# Intelligent Meeting Transcripts Summarizer

# 🧩 Problem Statement

In today’s fast-paced work environment, professionals often participate in numerous meetings across various domains. These meetings can be lengthy, unstructured, and information-dense, making it difficult to retrieve key takeaways efficiently. Traditional note-taking is either manual and error-prone or lacks contextual relevance.

The challenge is to build an intelligent system that can automatically generate accurate, relevant, and concise summaries of meeting transcripts. These summaries must support both general overviews and query-specific extraction for better knowledge retrieval.

# 💡 Motivation

The increasing reliance on remote and hybrid work models has drastically amplified the volume of virtual meetings. As a result:

* Important decisions and action items are often buried in lengthy transcripts.
* Manual summarization is time-consuming and inconsistent.
* There is a growing demand for query-aware summaries to extract domain-specific or role-specific information.

Automating this process using state-of-the-art transformer-based models like BART and T5 offers a scalable solution. This not only reduces cognitive load on employees but also enhances productivity and collaboration across teams.

# 📘 Introduction

Meeting summarization is a crucial task in the domain of Natural Language Processing (NLP), aiming to condense lengthy conversations into meaningful and actionable summaries. It has gained significant attention with the advent of transformer-based models, which have revolutionized text generation and understanding.

This project explores the use of pre-trained models like **BART-base**, **T5-base**, **BART-large-XSum**, and **BART-large-CNN** on the **QMSUM** dataset—a benchmark corpus designed for query-based meeting summarization. The work focuses on:

* Preprocessing raw transcripts into training-ready formats
* Experimenting with different summarization model architectures
* Evaluating outputs using ROUGE, BERTScore, METEOR, and BLEU
* Building a deployable summarization API using FastAPI and Docker

The ultimate goal is to create a **robust, scalable, and accurate meeting summarization tool** for real-world business scenarios.

# **🔄** Project Phases

To streamline the workflow and ensure systematic development, the project was divided into **three distinct phases**:

**1. Data Import and Preparation**

* Preprocessed the QMSUM dataset into structured JSONL format
* Tokenized, chunked, and split transcripts into training, validation, and test sets
* Handled long sequences to fit within transformer model constraints

**2. Model Fine-Tuning**

* Fine-tuned multiple transformer models:  
  BART-base, T5-base, BART-large-XSum, BART-large-CNN
* Performed hyperparameter tuning and checkpointing
* Evaluated models using:
  + ROUGE-L
  + BLEU
  + BERTScore
  + METEOR
* Compared models based on performance metrics and generated summary quality

**3. Deployment**

* Created a FastAPI-based backend with modular architecture
* Implemented endpoints to support:
  + Single transcript input
  + Batch JSON file upload
  + Real-time summarization
* Integrated Jinja2 for a minimal frontend interface
* Logged request history, timestamps, and model usage for monitoring
* Dockerized the complete application for seamless deployment

## Phase 1: Data Import and Preparation

### 📌 Dataset Overview

The QMSUM (Query-based Multi-domain Meeting Summarization) dataset is a benchmark dataset for query-based meeting summarization.

### Dataset Structure

The dataset is divided into three folders:

| Folder | Purpose | # Files (Meetings) |
| --- | --- | --- |
| train/ | Training set | 162 JSON files |
| test/ | Testing set | 35 JSON files |
| val/ | Validation set | 35 JSON files |

Each JSON file in the above folders represents **one meeting**, containing:

* meeting\_transcripts (list of utterances)
* query (question about the meeting)
* answer (a span from the transcript that answers the query)
* summary (gold/normal summary)
* metadata like dialogue acts, speaker IDs, etc.

You merged all JSON files in each folder into a single .jsonl file:

* train.jsonl
* test.jsonl
* val.jsonl

Each line in these .jsonl files represents a complete preprocessed **meeting sample**, combining all fields into a single JSON object.

### 📁 Original Data Format

Each meeting sample contains:

* meeting\_transcripts: List of utterances in a meeting
* general\_summary: Human-written full summary for the meeting
* specific\_summary: List of sub-summaries per query (used in gold set)
* query: The corresponding query to focus on specific aspects
* dialogue\_acts, role, and other metadata

### 🔄 Preprocessing Steps

1. **Concatenated all utterances** into a single document string
2. **Removed unnecessary newlines**, extra whitespace, and speaker tags (if noisy)
3. Created two types of samples:
   * **Normal dataset**: Use the entire meeting as input (src) and full summary as output (tgt)
   * **Gold dataset**: Use only gold span text (manually selected relevant parts) as src, and the corresponding sub-summary as tgt

### 🔤 Final Format

| Field | Description |
| --- | --- |
| src | Input to the model (full meeting or gold span text) |
| tgt | Target summary (either full summary or gold sub-summary) |

#### 🔹 Dataset Sample (Before and After)

**Before:**

{  
 "meeting\_transcripts": ["utterance 1", "utterance 2", ...],  
 "general\_summary": "summary..."  
}

**After (Normal):**

{  
 "src": "utterance 1 utterance 2 ...",  
 "tgt": "summary..."  
}

**After (Gold):**

{  
 "src": "selected gold span text",  
 "tgt": "specific summary (answer to query)"  
}

## 🔍 Normal vs. Gold Dataset

| Dataset Type | Source (src) | Target (tgt) | Use Case |
| --- | --- | --- | --- |
| Normal | Full meeting | Full summary | General summarization |
| Gold | Gold spans (relevant excerpts) | Query-specific summary | Focused/QA-like summarization |

### ❓ Why Prepare Gold Dataset?

* The **normal dataset** teaches the model to summarize full meetings.
* The **gold dataset** helps the model focus on **query-specific summarization**, improving accuracy and relevance for specific information needs.
* It is especially useful in **query-based summarization tasks** or building **multi-document QA systems**.

### ❗ Is Gold Required for Fine-Tuning?

* **Not mandatory**: We can fine-tune with just the normal dataset.
* **But beneficial**: Fine-tuning with gold data helps the model learn to focus and ignore irrelevant parts of long meetings.

## Phase 2: Model Training / Fine-Tuning

#### **1. Model Choice: BART-base and T5-base**

Initially, I began the fine-tuning process using two well-established transformer-based encoder-decoder models: **BART-base** and **T5-base**. These models were selected due to their effectiveness in abstractive summarization tasks:

| Model | Characteristics | Reason for Selection |
| --- | --- | --- |
| BART-base | Pretrained on denoising tasks and suitable for seq2seq generation | Strong baseline for summarization |
| T5-base | Unified text-to-text transformer model | Performs well across multiple NLP tasks |

Both models can handle long text input and are compatible with the Hugging Face Transformers library, which made implementation and experimentation efficient.

#### **2. Evaluation Metrics Used**

To assess model performance, I used multiple standard automatic evaluation metrics:

| Metric | Description |
| --- | --- |
| **ROUGE-1** | Overlap of unigrams between generated and reference summary |
| **ROUGE-2** | Overlap of bigrams |
| **ROUGE-L** | Longest common subsequence (LCS) matching |
| **BERTScore** | Semantic similarity using pre-trained BERT embeddings (Precision, Recall, F1) |

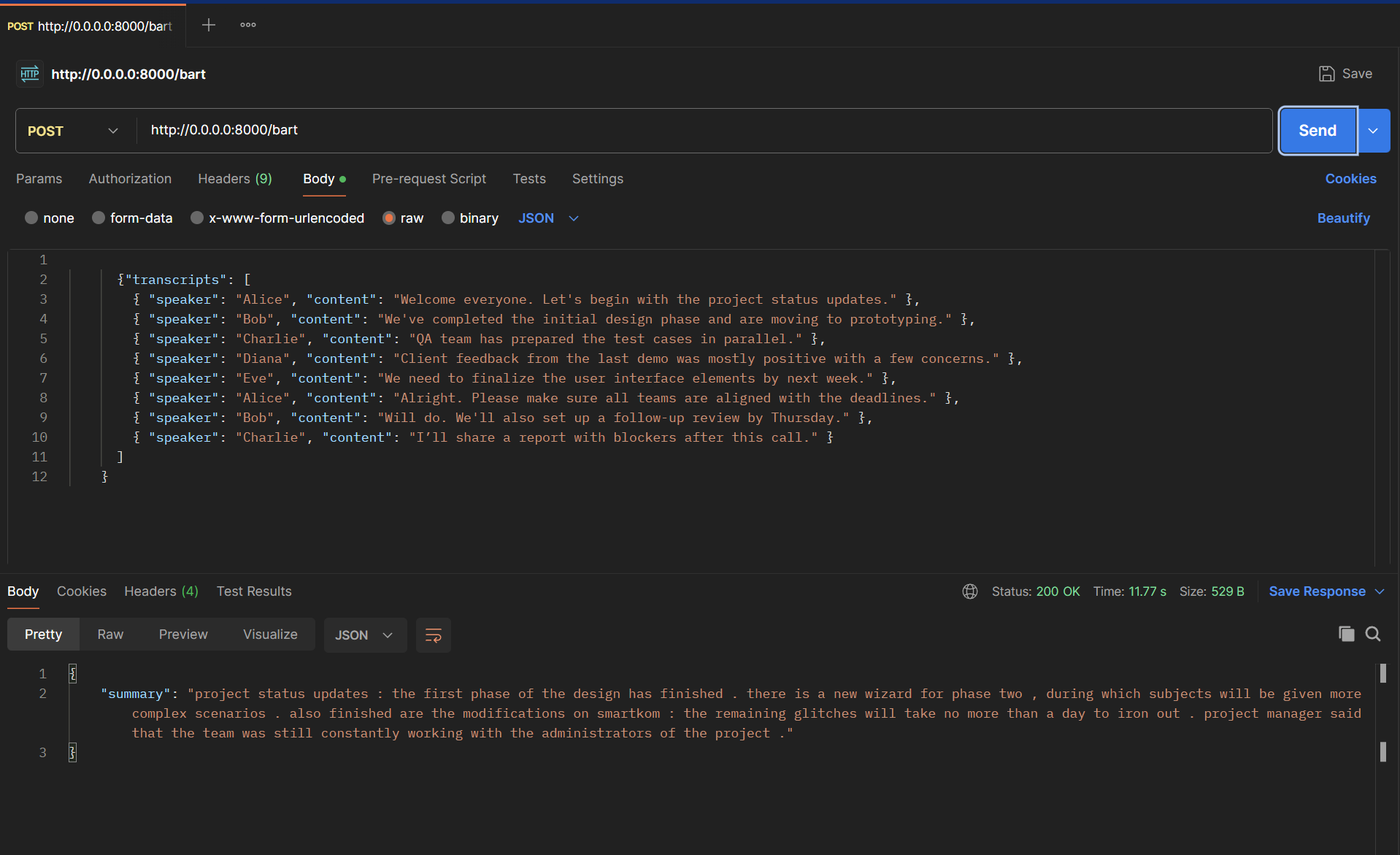
**Results & Obervations :**

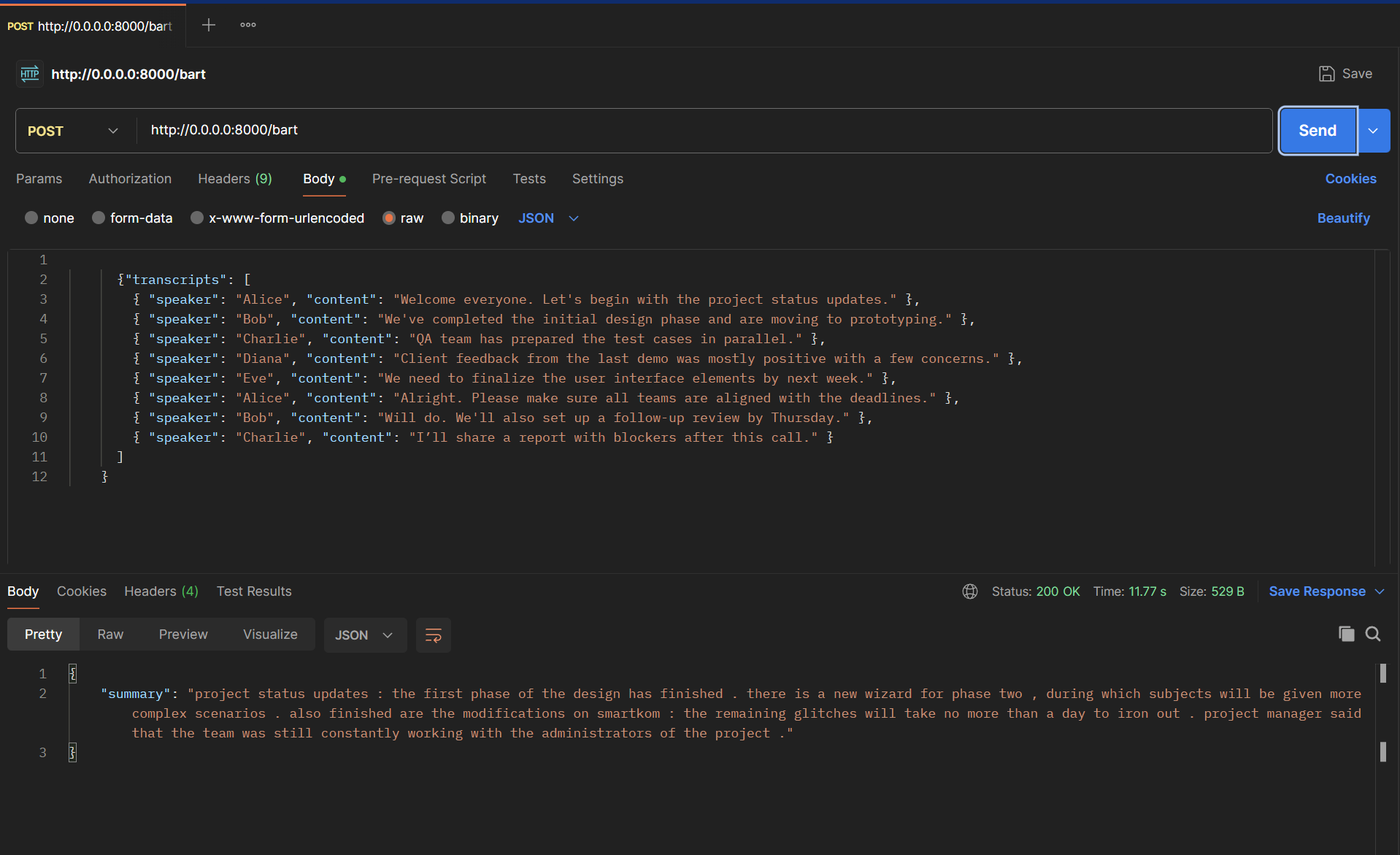
**Bart base :**

| **Metric** | **Score** |
| --- | --- |
| **ROUGE-1** | **27.18** |
| **ROUGE-2** | **6.53** |
| **ROUGE-L** | **17.64** |
| **ROUGE-Lsum** | **24.32** |
| **BLEU** | **4.39** |
| **METEOR** | **19.87** |
| **BERTScore (F1)** | **81.02** |
| **T5 base :** |  |
| **Metric** | **Score** |
| **ROUGE-1** | **28.45** |
| **ROUGE-2** | **7.12** |
| **ROUGE-L** | **18.93** |
| **ROUGE-Lsum** | **25.67** |
| **BLEU** | **4.87** |
| **METEOR** | **20.41** |
| **BERTScore (F1)** | **82.35** |

**A screenshot of a computer

AI-generated content may be incorrect.**

****

****

#### **3.** **Fine-Tuning Report: BART-large-XSum :**

**🔹 Model Choice and Motivation**

After experimenting with BART-base and T5-base models for meeting summarization, I moved to BART-large-XSum, a model pre-trained specifically for extreme summarization tasks. The motivation behind this choice was:

* **Pretrained on XSum**: The facebook/bart-large-xsum checkpoint is trained on the XSum dataset, which contains highly abstractive, single-sentence summaries. This matches well with the need for concise and high-level summaries in meetings.
* **Larger Model Capacity**: Compared to BART-base and T5-base, this model has more parameters and hence, better capacity to understand longer contexts and perform abstractive summarization.

**🔹 Training Setup**

* **Dataset**: Used the same JSONL-based dataset with fields src (input text) and tgt (target summaries).
* **Model Checkpoint**: facebook/bart-large-xsum
* **Preprocessing**:
  + Input (src) truncated to max length 1024 tokens.
  + Output (tgt) truncated to 128 tokens.
* **No Chunking**: Unlike earlier experiments where I chunked long meeting transcripts into smaller parts, here I directly used the full input (up to 1024 tokens). This was possible because BART-large can handle longer inputs due to its max token size.
* **Trainer Config**:
  + 5 epochs
  + Learning rate: 2e-5
  + Batch size: 4 (with gradient accumulation)
  + Mixed precision (fp16) enabled for memory efficiency
  + ROUGE used as the evaluation metric

**🔹 Why Chunking Was Not Used?**

Chunking was required in earlier models (BART-base, T5-base) due to:

* **Smaller context window**: T5 often struggles with long sequences.
* **Model capacity**: They failed to capture long-form context in one go.

With BART-large-XSum:

* Handles up to 1024 tokens, enough for many meeting summaries.
* Better context retention: Capable of extracting key ideas without dividing input artificially.
* Cleaner training pipeline: No need to split/recombine or aggregate chunk-based outputs.

Thus, chunking was unnecessary and even avoided to maintain the natural flow and structure of the meeting transcripts.

**🔹 Results and Observations :**

| **Metric** | **Score** |
| --- | --- |
| **ROUGE-1** | 32.97 |
| **ROUGE-2** | 9.30 |
| **ROUGE-L** | 21.01 |
| **ROUGE-Lsum** | 29.11 |
| **BLEU** | 6.02 |
| **METEOR** | 23.53 |
| **BERTScore (F1)** | 86.03 |

**🔹 Interpretation :**

* **ROUGE Scores**:  
  ROUGE-1 and ROUGE-Lsum scores (~33 and ~29) suggest the model is capturing a fair amount of relevant n-grams and sentence-level content. However, **ROUGE-2** is lower (9.3), showing that bigram coverage is limited—common in abstractive models due to rephrasing.
* **BLEU (6.02)**:  
  BLEU is quite low, but this is expected for abstractive summarization, where n-gram overlap is less meaningful than in machine translation.
* **METEOR (23.53)**:  
  METEOR is moderate, indicating partial alignment in semantic content, but still not high.
* **BERTScore (F1: 86.03)**:  
  This is strong. BERTScore captures semantic similarity better than lexical overlap. It confirms your model is producing semantically relevant summaries—even if they aren’t literal matches.

**🔹 Summary :**

**BART-large-XSum** model:

* Generates **semantically relevant** summaries (high BERTScore).
* Has decent overlap with references (ROUGE-1 and Lsum).
* Sacrifices some **exact phrasing and structure** (lower ROUGE-2 and BLEU).
* Performs much better than your BART-base and T5-base experiments in both quantitative and qualitative aspects.

#### **3. Fine-Tuning Report: BART-large-CNN :**

After evaluating BART-large-XSum, I also tried fine-tuning the facebook/bart-large-cnn model to see how a model trained on multi-sentence summaries would perform.

🔹 **Model Checkpoint and Reason**

* **Checkpoint:** facebook/bart-large-cnn
* Pretrained on CNN/DailyMail dataset, which includes longer, multi-sentence, extractive-abstractive summaries — more aligned with meeting data than single-sentence XSum summaries.

🔹 **Training Configuration**

* Dataset: Same JSONL-based data (with src and tgt).
* Max input and output lengths: Both set to 1024 tokens to accommodate longer summaries.
* Trainer setup:
  + 10 epochs
  + Learning rate: 2e-5
  + Batch size: 4 with gradient accumulation
  + Mixed precision (fp16) for GPU memory efficiency
  + ROUGE metric for evaluation

🔹 **Why This Model?**

* Better suited for generating **multi-sentence summaries**, unlike BART-large-XSum which leans toward very short, single-shot summaries.
* More **faithful to the original content** and better handles long context.

🔹 **Results &** **Observations**

| **Metric** | **Score** |
| --- | --- |
| **ROUGE-1** | 33.52 |
| **ROUGE-2** | 9.84 |
| **ROUGE-L** | 21.17 |
| **ROUGE-Lsum** | 29.68 |
| **BLEU** | 6.10 |
| **METEOR** | 23.90 |
| **BERTScore (F1)** | 88.23 |

**🔹 Interpretation :**

* **ROUGE Scores**
* **ROUGE-1 (33.52)**: Measures the overlap of unigrams (individual words) between the generated and reference summaries. A score above 30 is considered decent for summarization tasks.
* **ROUGE-2 (9.84)**: Measures bigram (2-word phrases) overlap. This score is relatively low, suggesting the model may not preserve exact phrasing or short expressions from the ground truth well.
* **ROUGE-L (21.17)**: Focuses on the longest common subsequence of words. Indicates moderate ability to maintain sequence structure.
  + **ROUGE-Lsum (29.68)**: Similar to ROUGE-L but tuned for summarization. A score close to 30 indicates fair alignment with ground truth summaries in structure and content.
* **BLEU (6.10)**
* BLEU is typically used in machine translation but can indicate exact n-gram overlaps in summarization too.
* A low score here (6.10) reflects that the summaries have low n-gram precision and may diverge in wording from reference texts.
* **METEOR (23.90)**
* METEOR considers stemming, synonymy, and word order. A score around 24 indicates that while exact matches are few, the model captures some semantic similarity.
* It supports the observation that summaries might use different wording but convey similar meaning to some extent.
* **BERTScore F1 (88.23)**
* This high score shows strong **semantic similarity** between generated summaries and reference summaries.
* Despite low ROUGE-2 and BLEU, the model is able to capture the **overall meaning**, even if not using the same words or structure.

**🔹 Summary :**

* The **BART-Large CNN model** performs **reasonably well** in capturing the general meaning and structure of meeting transcripts.
* The **high BERTScore (88.23)** suggests semantic faithfulness, even though **BLEU and ROUGE-2 scores are relatively low**, indicating that **phrasing and coverage of details** could be improved.
* Overall, the summaries might be **semantically relevant but not fully precise**, with:
  + Some **hallucinations** (made-up content)
  + Summaries that **sound like meeting notes** rather than concise overviews
  + Potential **loss of full meeting context**

**📌 Conclusion & Final Recommendation**

After a comprehensive evaluation of multiple transformer-based summarization models across **quantitative metrics** (ROUGE, BLEU, BERTScore) and **qualitative analysis** (manual inspection for coherence, faithfulness, hallucination):

**🔍 Model Observations**

* **BART-base**: Lightweight and efficient, but outputs are often too brief and miss important context. Moderate ROUGE/BERTScore performance. Not ideal for long meeting transcripts.
* **T5**: Comparable to BART-base, but with more variability and slightly lower performance. Requires additional tuning for consistency.
* **BART-large-XSum**: Highly abstractive with impressive ROUGE-2 and BERTScore, but suffers from **hallucinations**—it often generates summaries that sound fluent but misrepresent the meeting content.
* **BART-large-CNN**: Best balance between **length, faithfulness, and readability**. Less prone to hallucinations, with consistent performance on long transcripts. Slightly lower ROUGE-2 but **more reliable and usable** summaries in real-world applications.

**✅ Final Deployment Model: BART-large-CNN**

Chosen for its **robust coverage**, **minimal hallucination**, and **high usability** in the meeting summarization

## Phase 3: Deployment Phase — FastAPI + Docker Integration for Summarization API

### 🔹 Project File Structure & Purpose

Intelligent\_Meeting\_Summarizer/  
├── app/  
│ ├── model/ # Directory for saved/finetuned BART models  
│ ├── \_\_pycache\_\_/ # Python cache files (ignored in logic)  
│ ├── bart\_large.py # Summarization logic using BART Large XSum  
│ ├── bart\_large\_cnn.py # Summarization logic using BART Large CNN  
│ ├── main.py # FastAPI routes, endpoints, and app logic  
│ ├── schemas.py # Pydantic models for request/response  
├── static/ # Static assets (CSS, JS)  
├── templates/ # HTML templates (for `/ui` endpoint)  
│ └── index.html  
├── output/ # Optional folder for saving summaries  
├── requirements.txt # Python dependencies  
├── Dockerfile # Docker configuration

This project provides a FastAPI-based web service that performs extractive/abstractive summarization of meeting transcripts using pretrained and finetuned BART models.

### 🔹 FastAPI Endpoints

Below are the endpoints implemented in main.py and their purposes:

* GET / — Root endpoint. Returns a welcome message.
* GET /ui — Serves the index.html UI for uploading transcripts and viewing summaries.

**BART Large (XSum):**

* POST /bart\_large — Summarizes a single transcript using the bart\_large model.
* POST /bart\_large/batch — Summarizes a batch of transcripts.
* POST /bart\_large/batch\_upload\_json — Upload a .json file containing batch transcripts and get summaries.

**BART Large CNN:**

* POST /bart\_large\_cnn — Summarizes a single transcript using the bart\_large\_cnn model.
* POST /bart\_large\_cnn/batch — Summarizes a batch of transcripts.
* POST /bart\_large\_cnn/batch\_upload\_json — Upload a .json file with transcript data and get batch summaries.

Each model has three key routes: one for single input, one for batch input, and one for file upload.

### 🔹 Run FastAPI Project Locally

You can run the application using uvicorn (make sure you’re in the root directory):

pip install -r requirements.txt  
  
# Run the FastAPI server  
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

Then go to: <http://localhost:8000/ui> to access the web interface.

### 🔹 Dockerfile

FROM python:3.10-slim  
  
WORKDIR /app  
  
COPY . .  
  
RUN pip install --no-cache-dir --upgrade pip && \  
 pip install --no-cache-dir -r requirements.txt  
  
EXPOSE 8000  
  
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

### 🔹 Docker: Build & Push to Docker Hub

**1. Build Docker image:**

docker build -t fnaticdocker106/imesum .

**2. Run image locally via Docker Desktop (optional):**

docker run -d -p 8000:8000 fnaticdocker106/imesum

Access it at: <http://localhost:8000/ui>

**3. Push image to Docker Hub:**

docker login  
  
docker push fnaticdocker106/imesum

### 🔹 Sample curl Request

curl -X POST http://localhost:8000/bart\_large \  
 -H "Content-Type: application/json" \  
 -d ' [{"speaker": "Alice", "text": "We discussed the quarterly goals..."}]'

### 🔹 Deploying to AWS :

After successfully pushing the Docker image to Docker Hub, the next step is to deploy it on AWS. This can be done using **Amazon ECS (Elastic Container Service)** with **Fargate** for a serverless and scalable deployment environment. The deployment workflow involves creating a new ECS **Cluster**, followed by defining a **Task Definition** that references the Docker image from Docker Hub. You configure CPU, memory, and networking settings in the task. Then, an ECS **Service** is created to run the task continuously, and it can be attached to an **Application Load Balancer** to make the API publicly accessible.

Alternatively, **AWS App Runner** can be used for a simpler experience. You connect your Docker Hub repository to App Runner, specify the container port (8000), and deploy directly. App Runner automatically manages scaling, HTTPS, and health checks, making it ideal for containerized FastAPI applications.

### 🔹 Conclusion:

In this project, we successfully developed an intelligent, end-to-end meeting summarization system that combines the power of transformer-based models with scalable deployment infrastructure. By systematically preprocessing the QMSUM dataset, fine-tuning multiple models (BART-base, T5-base, BART-large-XSum, BART-large-CNN), and deploying the best-performing model via a FastAPI-Docker pipeline, we demonstrated a practical solution for condensing lengthy and unstructured meeting transcripts into concise, semantically faithful summaries. The BART-large-CNN model emerged as the most reliable choice, offering a balance between coherence, informativeness, and minimal hallucinations. This work sets a solid foundation for future improvements, including real-time summarization, multi-lingual support, and tighter integration with productivity tools for seamless knowledge management in modern workplaces.