AI Presentation Coach: Architecture Document

A Multi-Agent System for Communication Skill Analysis

Author:

MD Misbah Ur Rahman

B.Tech (Hons.), Chemical Engineering
Indian Institute of Technology Kharagpur

Abstract

This document details the architecture of the AI Presentation Coach, a prototype agent designed to automate the analysis of on-camera spoken performances. The system leverages a multi-agent, Orchestrator-Workers model to perform a multi-modal analysis of a user's presentation, covering vocal, visual, and content-based metrics. The core of the system is a fine-tuned language model that synthesizes this data into actionable, expert-level feedback. This document outlines the system's components, interaction flows, and the design rationale behind its construction.

Contents

1 Introduction: Automating the Feedback Loop **1**

2 High-Level Architecture: A Multi-Agent System **1**

3 Detailed Component Breakdown **2**
3.1 The Orchestrator Agent 2
3.2 Utility: The Audio Extractor 2
3.3 Worker: The Transcription Agent 2
3.4 Worker: The Vocal Analysis Agent 3
3.5 Worker: The Visual Analysis Agent 3
3.6 Worker: The Content Analysis Agent 3
3.7 Worker: The Synthesis Agent 4

4 Interaction Flow: A Step-by-Step Analysis **4**

5 Design Rationale & Conclusion **5**

September 13, 2025

1 Introduction: Automating the Feedback Loop

For any student, the ability to effectively communicate complex ideas is a critical skill for success. Whether presenting a project, facing a high-stakes internship interview, or delivering a monologue, the feedback loop is essential for improvement. However, this feedback is often slow, subjective, and difficult to obtain consistently. This project was born from a simple yet ambitious question: could this process be automated? Could an AI agent be built to serve as a personal, objective, and data-driven communication coach, available on-demand?

This document details the architecture of the **AI Presentation Coach**, an agent I designed and built to tackle this exact challenge. The agent's core mission is to automate the manual task of analyzing a solo on-camera spoken performance. It is engineered to **reason** about a user's communication, formulate a multi-faceted **plan** for analysis, and **execute** that plan using a suite of specialized AI models and algorithms. This system is a reflection of my belief in building technology that is empathetic, aware, and designed to augment human potential. I was driven by the challenge of creating not just a functional prototype, but a truly insightful system that could offer tangible value, and I am excited to present its architecture.

2 High-Level Architecture: A Multi-Agent System

To tackle the complexity of analyzing human communication, I decided against a monolithic design. A single, large agent would be difficult to build, debug, and improve. Instead, I architected the system as a **multi-agent collaboration**, a design that provides significant advantages in modularity, scalability, and specialization. This approach directly fulfills the optional "Planner + Executor" requirement of the assignment.

The core architectural pattern I implemented is formally known as **Role-Based Cooperation**, as detailed in the *Agent Design Pattern Catalogue*[1]. In our system, this pattern is realized through an **Orchestrator-Workers** model.

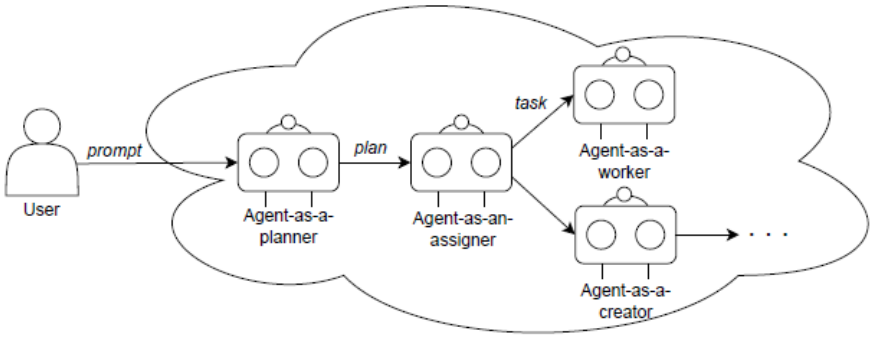

Figure 1: The formal "Role-Based Cooperation" pattern that our agent's architecture implements.

As illustrated in Figure 1, the workflow is managed by a central coordinating agent:

- **The Orchestrator Agent (The Planner):** This agent serves as the "brain" of the operation. When it receives a video, its sole responsibility is to reason about the request and generate a clear, sequential plan for the analysis. It then delegates the execution of each step to a specialized worker.
- **The Worker Agents (The Executors):** Each worker is a highly specialized, independent module designed to execute one specific analytical task. For example, the **Transcription-Worker** only handles speech-to-text, while the **Visual-Worker** only analyzes the video stream. These workers are custom tools that perform complex operations a generalist model cannot.

This separation of concerns is the system's greatest strength. It allowed me to develop and refine each analytical component in isolation, ensuring each part of the analysis is as robust and accurate as

[1]Liu, Yue, et al. "Agent Design Pattern Catalogue: A Collection of Architectural Patterns for Foundation Model Based Agents." arXiv preprint arXiv:2405.10467 (2024).

possible before being integrated into the whole. This design is not just functional; it is a flexible and powerful foundation for a truly intelligent system.

3 Detailed Component Breakdown

The agent's architecture is composed of a central Orchestrator and a suite of specialized worker agents and utilities. Each component was carefully selected and engineered to perform its task with precision, contributing to a final analysis that is comprehensive and data-driven.

3.1 The Orchestrator Agent

The **Orchestrator** is the central nervous system of the entire application. It is not a specialized analytical model itself, but rather the master controller that manages the entire workflow from start to finish, embodying the "Planner" role in our architecture.

- **Technology:** A core Python module (`orchestratorworker.py`) built with Python's `asyncio` capabilities.
- **Function:** Its function is purely procedural and strategic:
  1. It receives the initial video file from the API layer.
  2. It calls the **Audio Extractor** utility to standardize the audio format.
  3. It delegates analysis tasks to each **Worker Agent** in a logical sequence.
  4. It performs critical data marshalling, passing the output of one worker (e.g., the transcript) as the necessary input to subsequent workers.
  5. It aggregates the final structured metrics from all workers into a single, comprehensive data packet.
  6. It passes this complete data packet to the final **Synthesis-Worker** for interpretation.
  7. Finally, it manages the cleanup of any temporary files created during the process.

3.2 Utility: The Audio Extractor

The pipeline begins not with an agent, but with a critical utility. The **Audio Extractor** is a robust module responsible for data standardization.

- **Technology:** It leverages the industry-standard, command-line tool `ffmpeg`.
- **Function:** Its sole task is to take the user's uploaded video file (e.g., MP4, MOV) and extract the audio stream into a standardized, high-quality WAV format. This crucial first step ensures that all subsequent audio-processing workers receive a clean, consistent input format, which is vital for the reliability of the entire system.

3.3 Worker: The Transcription Agent

This is the first analytical agent in the pipeline, giving our system its "ears".

- **Technology:** OpenAI's **Whisper** model, specifically the `small.en` version, accessed via the Hugging Face Transformers library.
- **Function:** It takes the WAV file and performs speech-to-text transcription. The choice of the 'small.en' model was a deliberate trade-off, providing a significant accuracy improvement over smaller versions while remaining performant. This high-quality transcript forms the linguistic foundation for all further content analysis.

3.4 Worker: The Vocal Analysis Agent

This agent analyzes *how* the user spoke, providing metrics on the quality of the vocal delivery.

- **Technology:** The **Parselmouth** library, a Python interface for the powerful Praat phonetics software.
- **Function:** It performs a deep phonetic analysis of the audio file to extract objective metrics. This includes speaking pace (WPM), pitch variability (to measure monotony), pause frequency and duration (to measure hesitation), and the count of disfluent repetitions. This was one of the most exciting components to build, as it turns the subjective quality of "how someone sounds" into objective, actionable data.
- **Metrics Generated:**

  Speaking Pace (WPM): Words Per Minute, a measure of delivery speed.

  Pitch Variability (ST): The standard deviation of the vocal pitch in Semitones. A higher value indicates a more expressive, less monotone delivery.

  Pause Count: The number of significant silences (¿ 300ms) in the speech, indicating hesitation.

  Repetition Count: The number of immediately repeated words (e.g., "the the"), a sign of disfluency.

3.5 Worker: The Visual Analysis Agent

This agent serves as the "eyes" of our system, analyzing the video stream for non-verbal cues.

- **Technology:** Google's **MediaPipe** framework with OpenCV.
- **Function:** It performs a definitive, multi-metric analysis for visual engagement, including smile detection and gesture usage. Critically, I engineered it with a **graceful degradation** protocol for engagement tracking. In high-quality video, it uses a high-fidelity Tier 1 analysis of iris landmarks for precise gaze detection. If the video quality is too poor for this, it automatically falls back to a more robust Tier 2 heuristic based on head pose, ensuring a useful metric is always provided.
- **Metrics Generated:**

  Gaze Forward (%): (Tier 1) The percentage of time the user's pupils are detected to be looking forward. This is a high-fidelity proxy for eye contact.

  Head Pose Engagement (%): (Tier 2 Fallback) If iris tracking is unreliable due to video quality, this metric measures the percentage of time the user's head is oriented towards the camera.

  Smile Presence (%): The percentage of frames in which a smile is detected via mouth landmark geometry.

  Gesture Usage (%): The percentage of time hand gestures are detected in the frame.

3.6 Worker: The Content Analysis Agent

This agent analyzes the substance and clarity of the user's message.

- **Technology:** A hybrid approach using the lightweight `textstat` and `yake` libraries, combined with the powerful `facebook/bart-large-mnli` **zero-shot classification model** from Hugging Face.
- **Function:** It performs a multi-faceted linguistic analysis of the transcript. It calculates a Flesch readability score to measure language complexity, extracts the most salient keywords using YAKE, and—most powerfully—identifies the main themes of the presentation by classifying the text against predefined labels like "introduction" or "technical details".
- **Metrics Generated:**

  Readability Score: The Flesch Reading Ease score, where a higher score indicates simpler, more accessible language.

  Keywords: The most salient and frequently used terms in the speech.

  Main Themes: The conceptual topics of the speech, identified by a zero-shot classifier.

3.7 Worker: The Synthesis Agent

This is the final and most intelligent component, acting as the "brain" that brings all the analysis together.

- **Technology:** A fine-tuned version of Google's `gemma-2b-it` language model, specialized for our task using Parameter-Efficient Fine-Tuning (PEFT) with QLoRA.
- **Function:** This agent's sole purpose is to synthesize. It receives the complete, structured data packet containing the transcript and all metrics from the other workers. It then interprets this data and generates a single, coherent, human-readable feedback report, emulating the style of an expert communication coach. The fine-tuning process was a fascinating challenge, transforming a general-purpose model into a specialist capable of providing truly insightful, data-driven feedback.

4 Interaction Flow: A Step-by-Step Analysis

The agent's multi-component architecture operates as a sequential pipeline, managed entirely by the **Orchestrator**. This ensures a logical, traceable, and robust flow of data from the initial user request to the final analytical output. The entire end-to-end process is detailed below.

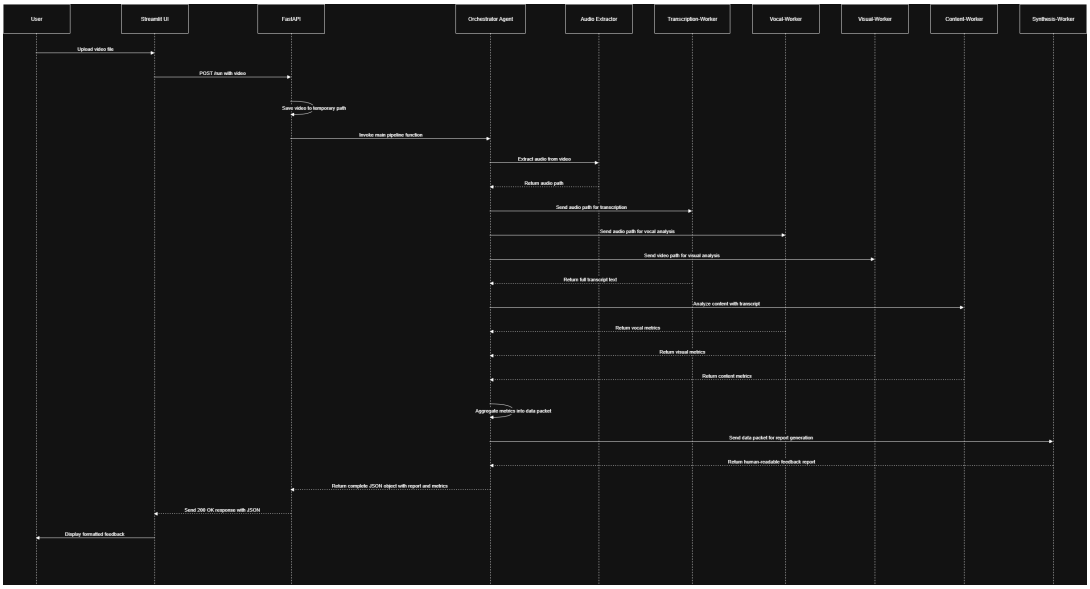

Figure 2: Sequence diagram illustrating the interaction between the UI, API, Orchestrator, and Worker agents.

1. **User Action:** The process begins when the user selects a video file and clicks the "Analyze Performance" button in the Streamlit user interface.
2. **API Request:** The UI sends the video file data via an HTTP POST request to the `/run` endpoint of our backend FastAPI server.
3. **Data Ingestion:** The backend server receives the file, saves it to a temporary location on the disk, and passes the file path to the **Orchestrator Agent**.
4. **Audio Extraction:** The Orchestrator's first action is to call the **Audio Extractor** utility. This utility uses `ffmpeg` to create a temporary, standardized WAV audio file from the video.
5. **Parallel Worker Execution:** The Orchestrator then initiates the data-gathering phase, calling the analytical workers.
    - The temporary WAV file path is passed to the **Transcription-Worker** and the **Vocal-Worker**.

- The original video file path is passed to the **Visual-Worker**.

6. **Dependent Worker Execution:** Once the **Transcription-Worker** completes its task and returns the full transcript text, the Orchestrator calls the **Content-Worker**, passing this transcript as its input.
7. **Data Aggregation:** The Orchestrator waits for all data-gathering workers to complete their analyses. It then collects their individual outputs (the transcript string, and the dictionaries for vocal, visual, and content metrics) into a single, comprehensive data packet.
8. **Final Synthesis:** This complete data packet is passed as input to the final agent in the chain: the fine-tuned **Synthesis-Worker**. This agent interprets the data and generates the final, human-readable feedback report.
9. **Cleanup:** The Orchestrator's `finally` block ensures that the temporary audio file created in Step 4 is deleted from the disk, regardless of whether the pipeline succeeded or failed.
10. **API Response:** The Orchestrator returns the final JSON object, containing the synthesized report and all the raw intermediate data, back to the API layer. FastAPI serializes this into a JSON response and sends it back to the UI.
11. **UI Presentation:** The Streamlit UI receives the JSON response, parses it, and displays the synthesized report and the detailed metrics in the interactive, tabbed interface.

This structured flow ensures that each component performs its dedicated task on the correct data and at the right time. The traceability of this process was a core design goal, making the system's behavior easy to debug, evaluate, and extend in the future.

5 Design Rationale & Conclusion

The architecture of the AI Presentation Coach was not arbitrary, it was the result of a series of deliberate design decisions aimed at creating a system that is robust, modular, and intelligent. My goal was to build not just a functional tool, but a superior one, and the following principles guided my work.

- **Modularity over Monolith:** The most fundamental decision was to adopt a multi-agent, Orchestrator-Workers architecture. As detailed in the research I consulted, this pattern provides immense benefits. It allowed me to develop, test, and refine each analytical capability in isolation. This separation of concerns was critical for managing the project's complexity and ensuring that each component is a best-in-class solution for its specific task.
- **Graceful Degradation for Robustness:** Real-world data is imperfect. A key design principle was that the agent must be resilient to low-quality video. This led to the implementation of the tiered analytical protocol in the **Visual-Worker**. By enabling the agent to fall back from high-fidelity iris tracking to a more robust head-pose estimation, the system guarantees that it will always provide a useful engagement metric, rather than failing completely. This is a crucial feature for a real-world application.
- **Specialization through Fine-Tuning:** While powerful, a generic large language model is not an expert. The decision to fine-tune a specialized **Synthesis-Agent** was a core part of the strategy to elevate the agent's output from simple data reporting to genuine, expert-level coaching. This dedication to building a specialized intelligence is, I believe, what transforms the prototype from a novelty into a genuinely useful tool.
- **Hybrid Tooling for Optimal Performance:** Rather than relying on a single, heavy model for all tasks, I chose a hybrid approach. For tasks like keyword extraction and readability, lightweight statistical libraries (`yake`, `textstat`) are more efficient. For complex tasks like thematic analysis and final synthesis, powerful deep learning models are necessary. This pragmatic approach of using the right tool for the right job ensures the agent is both powerful and performant.

In conclusion, the architecture of the AI Presentation Coach is a direct reflection of a modern, modular, and resilient approach to building agentic systems. Every component and interaction was designed with the end goal in mind: to create an empathetic, data-driven tool that can provide tangible value to anyone looking to improve their communication skills. I am proud of the resulting system and confident in its powerful and flexible foundation.