Intelligent Assistant for Solution Design: Capstone Project Documentation

Project Overview

Problem Statement

In the business world, meetings are a cornerstone for discussing requirements, plans, and roadmaps. Post-meeting, there is often a need for internal discussions within organizations like TCS to process the outcomes effectively. Key tasks include identifying problem statements, designing solutions, recommending technology stacks, and summarizing key points and action items. The challenge lies in automating this process to reduce manual effort, improve accuracy, and ensure actionable insights are captured systematically.

The expected result is a user-friendly UI interface where users can upload a video or transcript of a meeting. The solution should generate a detailed report, including a summary, identified problem statements, proposed solution designs, and recommended technology stacks.

Objectives

- Develop an intelligent assistant capable of processing meeting audio, video, or text transcripts.
- Extract key insights such as summaries, problem statements, action items, and solutions.
- Propose a technology stack tailored to the identified problems and solutions.
- Provide a professional report in Markdown and PDF formats for easy sharing and review.

Team Contributions and Milestones

Team Member	Contribution	Milestone Achieved
Shreesha B	- Research and Technology Selection - Local LLM experiments - Chunk Handling - Prompt Engineering - Recording Meetings - Structured report generation - Testing and debugging	- Completed Research and Technology Selection (Day 2) - Selected best LLM model (Day 7) - Generated structured reports (Day 9) - System tested and debugged (Day 11)
Ojas Soni	- Research and Technology Selection - Implementing File Upload and Speech-to-Text Conversion - Designing Scripts - Recording Meetings - Prompt Engineering - Testing and debugging	- Completed Research and Technology Selection (Day 2) - Implemented Speech-to-Text feature (Day 5) - System tested and debugged (Day 11)

Team Member	Contribution	Milestone Achieved
Sree Nidhi L	- Research and Technology Selection - Architecture Design - Recording Meetings - Frontend Designing - Prompt Engineering - Documentation - System Integration	- Completed Research and Technology Selection (Day 2) - Designed architecture (Day 3) - Completed frontend design (Day 10) - Integrated entire system (Day 11)

🎨 Color Legend

- **Shreesha B**: Lead AI Engineer
- **Ojas Soni**: Lead Software Developer
- **Sree Nidhi L**: Lead Architect and Documentarian

🎉 Key Milestones Overview

- **Day 2**: Research and tech selection completed (All team members).
- **Day 3**: Architecture design finalized (Sree Nidhi L).
- **Day 5**: Speech-to-Text conversion implemented (Ojas Soni).
- **Day 7**: Best LLM model selected (Shreesha B).
- **Day 9**: Structured reports generated (Shreesha B).
- **Day 11**: System fully integrated and tested (All team members).

Project Timeline

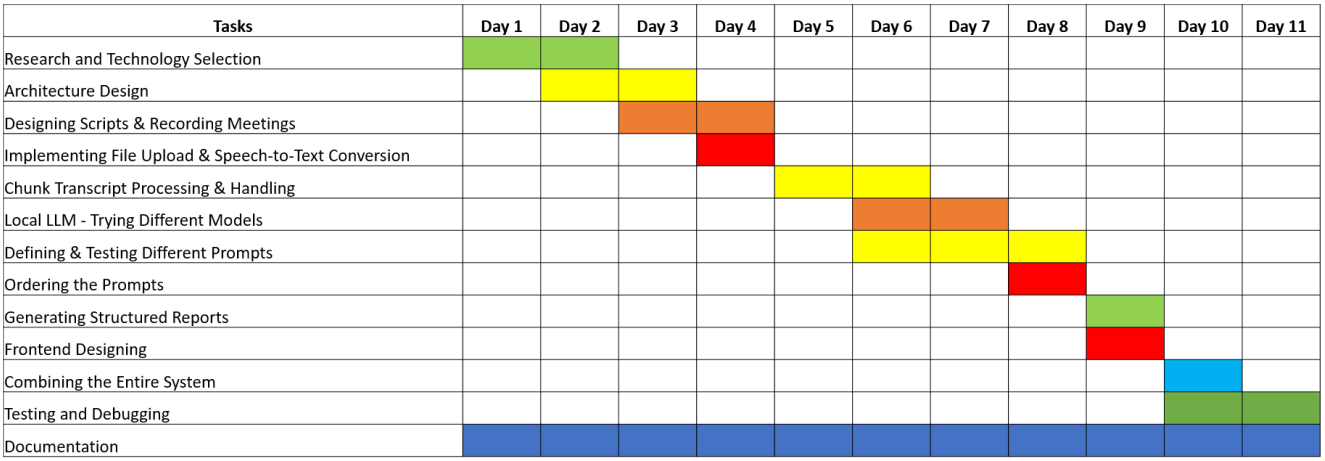

Figure: Project timeline spanning 11 days.

System Architecture

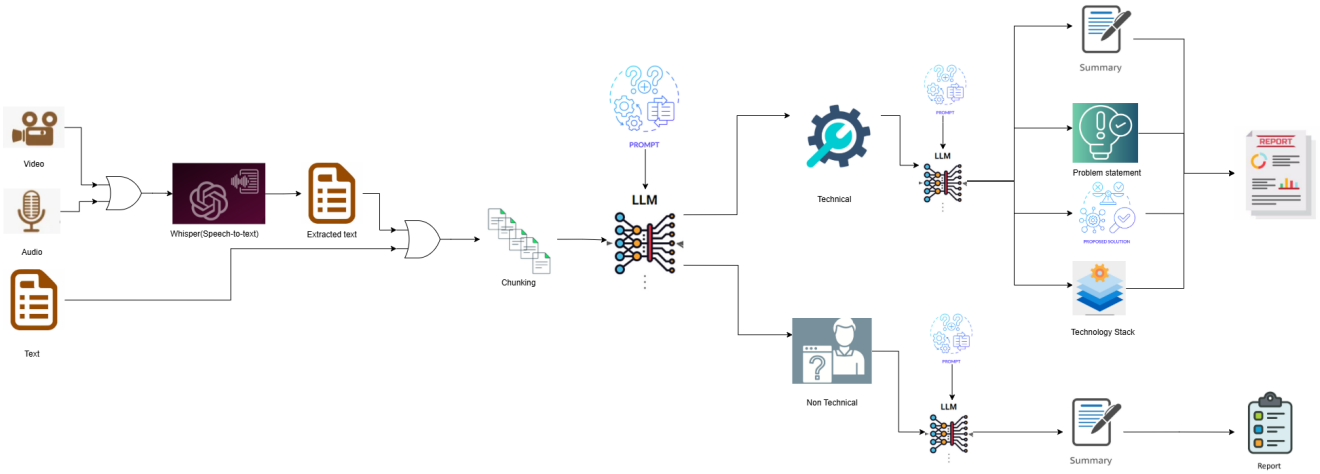

Figure: Workflow for text extraction and report generation.

The architecture diagram illustrates the end-to-end workflow of the Intelligent Assistant for Solution Design. Below is a detailed breakdown of the components and flow:

1. **Input Layer**:
    - **Sources**: The system accepts video (mp4, webm), audio (mp3), or text (txt) files as input.
    - **User Interface**: A Gradio-based UI allows users to upload files and initiate the processing pipeline.
2. **Transcription Layer**:
    - **Whisper Model (Speech-to-Text)**: The Whisper model (`base.en`) is used to transcribe audio or video files into text. If a text file is uploaded, it is directly processed.
    - **Output**: Extracted text from the input file.
3. **Text Processing Layer**:
    - **Chunking**: The transcript is divided into manageable chunks (4000 words with 1000-word overlap) to handle large inputs within the context limits of the language model.
    - **Output**: A list of transcript chunks.
4. **Meeting Classification**:
    - **LLM (Qwen2.5)**: The first chunk is analyzed using a classification prompt to determine if the meeting is technical or non-technical.
    - **Output**: Meeting type ("Technical" or "Non-technical").
5. **Information Extraction**:
    - **LLM (Qwen2.5)**: Depending on the meeting type, appropriate prompts (technical or non-technical) are used to extract summaries, key points, action items, problem statements, solution components, technology suggestions, and challenges.
    - **Output**: Extracted information per chunk in a structured format.
6. **Report Generation**:
    - **LLM (Qwen2.5)**: The extracted information from all chunks is combined and processed to

generate a comprehensive Markdown report tailored to the meeting type.
    - **Output**: A detailed Markdown report.

7. **Output Layer**:
    - **Markdown File**: The report is saved as a Markdown file (`report.md`).
    - **PDF Conversion**: The Markdown file is converted to a PDF (`report.pdf`).
    - **UI Output**: The report is displayed in the Gradio interface, with options to download the Markdown and PDF files.

Why Choose Whisper `base.en` for Transcription?

The choice of Whisper `base.en` for transcription was informed by the analysis in the obtained graph. The graph compares various models (BASE.EN WHISPER, WAV2VEC2, LARGE-V3-TURBO [WHISPER], VOSK-MODEL-EN-US-0.22-LGRAPH, VOSK-MODEL-EN-US-0.22) on Word Error Rate (WER) and Character Error Rate (CER). BASE.EN WHISPER achieved a WER of 34.06 and a CER of 15.24, which are competitive compared to larger models like LARGE-V3-TURBO (WER: 64.86, CER: 34.45) while requiring significantly fewer computational resources. Since the project prioritizes efficiency on varied hardware (including CPU setups), `base.en` provides a balanced trade-off between accuracy and resource usage, making it ideal for this application.

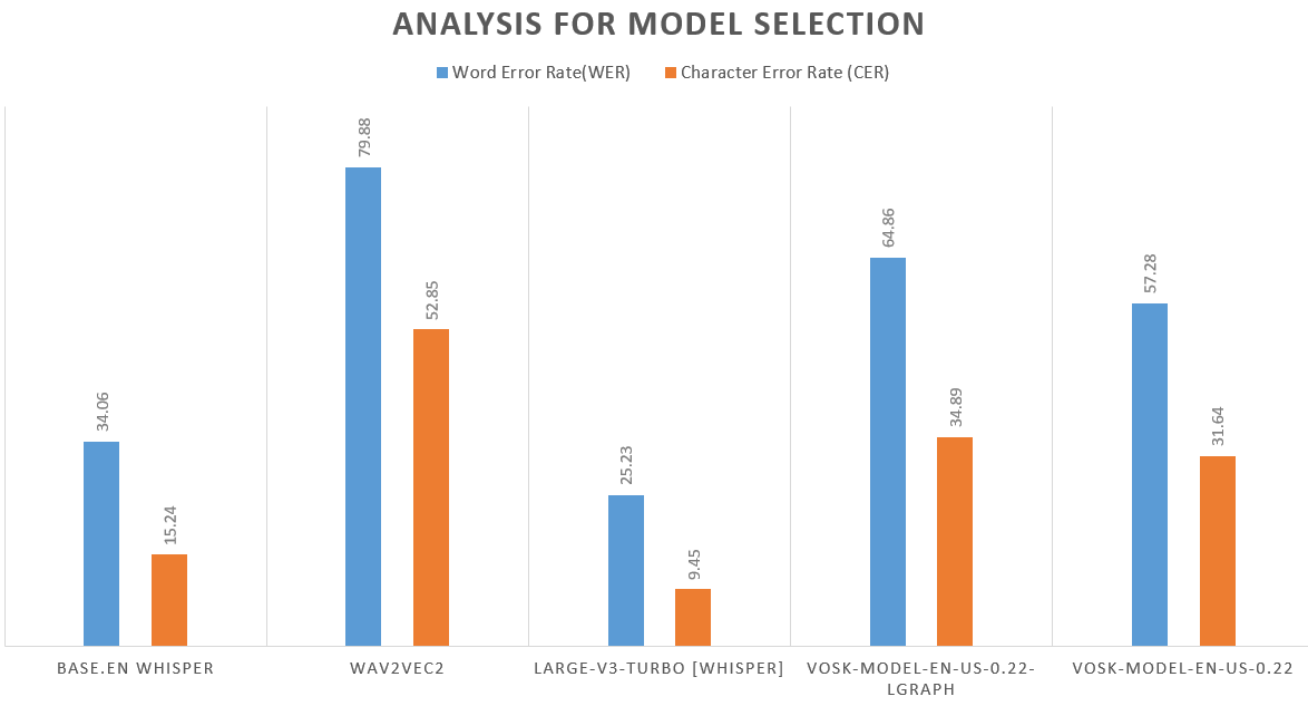

Figure: Comparison of Word Error Rate (WER) and Character Error Rate (CER) for different models.

Why Choose Qwen2.5 for Language Processing?

Qwen2.5, developed by Alibaba Cloud, was selected as the language model for this project due to its strong performance in natural language understanding and generation, particularly for structured tasks like information extraction and report generation. The 7B model (or 3B for CPU) offers a good balance of capability and efficiency, making it suitable for processing large meeting transcripts while maintaining high-quality output. Qwen2.5 excels in handling long contexts (up to 8192 tokens in this setup) and following complex prompts, which is critical for extracting nuanced information like problem statements

and solution designs. Additionally, its open-source availability via Ollama ensures accessibility and ease of deployment, aligning with the project's goal of creating a scalable and maintainable solution.

Setup and Installation

Prerequisites

- **Operating System**: Windows, macOS, or Linux.
- **Hardware**:
  - GPU (optional, recommended for faster processing with Whisper and Qwen2.5).
  - Minimum 8GB RAM (16GB recommended for GPU usage).
- **Software**:
  - Python 3.8 or higher.
  - FFmpeg (for audio/video processing).
  - Git (for cloning repositories, if needed).

Step-by-Step Installation

1. **Install FFmpeg**:
   - **Windows**: Download from the official FFmpeg website, extract, and add to PATH.
   - **macOS**: `brew install ffmpeg`
   - **Linux**: `sudo apt-get install ffmpeg`
   - Verify: `ffmpeg -version`
2. **Install Python Dependencies**: Create a virtual environment (optional but recommended):

   ```
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   ```

   Install required packages:

   ```
   pip install ffmpeg-python openai-whisper torch ollama gradio mdpdf
   ```

3. **Install and Configure Ollama**:
   - Download and install Ollama from the official website (https://ollama.ai).
   - Pull the Qwen2.5 model:
     - For GPU setups (7B model): `ollama pull qwen2.5`
     - For CPU setups (3B model): `ollama pull qwen2.5:3b`
   - Verify: `ollama list` (should show `qwen2.5` or `qwen2.5:3b`).
4. **Verify Whisper Installation**: Whisper is installed via `openai-whisper`. The `base.en` model will be automatically downloaded on first use.

Project Setup

1. Clone or create the project directory:

```
mkdir intelligent-assistant
cd intelligent-assistant
```

2. Save the provided code as `app.py` in the project directory.
3. Run the application:

```
python app.py
```

This will launch the Gradio interface, accessible via a browser URL (e.g., `http://127.0.0.1:7860`).

Code Explanation

The code is structured to handle the entire workflow from file input to report generation. Below is a detailed breakdown of the key components and functions:

1. **Imports and Setup**

```
import gradio as gr
import whisper
import ollama
import os
import torch
import re
import subprocess
from pathlib import Path
import shutil
```

- **Purpose**: Import necessary libraries for UI (Gradio), transcription (Whisper), language processing (Ollama), and file handling.
- **Whisper Model Loading**:

```
device = "cuda" if torch.cuda.is_available() else "cpu"
whisper_model = whisper.load_model("base.en", device=device)
```

  - Detects if a GPU is available and loads the Whisper `base.en` model accordingly.

2. **Transcription Function (`transcribe_file`)**

```
def transcribe_file(file, progress=gr.Progress()):
    progress(0, desc="Starting transcription...")
```

```
    try:
        if file.name.endswith('.txt'):
            with open(file.name, 'r', encoding='utf-8') as f:
                transcript = f.read()
        elif file.name.endswith(('.mp4', '.webm', '.mp3')):
            progress(0.2, desc="Transcribing audio/video...")
            result = whisper_model.transcribe(file.name, fp16=(device ==
"cuda"))
            transcript = result["text"]
        else:
            raise ValueError("Unsupported file type...")
        transcript = re.sub(r'\s+', ' ', transcript).strip()
        filler_words = r'\b(um|uh|like|you know|so|basically)\b'
        transcript = re.sub(filler_words, '', transcript, flags=re.IGNORECASE)
        transcript = re.sub(r'\s+', ' ', transcript).strip()
        return transcript
    except Exception as e:
        raise Exception(f"Transcription error: {str(e)}")
```

- **Purpose**: Transcribes audio/video files or reads text files.
- **Details**:
  - Supports `.txt`, `.mp4`, `.webm`, and `.mp3` files.
  - Uses Whisper for audio/video transcription with FP16 precision on GPU.
  - Cleans the transcript by removing filler words (e.g., "um", "uh") and normalizing whitespace.

3. **Chunking Function (`chunk_transcript`)**

```
def chunk_transcript(transcript, chunk_size=4000, overlap=1000,
progress=gr.Progress()):
    words = transcript.split()
    chunks = []
    start = 0
    total_chunks = max(1, (len(words) // (chunk_size - overlap)) + 1)
    for i in range(total_chunks):
        end = min(start + chunk_size, len(words))
        chunk = ' '.join(words[start:end])
        if chunk.strip():
            chunks.append(chunk)
        start = end - overlap if end < len(words) else len(words)
        progress((i + 1) / total_chunks, desc=f"Chunk {i + 1} of
{total_chunks} created.")
    return chunks
```

- **Purpose**: Splits the transcript into chunks to fit within the context limits of Qwen2.5 (8192 tokens).
- **Details**:
  - Chunk size is set to 4000 words with a 1000-word overlap to ensure continuity.
  - Progress tracking is integrated for user feedback.

4. **Prompts for LLM Processing**

- **Classification Prompt**:
  - Determines if the meeting is technical or non-technical based on the first chunk.
- **Technical Extraction Prompt**:
  - Extracts detailed information for technical meetings, including summaries, key points, action items, problem statements, solution components, technology suggestions, and challenges.
- **Non-Technical Extraction Prompt**:
  - Extracts summaries, key points, and action items for non-technical meetings.
- **Report Generation Prompts**:
  - Separate prompts for technical and non-technical meetings to generate structured Markdown reports.

5. **LLM Processing Function (`process_with_ollama`)**

```
def process_with_ollama(prompt, num_predict=8192, num_ctx=8192,
progress=gr.Progress()):
    try:
        response = ollama.generate(
            model="qwen2.5",
            prompt=prompt,
            options={"num_predict": num_predict, "num_ctx": num_ctx}
        )
        return response.get('response', '').strip()
    except Exception as e:
        raise Exception(f"Ollama processing error: {str(e)}")
```

- **Purpose**: Interfaces with the Qwen2.5 model via Ollama for classification, extraction, and report generation.
- **Details**:
  - Configures the maximum tokens to predict (`num_predict`) and context size (`num_ctx`).
  - Handles errors gracefully with detailed feedback.

6. **Report Generation Function (`generate_report`)**

```
def generate_report(file, progress=gr.Progress()):
    transcript = transcribe_file(file, progress)
    chunks = chunk_transcript(transcript, progress=progress)
    classification =
process_with_ollama(classification_prompt.format(chunk=chunks[0]), ...)
    meeting_type = classification.strip().lower()
    extracted_chunks = []
    for chunk in chunks:
        if meeting_type == "technical":
            extracted_chunk =
process_with_ollama(technical_extract_prompt.format(chunk=chunk), ...)
        else:
```

```
            extracted_chunk =
process_with_ollama(non_technical_extract_prompt.format(chunk=chunk), ...)
        extracted_chunks.append(extracted_chunk)
    combined_chunks = combine_chunks(extracted_chunks)
    if meeting_type == "technical":
        report_md =
process_with_ollama(technical_report_prompt.format(combined_chunks=combined_ch
unks), ...)
    else:
        report_md =
process_with_ollama(non_technical_report_prompt.format(combined_chunks=combine
d_chunks), ...)
    md_path = save_markdown_report(report_md, progress)
    pdf_path = convert_md_to_pdf(md_path, progress)
    return report_md, md_path, pdf_path
```

- **Purpose**: Orchestrates the entire workflow from transcription to report generation.
- **Details**:
    - Handles errors at each step and provides user feedback via progress updates.
    - Supports both Markdown and PDF output formats.

7. **Gradio Interface**

```
with gr.Blocks(title="Intelligent Assistant for Solution Design") as app:
    gr.Markdown("# Intelligent Assistant for Solution Design")
    file_input = gr.File(label="Upload Video, Audio, or Transcript",
file_types=[".mp4", ".webm", ".mp3", ".txt"])
    generate_btn = gr.Button("Generate Report", variant="primary")
    report_output = gr.Markdown(label="Generated Report")
    md_output = gr.File(label="Download Report as Markdown (.md)")
    pdf_output = gr.File(label="Download Report as PDF (.pdf)")
    generate_btn.click(fn=generate_report, inputs=file_input,
outputs=[report_output, md_output, pdf_output])
app.launch(share=True)
```

- **Purpose**: Provides a user-friendly interface for uploading files and viewing/downloading reports.
- **Details**:
    - Displays warnings if `mdpdf` is not installed.
    - Includes progress feedback for long-running tasks.

Solution Design

Transcription and Preprocessing

- **Whisper `base.en`**: Chosen for its balance of accuracy and efficiency, as discussed earlier.
- **Text Cleaning**: Removes filler words and normalizes whitespace to improve the quality of the transcript for LLM processing.

Meeting Classification

- Uses a classification prompt to determine the meeting type, ensuring the appropriate extraction and reporting logic is applied.

Information Extraction

- **Technical Meetings**:
  - Extracts detailed information, including problem statements and solution components, to support solution design.
  - Suggests technologies based on the extracted context (e.g., OpenCV for image processing).
- **Non-Technical Meetings**:
  - Focuses on summaries, key points, and action items, as technical details are not relevant.

Report Generation

- Generates a structured Markdown report tailored to the meeting type.
- For technical meetings, includes detailed sections on problem statements, solution designs, and technology stacks.
- Ensures reports are professional and comprehensive, spanning 3-4 pages for technical meetings.

Technology Stack

- **Python**: Core programming language for the project due to its extensive ecosystem and support for AI/ML libraries.
- **Whisper (`base.en`)**: For audio/video transcription, chosen for its efficiency and accuracy.
- **Ollama with Qwen2.5**: For language processing, selected for its strong NLP capabilities and open-source availability.
- **Gradio**: For building the user interface, offering a simple and interactive web-based solution.
- **FFmpeg**: For handling audio/video file processing, required by Whisper.
- **mdpdf**: For converting Markdown reports to PDF, ensuring professional output (optional).

Challenges and Limitations

- **Hardware Constraints**:
  - Users without GPUs may experience slower processing times, particularly for transcription and LLM inference.
  - The 3B Qwen2.5 model is used for CPU setups, which may have slightly lower accuracy compared to the 7B model.
- **Transcript Quality**:
  - Noisy audio or unclear speech can lead to transcription errors, impacting the quality of the extracted information.
  - Mitigation: Encourage users to upload high-quality recordings or pre-transcribed text.
- **Context Limitations**:
  - The chunking approach may miss cross-chunk context, potentially leading to fragmented problem statements or solutions.
  - Mitigation: The 1000-word overlap helps retain some continuity between chunks.

- **PDF Generation**:
  - Requires `mdpdf` to be installed, which may not be available on all systems.
  - Mitigation: Provide a warning in the UI and allow Markdown downloads as a fallback.

Future Enhancements

- **Improved Transcription**:
  - Explore larger Whisper models (e.g., `medium.en`) for better accuracy if hardware constraints are alleviated.
- **Advanced NLP**:
  - Fine-tune Qwen2.5 on meeting-specific datasets to improve extraction accuracy.
  - Incorporate multi-modal analysis (e.g., combining video visuals with audio) for richer insights.
- **UI Enhancements**:
  - Add real-time progress updates and error notifications.
  - Allow users to edit extracted information before report generation.
- **Scalability**:
  - Deploy the solution on a cloud platform (e.g., AWS) for better scalability and accessibility.
  - Implement batch processing for multiple files.

Conclusion

The Intelligent Assistant for Solution Design addresses the challenge of post-meeting analysis by automating the extraction of key insights and generating detailed reports. By leveraging Whisper for transcription, Qwen2.5 for language processing, and Gradio for the user interface, the solution provides a robust and user-friendly tool for business professionals. While there are challenges related to hardware constraints and transcript quality, the system offers a solid foundation for further enhancements and scalability.