving the parcel map on 1700 Park...
4  AlamedaCC_01192016  Recommendation to adopt resolutions for the Al...
```

8: Evaluate Full Meeting Summaries (The "Evaluate" Step)

This step consists of two parts: building the Golden Reference Summary by concatenating the segment summaries, and then performing the final ROUGE comparison.

8A: Function to Construct Golden Reference Summary

This function implements your logic: using source as the key and concatenating the summary fields, sorted by uid.

```
def load_full_reference_summaries(meeting_ids, raw_datasets):
    """
    Constructs the full Golden Reference Summary by combining the segment 'summary'
    fields, using the 'uid' for ordering and grouping by extracted Meeting ID.
    """

    # 1. Access the test set metadata
    test_metadata = pd.DataFrame(raw_datasets["test"])

    # Explicitly ensure the 'uid' column is of string type
    test_metadata['uid'] = test_metadata['uid'].astype(str)

    # 2. Extract Meeting ID from 'uid' using .apply() to avoid potential TypeError
    test_metadata['meeting_id'] = test_metadata['uid'].apply(lambda x: x.rsplit('_', 1)[0])

    # 3. Filter data for the required meeting IDs
    test_metadata = test_metadata[test_metadata['meeting_id'].isin(meeting_ids)].copy()

    # 4. Sort by 'uid' to ensure correct segment order
    test_metadata = test_metadata.sort_values(by=['uid'])

    # 5. Group by 'meeting_id' and concatenate the 'summary' fields
    grouped_references = test_metadata.groupby('meeting_id')['summary'].apply(
        lambda x: ' '.join(x.tolist())
    ).reset_index(name='concatenated_full_reference')

    # Convert to dictionary {meeting_id: full_summary_text}
    full_references_dict = grouped_references.set_index('meeting_id')['concatenated_full_reference'].to_dict()

    print(f"\nSuccessfully constructed full reference summaries for {len(full_references_dict)} meetings.")

    return full_references_dict
```

8B: Execute Meeting-Level ROUGE Evaluation

```
# Assuming full_generated_summaries_df (from Step 4) and rouge_metric are available

def evaluate_full_meeting_summaries(generated_df, rouge_metric, raw_datasets):
    """
    Compares the combined predicted summaries against the constructed Golden Summaries
    and computes the final Meeting-Level ROUGE and BERTScore.
    """
    # 1. Get list of Meeting IDs to evaluate
    meeting_ids = generated_df['meeting_id'].tolist()
    # 2. Load/Construct Golden Reference Summaries
    full_references_dict = load_full_reference_summaries(meeting_ids, raw_datasets)

    # 3. Map the reference summaries onto the generated DataFrame
    generated_df['full_reference_summary'] = generated_df['meeting_id'].map(full_references_dict)

    # Drop rows where the full reference summary was not found
    generated_df.dropna(subset=['full_reference_summary'], inplace=True)

    # 4. Extract lists for ROUGE calculation
    predictions = generated_df['concatenated_full_summary'].tolist()
    references = generated_df['full_reference_summary'].tolist()

    # 5. Compute ROUGE
    result = rouge_metric.compute(
        predictions=predictions,
        references=references,
        use_stemmer=True
    )

    # 6. Compute BERTScore
    print("Computing BERTScore...")
    bertscore = evaluate.load("bertscore")
    bert_results = bertscore.compute(predictions=predictions, references=references, lang="en")

    result["bertscore_precision"] = np.mean(bert_results["precision"])
    result["bertscore_recall"] = np.mean(bert_results["recall"])
    result["bertscore_f1"] = np.mean(bert_results["f1"])

    # 7. Format and Print Results
    # Convert scores to percentage and round to 4 decimal places
    # Note: BERTScore is already 0-1, so multiplying by 100 makes it percentage
    formatted_result = {k: round(v * 100, 4) for k, v in result.items()}

    print("\n" + "="*50)
    print("FINAL MEETING-LEVEL SCORES (Divide-and-Conquer)")
    print("="*50)
    print(formatted_result)

    return formatted_result

# Execution Example:
final_meeting_scores = evaluate_full_meeting_summaries(
    full_generated_summaries_df.copy(),
    rouge_metric,
    raw_datasets
)

final_meeting_scores_df = pd.DataFrame(final_meeting_scores, index=[0])
print(final_meeting_scores_df)
print("\n")

# Save final_meeting_scores to csv
save_evaluations_to_csv(final_meeting_scores_df, 'Fine-Tuned BART', 'meeting')
```

```
Successfully constructed full reference summaries for 559 meetings.
Computing BERTScore...
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

==================================================
FINAL MEETING-LEVEL SCORES (Divide-and-Conquer)
==================================================
{'rouge1': np.float64(62.8969), 'rouge2': np.float64(52.269), 'rougeL': np.float64(59.7957), 'rougeLsum': np.float64(59.7355), 'bertscore_precision': np.float64(93.3672), 'bertscore_recall': np.float64(89.6658), 'bertscore_f1': np.float64
    rouge1  rouge2   rougeL  rougeLsum  bertscore_precision  bertscore_recall  \
0  62.8969  52.269  59.7957    59.7355              93.3672           89.6658

   bertscore_f1
0       91.4372


Successfully saved Fine-Tuned BART ROUGE scores to: ./data/meetingbank/eval/fine_tuned_bart_rouge_scores_meeting.csv
```

Meeting Bank Temporal Alignment

1: Data Preparation and Pair Generation

The goal is to transform the `raw_datasets` into a binary classification format: (Summary Sentence, Transcript Segment) → Label (1 for Match, 0 for No Match).

1.1: Setup and Dependencies

```
# Install necessary libraries if not already installed
!pip install nltk datasets transformers pandas evaluate
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: datasets in /usr/local/lib/python3.12/dist-packages (4.0.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: evaluate in /usr/local/lib/python3.12/dist-packages (0.4.6)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets) (3.20.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.12/dist-packages (from datasets) (2.32.4)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets) (3.6.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2025.3.0)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.36.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from datasets) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets) (6.0.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (3.13.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.24.0->datasets) (1.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.32.2->datasets) (2025.11.12)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.8.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (6.7.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (0.4.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (1.22.0)
```

```
# --- TEMPORAL ALIGNMENT DEPENDENCIES & CONSTANTS ---
import nltk
import random
from itertools import product
from datasets import Dataset
import evaluate
from transformers import AutoModelForSequenceClassification, AutoTokenizer, TrainingArguments, Trainer, pipeline
import numpy as np
import torch

# Ensure NLTK punkt is available for sentence tokenization
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')

# Ensure 'punkt_tab' resource is available (as identified in previous turns)
try:
    nltk.data.find('tokenizers/punkt_tab')
except LookupError:
    nltk.download('punkt_tab')

# Define Constants for the Alignment Model
# IMPORTANT: Choose a path to save your trained BERT model
```

```
MODEL_SAVE_DIR = "./models/temporal_alignment/best_bert_model"
MODEL_NAME = "bert-base-uncased"
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

1.2: Function to Create Sentence Pairs (Positive and Negative Samples)

```
from datasets import load_dataset

# Load Meeting Bank data set with train, validation, and test splits
meeting_bank = load_dataset("huuuyeah/meetingbank")
```

```
def create_alignment_matching_dataset(raw_datasets, split_name="train", num_negatives=3):
    """
    Creates a sentence-pair dataset for Temporal Alignment from the MeetingBank segmented data.

    Positive Sample: (Summary Sentence, True Transcript Segment) -> Label 1
    Negative Sample: (Summary Sentence, Random Non-matching Transcript Segment from the same meeting) -> Label 0
    """

    data_df = pd.DataFrame(raw_datasets[split_name])
    matching_records = []

    # Create Meeting ID column for grouping and negative sampling
    data_df['meeting_id'] = data_df['uid'].apply(lambda x: x.rsplit('_', 1)[0])
    grouped_data = data_df.groupby('meeting_id')

    for meeting_id, meeting_segments in grouped_data:
        # Get all transcript segments for the current meeting
        all_transcript_segments = meeting_segments['transcript'].tolist()

        for index, row in meeting_segments.iterrows():
            summary_segment = row['summary']
            true_transcript = row['transcript']

            # 1. Split the reference summary segment into individual sentences
            sentences = nltk.sent_tokenize(summary_segment)

            for sentence in sentences:
                # --- Positive Sample (Label 1) ---
                matching_records.append({
                    'summary_sentence': sentence,
                    'transcript_segment': true_transcript,
                    'label': 1
                })

                # --- Negative Samples (Label 0) ---
                non_source_segments = [
                    t for t in all_transcript_segments if t != true_transcript
                ]

                # Sample negative segments
                if non_source_segments:
                    negative_samples = random.sample(
                        non_source_segments,
                        min(num_negatives, len(non_source_segments))
                    )

                    for neg_transcript in negative_samples:
                        matching_records.append({
                            'summary_sentence': sentence,
                            'transcript_segment': neg_transcript,
                            'label': 0
                        })

    alignment_df = pd.DataFrame(matching_records)
    print(f"Created {len(alignment_df)} sentence-pair records for Matching from the {split_name} split.")

    # Shuffle the dataset and reset index
    return alignment_df.sample(frac=1).reset_index(drop=True)

# Example Execution:
```

```
alignment_train_df = create_alignment_matching_dataset(meeting_bank, split_name="train", num_negatives=3)
alignment_test_df = create_alignment_matching_dataset(meeting_bank, split_name="test", num_negatives=3)

print(alignment_train_df.head())
```

```
Created 39128 sentence-pair records for Matching from the train split.
Created 3159 sentence-pair records for Matching from the test split.
                                    summary_sentence  \
0  AN ORDINANCE related to fees and charges for p...
1  Approves an official map amendment to rezone p...
2                                       (District 1)
3  A bill for an ordinance designating 4345 West ...
4  Solution to consider include preferential park...

                                  transcript_segment  label
0  Agenda Item 18 Council Bill 118 834 Relating t...      0
1  12 eyes, one abstention. Council Resolution 15...      0
2  A report from Develop and Services recommendat...      0
3  11 days. Uh, final consideration of Council 91...      0
4  Excuse me, sir, you know, you've already been ...      1
```

2: Model Setup and Training

This phase uses the generated DataFrame to fine-tune a BERT model for binary classification.

2.1: Tokenization

```
from datasets import Dataset, Value # Add Value here

MODEL_NAME = "bert-base-uncased"
alignment_tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

# Convert Pandas DataFrames to Hugging Face Dataset objects
train_dataset = Dataset.from_pandas(alignment_train_df.drop(columns=['__index_level_0__'], errors='ignore'))
test_dataset = Dataset.from_pandas(alignment_test_df.drop(columns=['__index_level_0__'], errors='ignore'))

# Ensure the 'label' column is integer type
train_dataset = train_dataset.cast_column("label", Value("int32")) # Use Value object
test_dataset = test_dataset.cast_column("label", Value("int32")) # Use Value object


def tokenize_function(examples):
    """Tokenizes the sentence-pair input for BERT."""
    # The two sequences are passed as arguments; BERT uses [SEP] to join them
    return alignment_tokenizer(
        examples['summary_sentence'],
        examples['transcript_segment'],
        padding="max_length",
        truncation=True,
        max_length=512 # Max input length for BERT
    )

# Apply tokenization to the datasets
tokenized_train_dataset = train_dataset.map(tokenize_function, batched=True)
tokenized_test_dataset = test_dataset.map(tokenize_function, batched=True)
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 5.64kB/s]
config.json: 100% 570/570 [00:00<00:00, 70.4kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.96MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 1.04MB/s]
Casting the dataset: 100% 39128/39128 [00:01<00:00, 30556.98 examples/s]
Casting the dataset: 100% 3159/3159 [00:00<00:00, 44828.89 examples/s]
Map: 100% 39128/39128 [01:39<00:00, 393.63 examples/s]
Map: 100% 3159/3159 [00:06<00:00, 489.43 examples/s]
```

2.2: Fine-Tuning BERT

```python
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer
import evaluate
import numpy as np

# 1. Define the metric function
# Temporal Alignment is a Binary Classification task, so we use accuracy/F1.
def compute_metrics(p):
    """Computes the evaluation metric (e.g., Accuracy)"""
    metric = evaluate.load("accuracy") # Changed from load_metric("accuracy")

    # Get the predicted labels (index of the highest logit)
    predictions = np.argmax(p.predictions, axis=1)

    return metric.compute(predictions=predictions, references=p.label_ids)


# 2. Define Training Arguments for Checkpoint Saving
# Assuming your output_dir is defined
output_directory = "./models/meetingbank/temporal/"

# Instantiate the model here, using MODEL_NAME
alignment_model_instance = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=2) # Using MODEL_NAME instead of undefined alignment_model

training_args = TrainingArguments(
    output_dir=output_directory,
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./alignment_logs',

    # --- CRITICAL CONFIGURATION FOR SAVING BEST MODEL ---
    eval_strategy="epoch",  # Evaluate and save at the end of every epoch
    save_strategy="epoch",        # Save checkpoint at the end of every epoch
    load_best_model_at_end=True,  # Load the model with the best metric after training finishes
    metric_for_best_model="accuracy", # Specify the metric to monitor for "best"
    save_total_limit=2,
    report_to=[] # Explicitly disable all reporting services, including wandb
)

# 3. Initialize Trainer with compute_metrics
trainer = Trainer(
    model=alignment_model_instance, # Use the instantiated model
    args=training_args,
    train_dataset=tokenized_train_dataset,
    eval_dataset=tokenized_test_dataset,
    tokenizer=alignment_tokenizer,
    compute_metrics=compute_metrics # Pass the metric function here
)

# 4. Start Training and Load Best Checkpoint
trainer.train()

# After training, the model in 'trainer.model' is the best checkpoint!
# 5. Save the final BEST model to a specific subdirectory
trainer.save_model(f"{output_directory}/best_temporal_alignment_model")
print(f"Best model saved to: {output_directory}/best_temporal_alignment_model")
```

```
model.safetensors: 100%                                 440M/440M [00:01<00:00, 297MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-2893676842.py:43: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
                                        [7338/7338 29:59, Epoch 3/3]
Epoch  Training Loss  Validation Loss  Accuracy
    1       0.315400         0.405384  0.802469
    2       0.247500         0.381501  0.842355
    3       0.190200         0.395005  0.845204

Downloading builder script:     4.20k/? [00:00<00:00, 472kB/s]
Best model saved to: ./models/meetingbank/temporal//best_temporal_alignment_model
```

3: Inference and Final Alignment

After training, you use the model to predict the best match for every summary sentence in the test set.

3.1: Alignment Prediction Loop

The prediction requires testing every summary sentence against every transcript segment within the same meeting.

```
# --- PHASE II & III: INFERENCE AND EVALUATION FUNCTIONS ---

# --- Step 3.1: Prepare Inference Data ---
def prepare_inference_data(meeting_id, raw_datasets, full_generated_summaries_df):
    """Creates all (Summary Sentence, Transcript Segment) pairs for scoring."""
    test_metadata = pd.DataFrame(raw_datasets["test"])
    if 'meeting_id' not in test_metadata.columns:
        test_metadata['meeting_id'] = test_metadata['uid'].apply(lambda x: x.rsplit('_', 1)[0])
    meeting_data = test_metadata[test_metadata['meeting_id'] == meeting_id].copy()
    candidate_segments = list(zip(meeting_data['transcript'].tolist(), meeting_data['uid'].tolist()))

    try:
        full_summary_text = full_generated_summaries_df[full_generated_summaries_df['meeting_id'] == meeting_id]['concatenated_full_summary'].iloc[0]
    except IndexError:
        print(f"Error: Generated summary not found for meeting ID {meeting_id}")
        return pd.DataFrame()

    query_sentences = nltk.sent_tokenize(full_summary_text)
    inference_records = []

    for query_sentence, (segment_text, segment_uid) in product(query_sentences, candidate_segments):
        inference_records.append({
            'query_sentence': query_sentence, 'transcript_segment': segment_text,
            'segment_uid': segment_uid, 'meeting_id': meeting_id
        })

    inference_df = pd.DataFrame(inference_records)
    return inference_df
```

3.2: Execute Inference (Score Alignment Pairs)

This step uses your trained BERT model to score every possible combination of summary sentences and transcript segments generated in Step 3.1.

```
from datasets import Dataset # Import Dataset for efficient processing

# --- Step 3.2: Execute Inference (Scoring) ---
def execute_inference(inference_df: pd.DataFrame, aligner) -> pd.DataFrame:
    """
    This function is no longer called directly from `run_alignment_pipeline`
    as the batching logic has been moved there for overall efficiency.
    Its previous logic for single-meeting inference is now integrated.
    """
    print("Warning: execute_inference was called, but its logic should be handled by run_alignment_pipeline for full dataset processing.")
    return inference_df
```

3.3: Final Alignment Decision

This step determines the best source segment for each unique summary sentence by finding the maximum score.

```
# --- Step 3.3: Final Alignment Decision ---
def make_alignment_decision(scored_inference_df: pd.DataFrame) -> pd.DataFrame:
    """Selects the segment with the highest 'match_score' for each sentence."""
    if scored_inference_df.empty: return pd.DataFrame()

    best_alignments = scored_inference_df.loc[
        scored_inference_df.groupby('query_sentence')['match_score'].idxmax()
    ]

    final_alignment_table = best_alignments[['meeting_id', 'query_sentence', 'segment_uid', 'match_score'
    ].rename(columns={'segment_uid': 'aligned_transcript_uid'})
    return final_alignment_table.reset_index(drop=True)
```

4: Execute Inference and Final Alignment Decision

This step runs the scoring and selects the best matching segment for each summary sentence across all test meetings.

```
# --- PHASE II: STEP 4 EXECUTION BLOCK ---

def run_alignment_pipeline(raw_datasets, full_generated_summaries_df, model_path, model_name):
    """
    Orchestrates the execution of the full Temporal Alignment process (Steps 3.1-3.3)
    by processing all data as a single Hugging Face Dataset.
    """

    # Load Alignment Pipeline once
    try:
        alignment_tokenizer = AutoTokenizer.from_pretrained(model_name)
        aligner = pipeline(
            "sentiment-analysis",
            model=model_path,
            tokenizer=alignment_tokenizer,
            device=0 if torch.cuda.is_available() else -1,
            max_length=512,
            truncation=True,
            padding=True
        )
    except Exception as e:
        print(f"Error: Could not load alignment model. Check model_path. Error: {e}")
        return pd.DataFrame()

    all_test_meeting_ids = full_generated_summaries_df['meeting_id'].unique()
    all_inference_records = []

    print(f"Preparing all inference data for {len(all_test_meeting_ids)} meetings...")

    # 1. Collect all inference data across all meetings
    for mid in all_test_meeting_ids:
        inference_df_for_meeting = prepare_inference_data(mid, raw_datasets, full_generated_summaries_df)
        if not inference_df_for_meeting.empty:
            # Store original index to reconstruct later if needed
            inference_df_for_meeting['original_index'] = range(len(inference_df_for_meeting))
            all_inference_records.append(inference_df_for_meeting)

    if not all_inference_records:
        print("No inference data prepared.")
        return pd.DataFrame()

    # Concatenate all meeting-specific DataFrames into one large DataFrame
    full_inference_df = pd.concat(all_inference_records, ignore_index=True)

    # Convert to Hugging Face Dataset for efficient batching
    # Explicitly remove 'segment_uid' and 'meeting_id' columns before creating pipeline inputs
    hf_inference_dataset = Dataset.from_pandas(full_inference_df).rename_columns({
        "query_sentence": "text",
        "transcript_segment": "text_pair"
    })
    # Remove columns that are not 'text' or 'text_pair' to avoid issues with the pipeline's tokenizer
    columns_to_remove = [col for col in hf_inference_dataset.column_names if col not in ['text', 'text_pair', 'original_index', 'segment_uid', 'meeting_id']]
    if 'segment_uid' in hf_inference_dataset.column_names: # Check if 'segment_uid' exists before removing
        columns_to_remove.append('segment_uid')
    if 'meeting_id' in hf_inference_dataset.column_names: # Check if 'meeting_id' exists before removing
        columns_to_remove.append('meeting_id')
    if 'original_index' in hf_inference_dataset.column_names: # Check if 'original_index' exists before removing
        columns_to_remove.append('original_index')

    # Remove duplicates before passing to remove_columns
    columns_to_remove = list(set(columns_to_remove))

    hf_inference_dataset = hf_inference_dataset.remove_columns(columns_to_remove)

    # Convert the Hugging Face Dataset into a list of dictionaries
    # that the pipeline can directly consume for batched text-pair input.
    pipeline_inputs = hf_inference_dataset.map(lambda x: {'text': x['text'], 'text_pair': x['text_pair']}).to_list()
    print(f"Starting batched Temporal Alignment inference for {len(pipeline_inputs)} pairs...")

    # 2. Execute Inference (Scoring) on the entire Dataset
    predictions_raw = aligner(pipeline_inputs, top_k=None, batch_size=32)

    match_scores = []
    for pred_list_for_example in predictions_raw:
        match_score = next((item['score'] for item in pred_list_for_example if item['label'] == 'LABEL_1'), 0.0)
        match_scores.append(match_score)

    # Add scores back to the full_inference_df
    full_inference_df['match_score'] = match_scores

    # 3. Make Alignment Decisions per meeting
    final_results_list = []
    for mid in all_test_meeting_ids:
        # Filter data for the current meeting
        meeting_specific_df = full_inference_df[full_inference_df['meeting_id'] == mid]
        if not meeting_specific_df.empty:
            alignment_result = make_alignment_decision(meeting_specific_df)
            final_results_list.append(alignment_result)

    final_alignment_df = pd.concat(final_results_list, ignore_index=True)
    return final_alignment_df

# --- EXECUTE THE ALIGNMENT PIPELINE ---
# The result of this execution is the final DataFrame needed for evaluation.
# Use the correct path for the saved model.
correct_model_path = os.path.join(output_directory, "best_temporal_alignment_model")

# Ensure full_generated_summaries_df is defined from previous steps
# If it's not defined, uncomment and run the cell that generates it
# full_generated_summaries_df = combine_segments_to_full_summaries(
#     fine_tuned_results,
#     raw_datasets,
#     tokenizer
# )

final_alignment_df = run_alignment_pipeline(
    meeting_bank,
    full_generated_summaries_df,
    correct_model_path,
    MODEL_NAME
)
print("Temporal Alignment Decisioning Complete.")
```

```
Device set to use cuda:0
Preparing all inference data for 559 meetings...
Map: 100% 2452/2452 [00:00<00:00, 11081.89 examples/s]
Starting batched Temporal Alignment inference for 2452 pairs...
Temporal Alignment Decisioning Complete.
```

5: Evaluate Temporal Alignment Accuracy

This step calculates the final Alignment Accuracy (Match Rate).

```
# --- Step 5: Evaluation Function ---
def evaluate_alignment(final_alignment_df: pd.DataFrame, raw_datasets) -> dict:
    """Evaluates the Temporal Alignment accuracy (Match Rate)."""

    test_metadata = pd.DataFrame(raw_datasets["test"])
    if 'meeting_id' not in test_metadata.columns:
        test_metadata['meeting_id'] = test_metadata['uid'].apply(lambda x: x.rsplit('_', 1)[0])

    truth_records = []
    for index, row in test_metadata.iterrows():
        summary_segment = row['summary']
        true_uid = row['uid']
        sentences = nltk.sent_tokenize(summary_segment)

        for sentence in sentences:
            truth_records.append({'query_sentence': sentence, 'true_transcript_uid': true_uid})

    ground_truth_df = pd.DataFrame(truth_records)
    comparison_df = pd.merge(final_alignment_df, ground_truth_df, on='query_sentence', how='inner')

    if comparison_df.empty:
        print("Warning: Prediction/Ground Truth sentences could not be matched.")
        return {'alignment_accuracy': 0.0}

    comparison_df['is_correct'] = (comparison_df['aligned_transcript_uid'] == comparison_df['true_transcript_uid'])
    accuracy = comparison_df['is_correct'].mean() * 100

    print("\n============================================")
    print("FINAL TEMPORAL ALIGNMENT ACCURACY (Match Rate)")
    print("============================================")
    print(f"Total Sentences Evaluated: {len(comparison_df)}")
    print(f"Alignment Accuracy: {accuracy:.2f}%")

    return {'alignment_accuracy': accuracy}
```

```
# --- PHASE III: STEP 5 EXECUTION BLOCK (EVALUATION) ---

# Calculate the Alignment Accuracy
alignment_results = evaluate_alignment(final_alignment_df, raw_datasets)

# Display the result
# print(alignment_results)

# Trun alignment_resutls into dataframe
alignment_results_df = pd.DataFrame([alignment_results])
print(alignment_results_df)

# Save the alignmnent_results_df
alignment_results_df.to_csv("./data/meetingbank/eval/alignment_results.csv", index=False)
```

```
============================================
FINAL TEMPORAL ALIGNMENT ACCURACY (Match Rate)
============================================
Total Sentences Evaluated: 14303
Alignment Accuracy: 0.81%
   alignment_accuracy
0            0.811019
```