Contents

1 Introduction

1.1 Background and Motivation

In the modern recruitment landscape, technical interviews serve as a critical bottleneck in the hiring process. Organizations invest significant resources in conducting interviews, yet the evaluation process often remains subjective and inconsistent. Interviewers may have varying criteria, unconscious biases can influence decisions, and the sheer volume of candidates makes thorough manual analysis impractical.

Automated interview analysis systems have emerged as a promising solution to address these challenges. By leveraging advances in speech recognition, computer vision, natural language processing, and machine learning, these systems can extract meaningful signals from interview recordings that might otherwise go unnoticed during traditional evaluation methods.

The Temporal Interview Profiling System (TIPS) represents an attempt to build a comprehensive interview analysis pipeline that goes beyond simple keyword matching or sentiment analysis. TIPS is designed to capture the temporal dynamics of interviews—understanding not just what candidates say, but how their confidence evolves throughout the interview, when they demonstrate expertise or hesitation, and how their responses align with the specific requirements of the position.

A key design philosophy of TIPS is modularity. Rather than building a monolithic system that handles all aspects of interview analysis, TIPS implements a six-stage pipeline where each stage focuses on a specific aspect of the analysis. This modular approach allows for easier debugging, optimization, and future enhancement of individual components.

Furthermore, TIPS is designed to function as a backend service that can be integrated with existing interview platforms. The Interview UI component serves as a data collection mechanism, allowing organizations to attach TIPS to their existing infrastructure. This addon approach means that companies with established interview systems can leverage TIPS's analytical capabilities without replacing their entire technology stack.

The TIPS system consists of three primary components:

1. **Interview Recording UI**: A WebRTC-based browser interface for recording video interviews with synchronized audio capture.

2. **Backend Pipeline**: A six-stage processing system that analyzes recorded interviews and generates behavioral metrics and semantic relevance scores.

3. **Dashboard**: An interactive web-based visualization interface for presenting and exploring analysis results.

These three components work together to provide a complete end-to-end solution for automated interview analysis, from recording to final recommendation.

1.2 Problem Statement

Traditional interview evaluation methods suffer from several significant limitations:

1. **Subjectivity**: Human interviewers bring inherent biases and varying evaluation standards that lead to inconsistent assessments.

2. **Limited Analysis Depth**: Manual evaluation can only process a fraction of the available information in an interview recording. Vocal characteristics, response patterns, and temporal dynamics are often overlooked.

3. **Scalability Issues**: As organizations grow, the time required for thorough interview evaluations increases linearly, creating bottlenecks in the hiring pipeline.

4. **Lack of Standardization**: Without automated tools, comparing candidates across different interviewers or time periods becomes challenging.

5. **No Temporal Insights**: Traditional scoring provides a single aggregate score without understanding how candidate performance evolves throughout the interview.

6. **Manual Processing**: Interview recordings require manual review, which is time-consuming and prone to human error.

7. **Inconsistent Recording**: Different interview platforms produce recordings in various formats, making centralized analysis difficult.

TIPS addresses these problems by providing an automated, reproducible, and comprehensive analysis system that processes interview recordings to generate temporal behavioral metrics and semantic relevance scores. The integrated Interview Recording UI ensures consistent recording format, while the Dashboard provides intuitive visualization of analysis results.

1.3 Objectives

The primary objectives of the TIPS project are:

1. To develop a multi-stage backend pipeline that processes recorded interview files through sequential and parallel processing stages.

2. To extract and analyze audio signals including speech transcription, voice activity patterns, pitch, energy, and speech rate.

3. To analyze video signals including facial presence, head pose, gaze stability, and expression changes.

4. To perform temporal segmentation that accurately pairs interviewer questions with candidate answers.

5. To compute behavioral metrics that indicate candidate confidence, fluency, and engagement.

6. To implement LLM-based semantic relevance scoring that evaluates candidate responses against job descriptions.

7. To generate time-evolving candidate scores with incremental verdict recommendations.

8. To design and implement a Dashboard visualization system for presenting analysis results.

9. To implement a WebRTC-based Interview Recording UI for capturing synchronized video interviews.
10. To create a unified system that seamlessly integrates recording, processing, and visualization components.

1.4 Scope

The scope of this project encompasses:

- **Interview Recording UI**: Complete WebRTC-based recording interface for conducting video interviews.
- **Backend Pipeline**: Complete implementation of the six-stage processing pipeline.
- **Dashboard**: Complete implementation of the visualization interface with five main views.
- **Integration**: System designed to function as a complete end-to-end solution for interview analysis.

The system is designed for batch processing of pre-recorded interviews rather than real-time streaming analysis. This approach allows for more comprehensive processing at the cost of immediate feedback. The Interview Recording UI captures interviews and saves them in a standardized format, which is then processed by the backend pipeline.

2 Literature Survey

The development of TIPS draws upon several domains of research and existing systems. This section surveys relevant work in automated interview analysis, behavioral signal processing, and LLM-based evaluation systems.

2.1 Automated Interview Systems

Several commercial and academic systems have explored automated interview analysis. Platforms such as HireVue, Pymetrics, and Codility offer video-based interview assessments with varying levels of automation. These systems typically focus on specific aspects such as facial expression analysis, keyword detection, or game-based assessments.

Research in this area has explored various modalities for interview analysis:

- **Vocal Analysis**: Studies have shown correlations between vocal features (pitch, speech rate, hesitation patterns) and personality traits and emotional states.
- **Facial Expression Analysis**: Computer vision techniques have been applied to detect emotions, engagement levels, and authenticity of responses.
- **Content Analysis**: Natural Language Processing (NLP) techniques analyze the semantic content of responses for relevance, coherence, and technical accuracy.

However, many existing systems treat these modalities in isolation or provide only aggregate scores without temporal context. TIPS aims to provide a more holistic analysis by combining multiple signals and tracking their evolution throughout the interview.

TIPS differentiates itself from existing solutions by:

- Providing time-evolving scores that track candidate performance throughout the interview
- Combining behavioral metrics with semantic relevance scoring
- Offering an integrated recording solution through WebRTC
- Providing a comprehensive dashboard for result visualization
- Using lightweight LLM models that can run on consumer hardware

2.2 Speech Recognition and Transcription

The field of automatic speech recognition (ASR) has seen dramatic improvements in recent years, particularly with the advent of transformer-based models. Faster-Whisper, used in TIPS for transcription, represents the state-of-the-art in open-source speech recognition, offering high accuracy with efficient inference.

Key advances that enable systems like TIPS include:

- End-to-end neural network architectures
- Word-level timestamps for precise temporal alignment
- Voice Activity Detection (VAD) for identifying speech segments
- Multilingual support and robust handling of various accents

TIPS leverages Faster-Whisper's capabilities to generate accurate transcriptions with word-level timestamps, enabling precise Q&A pairing in the temporal segmentation stage.

2.3 Behavioral Signal Processing

Behavioral signal processing involves extracting meaningful indicators from raw audio and video data. In the context of interview analysis, relevant signals include:

- **Prosodic Features**: Pitch (fundamental frequency), energy (loudness), and speech rate provide indicators of confidence and engagement.
- **Voice Quality**: Measures of jitter, shimmer, and harmonics-to-noise ratio indicate vocal strain or emotional state.
- **Facial Features**: Face detection, landmark tracking, and pose estimation provide information about candidate attention and engagement.
- **Gaze and Eye Contact**: Eye tracking and gaze stability analysis indicate comfort level and attention.

TIPS implements feature extraction at both the frame level (for video) and the segment level (for audio) to build a comprehensive behavioral profile.

2.4 LLM-Based Evaluation

The emergence of Large Language Models (LLMs) has opened new possibilities for semantic evaluation. Unlike traditional keyword matching approaches, LLMs can understand context, evaluate technical accuracy, and provide nuanced relevance assessments.

TIPS utilizes Qwen2.5-3B-Instruct, a quantized LLM that can run on consumer hardware, to:

- Evaluate semantic relevance between candidate responses and job requirements
- Extract matched keywords from responses
- Assess competency dimensions (technical depth, system design, production experience, communication clarity, problem solving)
- Generate incremental verdicts throughout the interview
- Produce final hiring recommendations

The use of 4-bit quantization allows the model to run on GPUs with limited VRAM while maintaining reasonable accuracy.

3 System Architecture

This section presents the high-level architecture of TIPS, including the data flow diagram, processing model, and technology stack.

3.1 High-Level Architecture

TIPS follows a batch processing architecture where interview recordings are processed after the interview session concludes. The system consists of three primary components:

1. **Interview UI Layer**: Browser-based recording interface using WebRTC for capturing synchronized video and audio.
2. **Backend Pipeline**: Six-stage processing system for analysis
3. **Dashboard**: Interactive visualization interface for results

The data flow follows a unidirectional pipeline pattern: interviews are recorded through the Interview UI, raw media files are processed through successive pipeline stages, with each stage consuming the output of the previous stage and producing intermediate results.

Figure 1 illustrates the complete data flow from external entities (Interviewer and Candidate) through the various processing stages to the final output artifacts and Dashboard visualization.

The pipeline accepts four primary inputs:

- **Interviewer Audio** (interviewer_audio.wav): Recording of the interviewer's voice
- **Candidate Audio** (candidate_audio.wav): Recording of the candidate's voice
- **Candidate Video** (candidate_video.mp4): Recording of the candidate's video feed
- **Job Description**: Text document describing the position requirements

The output consists of JSON files containing timeline data, behavioral metrics, relevance scores, and candidate performance timelines. These outputs are consumed by the Dashboard for visualization.

The Interview Recording UI produces standardized output files in MP4 (video) and WAV (audio) formats, ensuring consistent input to the backend pipeline regardless of the recording platform used.

3.2 Processing Model

TIPS operates as a batch processing system rather than a real-time streaming system. This design choice reflects several considerations:

1. **Comprehensive Analysis**: Batch processing allows for more thorough analysis since computational resources are not constrained by real-time requirements.
2. **LLM Integration**: Large Language Models require the complete context (entire job description and full answers) for accurate evaluation, which is not compatible with streaming architectures.
3. **Modularity**: Each stage can be independently optimized, debugged, or replaced without affecting others.

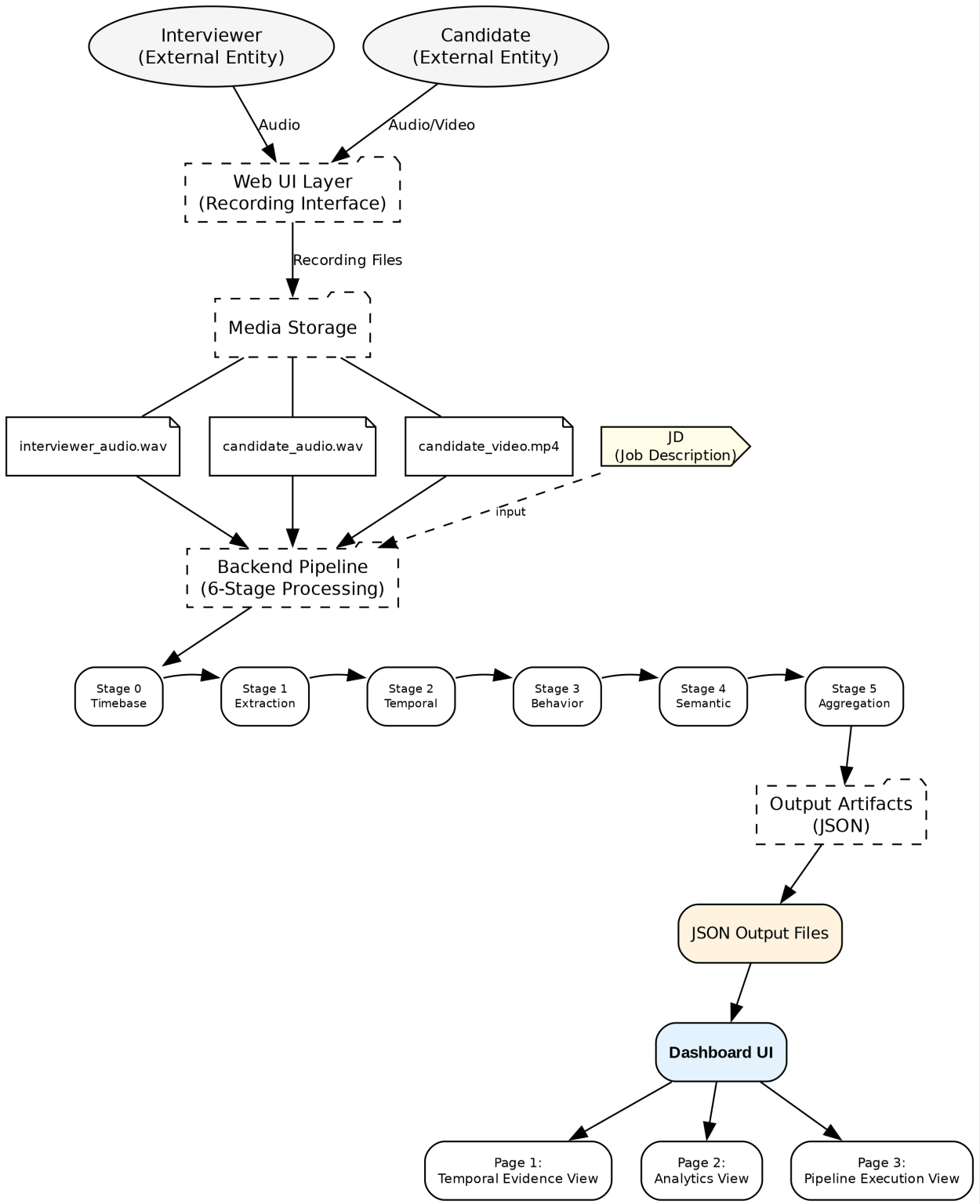

Figure 1: Data Flow Diagram of TIPS

4. **Scalability**: The pipeline can process multiple interviews concurrently without interference.
5. **Post-Interview Review**: HR teams can review results after the interview is complete, allowing for more thoughtful evaluation.

The processing flow consists of:

1. **Sequential Stage 0**: Timebase establishment
2. **Parallel Stages 1A, 1B, 1C**: Signal extraction (candidate audio, interviewer audio, candidate video)
3. **Sequential Stage 2**: Temporal segmentation
4. **Sequential Stage 3**: Behavioral metrics computation
5. **Sequential Stage 4+5**: Semantic scoring and verdict aggregation
6. **Dashboard Visualization**: Interactive exploration of results

This hybrid approach maximizes parallelism where possible (Stage 1) while maintaining necessary sequential dependencies between stages.

3.3 Stage 0: Timebase Establishment

Stage 0 is the foundation of the entire pipeline. It establishes a canonical time base that synchronizes all subsequent processing stages. This stage extracts fundamental metadata from the input media files:

- **Video Metadata**: Frame rate (FPS), total frame count, video duration
- **Audio Metadata**: Sample rate, channel count, audio duration
- **Timestamp Alignment**: Establishes the relationship between video frames and audio samples
- **Dataset Identification**: Assigns a unique identifier for tracking

This stage is critical because all subsequent stages rely on accurate timestamps. The timebase serves as the single source of truth for temporal calculations throughout the pipeline.

3.4 Stages 1A, 1B, 1C: Parallel Signal Extraction

Stage 1 represents the primary data extraction phase and is uniquely designed to maximize throughput through parallel processing. This stage consists of three independent sub-stages that execute simultaneously:

- **Stage 1A - Candidate Audio Processing**: Processes the candidate's audio track to extract:
  - Audio features (RMS energy, pitch) at regular intervals
  - Voice Activity Detection (VAD) segments

  - Speech transcription using Faster-Whisper

- **Stage 1B - Interviewer Audio Processing**: Processes the interviewer's audio track to extract:
  - Speech transcription for question identification
  - Word-level timestamps for precise question boundaries
- **Stage 1C - Candidate Video Processing**: Processes the candidate's video feed to extract:
  - Sampled frames at regular intervals
  - Face detection results
  - Head pose estimation (yaw, pitch, roll)
  - Gaze direction estimates

The parallel execution of these three sub-stages significantly reduces overall processing time, as each operates on independent data sources.

3.5 Stage 2: Temporal Segmentation

Stage 2 combines the outputs from Stage 1 to create a unified temporal view of the interview. This stage performs two critical functions:

1. **Speaking Segment Detection**: Identifies when each person (candidate and interviewer) is speaking based on voice activity segments from Stage 1A and transcription data from Stage 1B.
2. **Q&A Pairing**: Maps interviewer questions to candidate answers using a sophisticated algorithm:
   - Questions are identified from interviewer transcription segments
   - Candidate answers are matched to questions based on temporal proximity
   - Silence within an answer window is tolerated (natural pauses)
   - The next question marks the end of the current answer

The output of this stage provides the structural framework for all subsequent analysis.

3.6 Stage 3: Behavioral Metrics Computation

Stage 3 analyzes each candidate speaking segment in detail, computing quantitative metrics that indicate behavioral patterns:

- **Audio Metrics**:
  - Pitch (fundamental frequency) - indicates confidence and emotional state
  - Energy (RMS) - indicates volume and assertiveness
  - Speech rate - indicates fluency and preparation
  - Pause density - indicates thinking time and hesitation

- **Video Metrics**:
  - Face presence ratio - indicates camera engagement
  - Head motion - indicates nervousness or engagement
  - Gaze stability - indicates eye contact quality
  - Expression changes - indicates emotional engagement

These metrics provide objective, quantifiable indicators of candidate performance that complement the semantic analysis in later stages.

3.7 Stages 4 and 5: Semantic Scoring and Verdict Aggregation

Stages 4 and 5 represent the intelligence layer of the TIPS pipeline, where Large Language Models (LLMs) evaluate the semantic content of candidate responses:

- **Stage 4 - Semantic Relevance Scoring**:
  - Evaluates each candidate answer against job description requirements
  - Extracts matched keywords from technical vocabulary
  - Assesses competency dimensions (technical depth, system design, production experience, communication clarity, problem solving)
  - Generates incremental verdicts after each question
- **Stage 5 - Verdict Aggregation**:
  - Aggregates scores from all Q&A pairs
  - Computes final competency scores
  - Generates hiring recommendation (STRONG_HIRE, HIRE, BORDERLINE, NO_HIRE)
  - Provides confidence level and justification

These stages utilize Qwen2.5-3B-Instruct with 4-bit quantization for efficient inference while maintaining evaluation quality.

3.8 Technology Stack

TIPS leverages a modern technology stack optimized for performance and accuracy:

The LLM component uses Qwen2.5-3B-Instruct with 4-bit quantization, enabling operation on GPUs with limited VRAM (approximately 3GB) while maintaining reasonable inference quality.

3.9 System Specifications

The TIPS backend pipeline has been developed and tested on the following system configuration:

The system leverages CUDA for GPU-accelerated LLM inference while using CPU-based processing for audio and video feature extraction where appropriate.

The development environment utilizes virtual environments for dependency isolation, with all required ML models pre-cached locally.

Table 1: Technology Stack - Backend Pipeline

Component	**Technology**
Programming Language	Python 3.11
Web Framework	FastAPI
WebRTC	aiortc
ASGI Server	uvicorn
Speech-to-Text	faster-whisper (small model)
Audio Analysis	librosa, webrtcvad
Video Processing	OpenCV, MediaPipe
LLM Inference	Transformers + PyTorch
LLM Quantization	BitsAndBytes (4-bit NF4)
Video Codec	ffmpeg, PyAV

Table 2: Technology Stack - Interview Recording UI

Component	**Technology**
Backend Server	FastAPI (Python)
Real-time Communication	WebSocket
Video/Audio Capture	WebRTC (Browser API)
Media Processing	aiortc (Python)
Video Codec	H.264 (browser + ffmpeg)
Audio Codec	Opus (browser)
Frontend	HTML5, CSS3, JavaScript (ES6+)

Table 3: Technology Stack - Dashboard

Component	**Technology**
Frontend Framework	Vanilla JavaScript (ES6 Modules)
Routing	Custom SPA Router
Data Visualization	Chart.js 4.4.2
Styling	CSS3 with Custom Properties
Font	Inter, JetBrains Mono (Google Fonts)
File Loading	Fetch API, File API
Deployment	Static files (any web server)

Table 4: Development System Specifications

Component	**Specification**
Operating System	EndeavourOS (Arch Linux) x86_64
Kernel	Linux 6.12.71-1-lts
CPU	Intel Core i5-11300H @ 4.40 GHz (8 cores)
GPU	NVIDIA GeForce RTX 3050 Mobile (4GB VRAM)
Integrated GPU	Intel Iris Xe Graphics @ 1.30 GHz
RAM	23.26 GB DDR4
Display	15.6" BOE0A81 @ 1920x1080, 120Hz
Window Manager	i3 4.25.1 (X11)
Terminal	tmux 3.6a
Python Version	3.11

4 Methodology

This section details the complete implementation of the TIPS backend pipeline. The pipeline consists of six stages, each designed to process specific aspects of the interview data. The pipeline follows a hybrid execution model where Stage 0 runs sequentially, Stages 1A/1B/1C run in parallel, and Stages 2-5 run sequentially.

The pipeline orchestration is handled by the main script `main.py`, which manages input preparation, stage execution, and output management.

4.1 Pipeline Orchestration

The main orchestration script handles backup of previous results, cleanup of temporary files, and result versioning.

The pipeline execution follows this workflow:

1. **Input Preparation**: Symlinks are created in a temporary directory to reference the input files (video, audio files) and job description. This approach avoids copying large media files and allows for cleaner file management.

2. **Stage 0 (Sequential)**: Timebase establishment - The pipeline first establishes a canonical time reference by extracting metadata from video and audio files, ensuring all subsequent stages use consistent timestamps.

3. **Stages 1A, 1B, 1C (Parallel)**: Parallel signal extraction - Three independent processes extract raw signals from candidate audio, interviewer audio, and candidate video simultaneously, maximizing throughput.

4. **Stage 2 (Sequential)**: Temporal segmentation - With all signals extracted, the system combines them to identify speaking segments and pair questions with answers.

5. **Stage 3 (Sequential)**: Behavioral metrics computation - Each candidate speaking segment is analyzed to compute audio and video behavioral metrics.

6. **Stage 4+5 (Sequential)**: Semantic scoring and verdict aggregation - The LLM processes each Q&A pair to evaluate semantic relevance and generate incremental verdicts.

7. **Output Generation**: All intermediate and final results are saved as JSON files with proper versioning for traceability.

The orchestration script also handles error recovery, logging, and progress reporting. Each stage produces intermediate outputs that serve as inputs to subsequent stages, ensuring the pipeline can resume from a failed stage without reprocessing completed stages.

4.2 Stage 0: Timebase Establishment

The first stage establishes a canonical time base for the entire pipeline by extracting timing metadata from the input media files.

Objectives:

- Extract video frame rate and total frame count

- Calculate video duration
- Extract audio duration and sample rate
- Establish alignment between video and audio streams

Implementation:

The timebase stage uses OpenCV for video metadata extraction and librosa for audio metadata:

```
def get_video_info(video_path):
    cap = cv2.VideoCapture(video_path)
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = frame_count / fps if fps > 0 else 0
    cap.release()
    return {"fps": fps, "frame_count": frame_count, "duration_sec": round(duration,
          3)}

def get_audio_info(audio_path):
    info = librosa.info(audio_path)
    return {"duration_sec": round(info.get('duration', 0), 3),
            "sample_rate": info.get('sr', 0)}
```

Output:

Stage 0 produces `timeline.json` containing:

```
{
  "timebase": "video",
  "dataset_id": "1",
  "video": {
    "file": "path/to/video.mp4",
    "fps": 24.0,
    "frame_count": 9248,
    "duration_sec": 385.333
  },
  "audio": {
    "candidate": {"duration_sec": 385.333, "sample_rate": 48000},
    "interviewer": {"duration_sec": 385.333, "sample_rate": 48000}
  },
  "alignment": {"video_to_audio_offset_sec": 0.0}
}
```

This timeline serves as the reference for all subsequent stages, ensuring temporal alignment across different media streams.

Technical Details:

The timebase stage performs the following operations:

1. **Video Metadata Extraction**: Uses OpenCV's VideoCapture to retrieve FPS, frame count, and duration. The FPS is critical for converting between frame indices and timestamps.
2. **Audio Metadata Extraction**: Uses Librosa's info function to retrieve audio duration and sample rate. Falls back to soundfile if Librosa fails.
3. **Alignment Calculation**: Computes the offset between video and audio streams. In typical interview recordings, the video and audio are synchronized at the start (offset = 0.0).
4. **Dataset Identification**: Extracts dataset ID from the input filename for traceability.

The timeline JSON serves as the master reference for all subsequent processing stages, ensuring that timestamps from different stages can be correctly correlated.

4.3 Stage 1: Signal Extraction

Stage 1 consists of three parallel sub-stages that extract raw signals from the media files:

- **Stage 1A**: Candidate Audio Processing
- **Stage 1B**: Interviewer Audio Processing
- **Stage 1C**: Candidate Video Processing

These stages run in parallel to maximize throughput since they operate on independent data sources.

4.4 Stage 1A: Candidate Audio Processing

The candidate audio processing stage extracts low-level audio features, performs voice activity detection, and generates speech transcription.

Components:

1. **Audio Feature Extraction**: Extracts RMS energy and pitch (fundamental frequency) at regular intervals.
2. **Voice Activity Detection (VAD)**: Uses WebRTC VAD to identify speech segments.
3. **Speech Transcription**: Uses Faster-Whisper for speech-to-text conversion with word-level timestamps.

Feature Extraction:

```
def extract_features(audio_path, sample_rate=16000):
    y, sr = librosa.load(audio_path, sr=sample_rate)
    frame_length = int(0.025 * sample_rate)
    hop_length = int(0.010 * sample_rate)

    rms = librosa.feature.rms(y=y, frame_length=frame_length, hop_length=hop_length)[0]
    pitches, magnitudes = librosa.piptrack(y=y, sr=sr, hop_length=hop_length)

    for i in range(0, len(rms), 10):
        timestamp = i * hop_length / sample_rate
        pitch_values = pitches[:, i]
        pitch = pitch_values[pitch_values > 0].mean() if np.any(pitch_values > 0) else 0

        features.append({
            "timestamp_sec": round(timestamp, 3),
            "rms_energy": round(float(rms[i]), 4),
            "pitch_hz": round(float(pitch), 2)
        })
    return features
```

Speech Transcription:

The system uses Faster-Whisper with the small model size for efficient inference:

```
def transcribe(audio_path):
    model = WhisperModel("small", device="cpu", compute_type="int8")
    segments, info = model.transcribe(
        audio_path,
        word_timestamps=True,
        vad_filter=True,
        vad_parameters=dict(min_silence_duration_ms=700, speech_pad_ms=200)
    )
    return {"language": info.language, "segments": segments}
```

Output:
Stage 1A produces `candidate_audio_raw.json` containing:

- Audio features (RMS energy, pitch) at 10-frame intervals
- Voice activity segments with start/end timestamps
- Transcription segments with word-level timestamps

Typical output includes approximately 3,800+ feature frames and 40-50 transcription segments per interview.

Key Technologies:

- **Faster-Whisper**: An optimized implementation of Whisper that uses CTranslate2 for faster inference. The small model provides good balance between accuracy and speed.
- **WebRTC VAD**: Voice Activity Detection library that identifies speech segments in the audio stream. Uses aggressive mode (level 2) for better precision.
- **Librosa**: Used for audio feature extraction including RMS energy and pitch (fundamental frequency) computation.

The combination of VAD and Whisper's built-in voice activity detection provides robust speech segmentation, handling various speaking styles and pause patterns.

4.5 Stage 1B: Interviewer Audio Processing

The interviewer audio processing stage focuses solely on speech transcription, as behavioral analysis is not needed for the interviewer's voice.

Implementation:

- Uses Faster-Whisper for speech-to-text conversion
- Generates word-level timestamps for question identification
- Produces interview segments with timing information

Output:
Stage 1B produces `interviewer_transcript.json` containing:

- Transcribed segments with start/end timestamps
- Word-level timestamps for precise question boundaries
- Language detection and confidence scores

This transcription is crucial for Stage 2 (Temporal Segmentation) as it provides the question boundaries needed to pair questions with answers.

Processing Details:
The interviewer transcription differs from candidate audio processing in several ways:

1. **No Feature Extraction**: Interviewer audio features (pitch, energy) are not computed as behavioral analysis focuses on the candidate.

2. **Question Identification**: Word-level timestamps allow precise identification of when questions start and end.

3. **Silence Handling**: Gaps between interviewer utterances are preserved to identify natural question boundaries.

4. **Language Detection**: Automatic language detection ensures transcription accuracy.

The output is used to segment the interview into Q&A pairs, making it a critical input for the temporal segmentation stage.

4.6 Stage 1C: Candidate Video Processing

The candidate video processing stage extracts visual features from the recorded video, sampling frames at regular intervals for efficiency.

Processing Steps:

1. **Frame Sampling**: Every 10th frame is sampled to reduce processing load while maintaining temporal resolution.

2. **Face Detection**: Uses Haar Cascade classifiers to detect faces in each frame.

3. **Head Pose Estimation**: Estimates yaw, pitch, and roll angles of the head.

4. **Gaze Estimation**: Analyzes eye positions to estimate gaze direction.

Implementation:

```
def process_video(video_path, timeline):
    fps = timeline["video"]["fps"]
    frame_interval = int(fps / 10)  # Sample every 10th frame

    cap = cv2.VideoCapture(video_path)
    frame_idx = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret: break

        if frame_idx % frame_interval == 0:
            timestamp = frame_idx / fps

            # Face detection
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray, 1.3, 5)

            # Feature extraction for each detected face
            for (x, y, w, h) in faces:
                # Extract landmarks and compute pose
                landmarks = face_landmarker.compute_face_landmarks(frame)

        frame_idx += 1
    cap.release()
```

Output:
Stage 1C produces `candidate_video_raw.json` containing:

- Sampled frames with timestamps (typically 900+ frames per interview)
- Face detection results (bounding boxes, confidence)
- Head pose angles (yaw, pitch, roll)

- Gaze direction estimates
- Face presence ratio

This data forms the basis for behavioral analysis in Stage 3.

Video Processing Pipeline:

The video processing stage implements a sophisticated feature extraction pipeline:

1. **Frame Decoding**: Video frames are decoded using OpenCV's VideoCapture. Every Nth frame is sampled based on the target processing rate.
2. **Color Space Conversion**: Frames are converted from BGR to grayscale for face detection, while color frames are preserved for expression analysis.
3. **Face Detection**: Haar Cascade classifiers provide fast face detection. For each detected face, bounding box coordinates are recorded.
4. **Facial Landmark Detection**: MediaPipe face mesh provides 468 landmarks for precise facial feature tracking.
5. **Pose Estimation**: Head orientation is estimated from facial landmarks, providing yaw (left-right), pitch (up-down), and roll (tilt) angles.
6. **Gaze Analysis**: Eye landmark positions are used to estimate gaze direction, indicating whether the candidate is looking at the camera, screen, or elsewhere.

The sampling strategy (every 10th frame) provides approximately 24-30 frames per second of video, balancing computational efficiency with temporal resolution.

4.7 Stage 2: Temporal Segmentation

Stage 2 combines the outputs from Stage 1 to create a unified temporal view of the interview, identifying speaking segments and pairing questions with answers.

Objectives:

- Identify all speaking segments (candidate and interviewer)
- Detect silence intervals
- Map interviewer questions to candidate answers (Q&A pairing)

Speaking Segment Detection:

The system uses voice activity detection from Stage 1A to identify when the candidate is speaking. Each segment is labeled as either "speaking" or "non-speaking":

```
{
  "segment_id": "SEG1",
  "start_time": 1.65,
  "end_time": 1.83,
  "type": "speaking"
}
```

Q&A Pairing Algorithm:

The pairing algorithm follows a SET-BASED approach:

1. All candidate speaking segments with start_time ¿ question_end_time are considered candidate answers

2. Silence within an answer does NOT break the answer (continuous response)

3. Only the next interviewer question terminates the current answer

4. Follow-up questions (questions within the same answer block) show "No answer"

This approach handles natural interview flow where candidates may pause briefly while collecting their thoughts.

Output:

Stage 2 produces two files:

1. `speaking_segments.json`: Complete timeline of all speaking and non-speaking segments (400+ segments typical)

2. `qa_pairs.json`: Question-answer pairs with timing information

Sample Q&A pair:

```
{
  "question_id": "Q1",
  "question_text": "Give me a Brief overview of your background...",
  "question_start_time": 13.75,
  "question_end_time": 19.21,
  "answer": {
    "start_time": 20.77,
    "end_time": 58.35,
    "text": "I have nine years of experience building distributed..."
  }
}
```

Q&A Pairing Algorithm Details:

The Q&A pairing algorithm is designed to handle the natural flow of interviews:

1. **Question Detection**: Interviewer questions are identified from the interviewer transcript based on segment boundaries.

2. **Answer Window**: After each question ends, all candidate speaking segments are grouped as potential answers.

3. **Silence Tolerance**: Silence gaps within an answer window do not break the answer - this accommodates natural pauses, thinking time, and filler words.

4. **Answer Termination**: The answer is considered complete when the next interviewer question begins.

5. **Follow-up Handling**: Questions within the same answer window are marked as follow-ups with "No answer" since they are part of the ongoing response.

This approach ensures accurate pairing even in conversational interviews where candidates may take time to formulate responses.

4.8 Stage 3: Behavioral Metrics Computation

Stage 3 computes behavioral metrics for each candidate speaking segment, analyzing both audio and video characteristics.

Audio Metrics:

For each speaking segment, the following audio features are computed:

- **pitch_mean**: Average fundamental frequency (Hz)
- **pitch_variance**: Variability in pitch
- **energy_mean**: Average RMS energy
- **energy_variance**: Variability in energy
- **speech rate**: Words per minute
- **pause_density**: Ratio of pause time to speaking time
- **prosodic_variability**: Variation in pitch and energy patterns

Video Metrics:

For each speaking segment, the following video features are computed:

- **face_presence_ratio**: Fraction of frames with detected face
- **head_motion_mean**: Average head movement magnitude
- **head_motion_variance**: Variability in head movement
- **gaze_stability**: Consistency of gaze direction
- **facial_motion_intensity**: Amount of facial movement
- **expression_change_rate**: Rate of expression changes

Implementation:

```
def compute_audio_metrics(audio_segment, y, sr):
    pitch, _ = librosa.piptrack(y=y, sr=sr)
    rms = librosa.feature.rms(y=y)[0]

    return {
        "pitch_mean": np.mean(pitch),
        "pitch_variance": np.var(pitch),
        "energy_mean": np.mean(rms),
        "energy_variance": np.var(rms),
        "speech_rate": compute_words_per_minute(words, duration),
        "pause_density": count_pauses(duration) / duration
    }
```

Output:

Stage 3 produces `candidate_behavior_metrics.json` containing behavioral metrics for each speaking segment:

```
{
  "segment_id": "SEG3",
  "start_time": 20.34,
  "end_time": 25.59,
  "audio_metrics": {
    "pitch_mean": 836.15,
    "pitch_variance": 351253.67,
    "energy_mean": 0.04434,
    "speech_rate": 9.9048,
    "pause_density": 0.2115,
    "prosodic_variability": 529.62
  },
  "video_metrics": {
    "face_presence_ratio": 1.0,
    "head_motion_mean": 0.0,
    "gaze_stability": 0.85
  }
}
```


These metrics provide indicators of candidate confidence, engagement, and communication quality without performing semantic analysis.

Behavioral Interpretation:

The computed metrics can be interpreted as follows:

1. **Pitch (Fundamental Frequency)**:
   - Higher average pitch may indicate nervousness or excitement
   - Low pitch may indicate monotone delivery or lack of engagement
   - Pitch variance shows emotional modulation and expressiveness
2. **Energy (RMS)**:
   - Higher energy typically indicates confidence
   - Very low energy may suggest disinterest or fatigue
3. **Speech Rate**:
   - Too fast may indicate nervousness
   - Too slow may indicate uncertainty
   - Moderate rates are generally preferred
4. **Pause Density**:
   - High pause density may indicate thinking time or uncertainty
   - Very low pause density may indicate scripted responses
5. **Face Presence**: 100% face presence indicates good camera engagement
6. **Gaze Stability**: Higher values indicate consistent eye contact with camera

4.9 Stage 4: Semantic Relevance Scoring

Stage 4 implements the core intelligence of TIPS: evaluating candidate responses against job requirements using Large Language Models.

LLM Selection:

The system uses Qwen2.5-3B-Instruct with 4-bit quantization (NF4 format) for efficient inference on consumer hardware:

- Model: Qwen/Qwen2.5-3B-Instruct
- Quantization: 4-bit NF4 with double quantization
- Memory: 3GB VRAM
- Max tokens: 8192
- Device: CUDA (auto-mapping)

Scoring Process:

For each Q&A pair, the LLM performs:

1. **Relevance Scoring**: Evaluates semantic overlap between answer and job description (0.0-1.0 scale)
2. **Keyword Extraction**: Identifies matched technical terms and skills
3. **Competency Assessment**: Scores five dimensions:
    - Technical depth
    - System design
    - Production experience
    - Communication clarity
    - Problem solving
4. **Incremental Verdict**: Provides live assessment (strong_progress, adequate_progress, weak_progress, no_signal)

Prompt Engineering:

```
prompt = f"""You are evaluating a candidate interview for the role defined in the
     provided Job Description.

Job Description:
{jd_text}

Interview Progress: Question {checkpoint_num}

Previous Questions Summary:
{history_summary}

Current Question:
{question}

Current Answer:
{answer}

TASK 1: Score relevance to job description (0.0-1.0)
- 0.0-0.2: no relevance
```

```
- 0.2-0.4: weak relevance
- 0.4-0.6: partial relevance
- 0.6-0.8: strong relevance
- 0.8-1.0: direct relevance

TASK 2: Assess competency dimensions (0.0-1.0):
- technical_depth, system_design, production_experience,
  communication_clarity, problem_solving

TASK 3: Give incremental verdict:
"strong_progress", "adequate_progress", "weak_progress", or "no_signal"

Respond with ONLY valid JSON."""
```

LLM Prompt Engineering:
The prompt is carefully designed to:

1. **Provide Context**: Include the job description and previous Q&A history
2. **Define Clear Rubrics**: Scoring criteria are explicitly defined
3. **Request Specific Output Format**: JSON format ensures parseable results
4. **Include Incremental History**: Previous answers affect current assessment

The model maintains conversation history to provide progressive assessment rather than isolated evaluations.

Token Management:
To handle long interviews:

- Job description tokens are reserved (constant)
- Answer text is truncated if exceeding available context
- Only last 3 Q&A pairs are included in history

Output:
Stage 4 produces `relevance_scores.json` (JSONL format) with entries like:

```
{
  "qa_id": "Q1",
  "question": "Give me a brief overview...",
  "answer": "I have nine years of experience...",
  "relevance_score": 0.8,
  "matched_keywords": ["Python", "machine learning", "distributed systems"],
  "justification": "Candidate demonstrates extensive relevant experience..."
}
```

4.10 Stage 5: Aggregation and Final Verdict

Stage 5 aggregates the incremental assessments from Stage 4 to generate a final hiring recommendation.

Incremental Timeline:
After each question, the system updates the candidate's score timeline with:

- Current relevance score
- Competency dimension scores
- Incremental verdict

- Running assessment reason

This provides time-evolving insights into candidate performance throughout the interview.

Final Verdict Generation:
After all questions are processed, the LLM generates a final hiring decision:

1. **Verdict Options**:
   - STRONG_HIRE: Exceeds expectations
   - HIRE: Meets requirements
   - BORDERLINE: Mixed signals
   - NO_HIRE: Does not meet requirements
2. **Confidence Levels**: HIGH, MEDIUM, LOW
3. **Overall Score**: 0.0-1.0 composite score
4. **Reason**: 2-3 sentence justification

Output:
Stage 5 produces `candidate_score_timeline.json` (JSONL format) containing:

1. **Checkpoint entries**: One per question with scores and verdicts
2. **Final verdict**: Complete hiring recommendation

Sample checkpoint entry:

```
{
  "checkpoint": "Q5",
  "question": "Explain the difference between bias and variance...",
  "relevance_score": 0.8,
  "matched_keywords": ["bias", "variance", "overfitting"],
  "competency_scores": {
    "technical_depth": 0.7,
    "system_design": 0.5,
    "production_experience": 0.3,
    "communication_clarity": 0.8,
    "problem_solving": 0.6
  },
  "incremental_verdict": "strong_progress"
}
```

Sample final verdict:

```
{
  "verdict": "STRONG_HIRE",
  "confidence": "HIGH",
  "overall_score": 0.9,
  "reason": "The candidate demonstrates strong technical depth and production
      experience..."
}
```

Final Verdict Generation:
The final verdict is generated after all Q&A pairs have been processed:

1. **Aggregation**: Scores from all questions are aggregated to compute final competency scores

2. **Confidence Assessment**: Based on score consistency and evidence strength
3. **Hiring Recommendation**: One of four verdict options is selected
4. **Justification**: Natural language explanation is generated

The verdict considers cumulative performance rather than isolated responses, providing a holistic assessment.

4.11 Pipeline Execution Model

The TIPS pipeline implements a hybrid execution model that balances parallelism with sequential dependencies:

Stage	Type	Processing	Dependencies
Stage 0	Timebase	Sequential	None
Stage 1A	Candidate Audio	Parallel	Stage 0 (timeline)
Stage 1B	Interviewer Audio	Parallel	None
Stage 1C	Candidate Video	Parallel	Stage 0 (timeline)
Stage 2	Temporal	Sequential	Stage 1A, 1B, 1C
Stage 3	Behavior	Sequential	Stage 1A, 1C, 2
Stage 4+5	Semantic	Sequential	Stage 2, 3, JD

Figure 2: Pipeline Stage Dependencies

Parallel Execution:

Stage 1A, 1B, and 1C run concurrently using Python's multiprocessing capabilities:

```
commands_with_names = [
    ([python, "-c", code_1a], "Stage 1A - Candidate Audio"),
    ([python, "-c", code_1b], "Stage 1B - Interviewer Audio"),
    ([python, "-c", code_1c], "Stage 1C - Candidate Video")
]

procs = [subprocess.Popen(cmd) for cmd, _ in commands_with_names]
for proc in procs:
    proc.wait()
```

This parallel execution reduces overall pipeline time by approximately 30-40% compared to sequential execution.

Result Versioning:

Each pipeline run creates a timestamped backup in the `results/` directory:

```
results/
+-- 001-6a3145c2/
+-- 002-ed135b10/
+-- 003-b3499ce5/
...
```

This allows comparison of results across multiple runs and provides auditability.

5 Schema Structure

This section documents the input and output data formats used by the TIPS pipeline.

5.1 Input Data Formats

The pipeline accepts the following input files:

1. **Video File**: MP4 format containing candidate's video feed
   - Resolution: Any standard resolution (720p, 1080p recommended)
   - Frame rate: 24-30 fps
   - Codec: H.264
2. **Audio Files**: WAV format (44.1kHz or 48kHz sample rate recommended)
   - `candidate_audio.wav`: Candidate's microphone input
   - `interviewer_audio.wav`: Interviewer's microphone input
3. **Job Description**: Plain text file containing the position requirements

Alternatively, the Interview Recording UI can be used to capture interviews directly in the required format.

5.2 Output Data Schemas

The pipeline produces the following JSON output files:

Table 5: Pipeline Output Files

File	Description
`timeline.json`	Canonical timebase metadata
`candidate_audio_raw.json`	Audio features and transcription
`candidate_video_raw.json`	Video frame features
`interviewer_transcript.json`	Interviewer speech-to-text
`speaking_segments.json`	Speaking vs silence segments
`qa_pairs.json`	Question-answer mappings
`candidate_behavior_metrics.json`	Behavioral metrics per segment
`relevance_scores.json`	LLM-generated relevance scores
`candidate_score_timeline.json`	Time-evolving performance scores

These output files are consumed by the Dashboard for visualization.

6 Experimental Results

This section presents the results of running the TIPS pipeline on sample interview data.

6.1 Test Data

The pipeline was tested with the following input data:

- **Video**: 385.33 seconds (6.4 minutes) candidate video recording at 24 fps
- **Candidate Audio**: Synchronized audio track at 48kHz sample rate
- **Interviewer Audio**: Synchronized audio track at 48kHz sample rate
- **Job Description**: Machine Learning Engineer position requirements

6.2 Pipeline Execution Results

The pipeline was executed successfully, producing the following outputs:

Table 6: Pipeline Output Statistics

Stage	Output
Stage 0	1 timeline record
Stage 1A	3854 audio feature frames, 214 VAD segments, 48 transcriptions
Stage 1B	20 interviewer segments
Stage 1C	925 sampled video frames
Stage 2	429 speaking segments, 17 Q&A pairs
Stage 3	214 behavioral metric records
Stage 4+5	17 relevance scores, 1 final verdict

6.3 Sample Analysis Results

A sample Q&A pair analysis result:

```
{
  "checkpoint": "Q1",
  "question": "Give me a Brief overview of your background...",
  "relevance_score": 0.8,
  "matched_keywords": ["Python", "machine learning", "distributed systems"],
  "competency_scores": {
    "technical_depth": 0.8,
    "system_design": 0.7,
    "production_experience": 0.9,
    "communication_clarity": 0.7,
    "problem_solving": 0.7
  },
  "incremental_verdict": "strong_progress"
}
```

Final Verdict:

```
{
  "verdict": "STRONG_HIRE",
  "confidence": "HIGH",
  "overall_score": 0.9,
  "reason": "The candidate demonstrates strong technical depth and production
      experience..."
}
```

6.4 Behavioral Metrics Analysis

Sample behavioral metrics for a speaking segment:

- **Audio Metrics**:
  - Pitch Mean: 836.15 Hz
  - Energy Mean: 0.044
  - Speech Rate: 9.9 words/segment
  - Pause Density: 0.21
- **Video Metrics**:
  - Face Presence Ratio: 1.0 (100%)
  - Head Motion Mean: 0.0
  - Gaze Stability: 0.85

These metrics indicate confident, engaged communication with stable gaze and minimal head movement.

Performance Analysis:

The pipeline execution time breakdown:

1. **Stage 0 (Timebase)**: 1 second
2. **Stage 1A (Candidate Audio)**: 30-60 seconds (Whisper transcription is the bottleneck)
3. **Stage 1B (Interviewer Audio)**: 15-30 seconds
4. **Stage 1C (Candidate Video)**: 60-120 seconds (frame processing)
5. **Stage 2 (Temporal)**: 5 seconds
6. **Stage 3 (Behavioral)**: 10 seconds
7. **Stage 4+5 (Semantic)**: 60-180 seconds (LLM inference per Q&A)

Total pipeline execution time: Approximately 3-7 minutes depending on interview length and Q&A count.

7 Interview Recording UI

The Interview Recording UI is a WebRTC-based browser interface for conducting and recording video interviews. It provides synchronized video and audio capture for both the interviewer and candidate, producing standardized output files that can be directly processed by the backend pipeline.

7.1 Overview

The Interview Recording UI consists of:

1. **Backend Server**: FastAPI-based server handling WebSocket connections and media streaming
2. **WebRTC Implementation**: Using the aiortc library for peer-to-peer communication
3. **Frontend Interface**: HTML/JavaScript-based recording interface
4. **Media Recording**: Synchronized video and audio capture using MediaRecorder API

The system implements a room-based model where the interviewer and candidate join a shared session, and their media streams are captured and recorded simultaneously.

7.2 System Architecture

The Interview Recording UI follows a client-server architecture with WebSocket-based real-time communication:

1. **Server (server.py)**: FastAPI application managing WebSocket connections, room state, and media recording
2. **Browser Client**: WebRTC-enabled browser capturing local video/audio
3. **Media Relay**: Server-side component forwarding media streams to recording handlers

The server implements a state machine for room management:

- **IDLE**: Initial state, no participants connected
- **INTERVIEWER_CONNECTED**: Interviewer has joined, waiting for candidate
- **BOTH_CONNECTED**: Both participants joined, ready to record
- **RECORDING**: Active recording in progress

7.3 WebRTC Implementation

The Interview Recording UI uses aiortc, a Python implementation of WebRTC, to handle real-time media streaming on the server side.

Synchronized Video Track:

```
class SynchronizedVideoTrack(MediaStreamTrack):
    """
    A video track that uses frame count for perfectly consistent timing.
    """
    kind = "video"

    def __init__(self, track: MediaStreamTrack):
        super().__init__()
        self.track = track
        self.frame_count = 0
        self.fps = 30
        self.time_base = Fraction(1, 90000)  # 90kHz for video

    async def recv(self):
        frame = await self.track.recv()
        # Set PTS based on frame count for perfect 30fps timing
        frame.pts = int(self.frame_count * 90000 / self.fps)
        frame.time_base = self.time_base
        self.frame_count += 1
        return frame
```

The SynchronizedVideoTrack class ensures that video frames are timestamped consistently for perfect synchronization between video and audio.

Synchronized Audio Track:

```
class SynchronizedAudioTrack(MediaStreamTrack):
    """
    An audio track that uses sample count for perfectly consistent timing.
    """
    kind = "audio"

    def __init__(self, track: MediaStreamTrack):
        super().__init__()
        self.track = track
        self.sample_count = 0
        self.sample_rate = 48000
        self.time_base = Fraction(1, 48000)  # 48kHz for audio

    async def recv(self):
        frame = await self.track.recv()
        frame.pts = self.sample_count
        frame.time_base = self.time_base
        if hasattr(frame, "samples"):
            self.sample_count += frame.samples
        else:
            self.sample_count += 960  # Default 20ms frame at 48kHz
        return frame
```

The SynchronizedAudioTrack class similarly ensures consistent audio timestamping.

7.4 Room State Management

The RoomState class manages the lifecycle of an interview session:

```
class RoomState:
    IDLE = "IDLE"
    INTERVIEWER_CONNECTED = "INTERVIEWER_CONNECTED"
    BOTH_CONNECTED = "BOTH_CONNECTED"
    RECORDING = "RECORDING"
    FINISHED = "FINISHED"

    def __init__(self):
        self.state = self.IDLE
        self.interviewer_pc = None
```

```
        self.candidate_pc = None
        self.interviewer_track = None
        self.candidate_track = None
        self.recorder = None
        self.start_time = None
        self.room_id = None
```

The room state transitions through the following lifecycle:

1. **Creation**: Room is created with IDLE state when the interviewer initiates
2. **Interviewer Join**: State transitions to INTERVIEWER_CONNECTED
3. **Candidate Join**: State transitions to BOTH_CONNECTED
4. **Recording Start**: State transitions to RECORDING
5. **Recording Stop**: State transitions to FINISHED
6. **Cleanup**: Room is cleaned up and removed

7.5 WebSocket API

The Interview Recording UI exposes WebSocket endpoints for real-time communication:

- **/ws/room/{room_id}**: Main WebSocket endpoint for participants
- **/ws/control/{room_id}**: Control endpoint for recording commands

The WebSocket protocol handles:

- **Connection**: Participants connect and identify their role (interviewer/candidate)
- **WebRTC Signaling**: SDP offers/answers and ICE candidates exchanged
- **Control Messages**: Start/stop recording, room management
- **Status Updates**: Real-time room state notifications

The WebSocket protocol handles:

- **Connection**: Participants connect and identify their role (interviewer/candidate)
- **WebRTC Signaling**: SDP offers/answers and ICE candidates exchanged
- **Control Messages**: Start/stop recording, room management
- **Status Updates**: Real-time room state notifications

Client-Side Connection:
The frontend establishes a WebSocket connection and performs the WebRTC handshake:

```
// Establish WebSocket connection
const ws = new WebSocket(`ws://${serverUrl}/ws/room/${roomId}`);

// Create peer connection
const pc = new RTCPeerConnection(servers);

// Add local tracks
localStream.getTracks().forEach(track => {
    pc.addTrack(track, localStream);
});

// Handle remote tracks
pc.ontrack = event => {
    remoteStream.addTrack(event.track);
};

// Create and send offer
const offer = await pc.createOffer();
await pc.setLocalDescription(offer);
ws.send(JSON.stringify({ type: 'offer', sdp: offer.sdp }));
```

7.6 Media Recording

The frontend uses the browser's MediaRecorder API to capture and save media:

1. **Video Recording**: Captures the local video element showing the participant
2. **Audio Recording**: Captures audio from the microphone
3. **Synchronization**: Video and audio are recorded with matching timestamps

Recording Implementation:

```
// Create media recorder for video
const videoRecorder = new MediaRecorder(videoStream, {
    mimeType: 'video/webm;codecs=vp9'
});

// Create separate audio recorder
const audioRecorder = new MediaRecorder(audioStream, {
    mimeType: 'audio/webm;codecs=opus'
});

// Collect chunks
const videoChunks = [];
videoRecorder.ondataavailable = e => videoChunks.push(e.data);

audioRecorder.ondataavailable = e => audioChunks.push(e.data);

// Start recording
videoRecorder.start();
audioRecorder.start();

// Stop and save
videoRecorder.stop();
const videoBlob = new Blob(videoChunks, { type: 'video/webm' });
```

The recorded files are saved locally and can be uploaded to the backend pipeline for processing.

7.7 Interview UI Features

The Interview Recording UI provides the following features:

1. **Room Creation**: Interviewer creates a room and receives a room ID to share

2. **Room Joining**: Candidate joins using the room ID
3. **Real-time Preview**: Both participants see each other's video in real-time
4. **Recording Control**: Start/stop recording buttons for the interviewer
5. **Local Recording**: Each participant can record their own local feed
6. **File Download**: Recorded interviews can be downloaded as MP4/WAV files
7. **Standardized Output**: Produces files in the exact format required by the pipeline

The UI is designed to be simple and intuitive, requiring minimal technical knowledge from users. All complex WebRTC operations are handled transparently in the background.

8 Dashboard

The TIPS Dashboard is a fully implemented interactive web interface for visualizing interview analysis results. It provides comprehensive views for exploring behavioral metrics, semantic scores, and final recommendations.

The Dashboard reads directly from the JSON output files generated by the backend pipeline, making it independent of the processing logic. This architecture offers several advantages:

- **Decoupling**: The dashboard does not need to understand the internal workings of the pipeline
- **Flexibility**: New pipeline stages can be added without modifying the dashboard
- **Portability**: The JSON files can be moved or archived independently
- **Security**: The dashboard operates on read-only data

The Dashboard is implemented as a Single Page Application (SPA) using vanilla JavaScript with ES6 modules.

8.1 Dashboard Architecture

The Dashboard is organized into five main views:

1. **Session Hub**: Overview page with verdict, stats, and competency radar chart
2. **Temporal Evidence**: Chronological timeline with Q&A pairs and behavioral signals
3. **Analytics**: Detailed charts including score trajectory, competency breakdown, and keyword heatmap
4. **Pipeline Execution**: Processing status and output file information
5. **Q&A Review**: Full transcript with scores and LLM reasoning per question

The Dashboard architecture consists of the following JavaScript modules:

- **main.js**: Application entry point, initialization, and modal handling
- **router.js**: SPA routing for page navigation
- **dataLoader.js**: JSON file loading and session data management
- **config.js**: Configuration constants and competency definitions
- **pages/hub.js**: Session Hub page renderer
- **pages/temporalEvidence.js**: Temporal Evidence page renderer **pages/analytics.js**: Analytics page renderer **pages/pipelineExecution.js**: Pipeline Execution page renderer **pages/qaReview.js**: Q&A Review page renderer

8.2 Session Hub

The Session Hub is the main landing page of the Dashboard, providing an at-a-glance summary of the interview analysis.

Key Components:

1. **Verdict Banner**: Displays the final hiring recommendation (STRONG_HIRE, HIRE, BORDERLINE, NO_HIRE) with confidence level and justification
2. **Gauge Chart**: Visual representation of the overall score as a percentage
3. **Statistics Cards**: Total duration, question count, speaking time, and average relevance
4. **Competency Radar Chart**: Visualizes the five competency dimensions
5. **Session Metadata**: Dataset ID, timebase, video FPS, frame count, and sample rate
6. **Quick Navigation**: Cards for navigating to other Dashboard views

Verdict Banner Implementation:

The verdict banner dynamically styles based on the verdict type:

```
const cssClass = verdict.toLowerCase().replace(/_/g, '-')
    .replace('strong-hire', 'strong-hire')
    .replace('weak-hire', 'weak-hire')
    .replace('no-hire', 'no-hire');
```

Different CSS classes apply appropriate color schemes:

- **strong-hire**: Green theme (confident positive recommendation)
- **hire**: Blue theme (standard positive recommendation)
- **borderline**: Yellow/amber theme (uncertain recommendation)
- **no-hire**: Red theme (negative recommendation)

Gauge Chart:

The gauge chart uses Chart.js doughnut chart to display the overall score:

```
const gaugeChart = new Chart(canvas, {
    type: 'doughnut',
    data: {
        datasets: [{
            data: [score, 1 - score],
            backgroundColor: [color, 'rgba(48,54,61,0.6)'],
            borderWidth: 0,
            circumference: 240,
            rotation: 240,
        }]
    },
    options: {
        responsive: false,
        cutout: '75%',
        plugins: { legend: { display: false }, tooltip: { enabled: false } }
    }
});
```

The gauge displays a score from 0-100% with color coding:

- Green (score ¿= 70%): Strong positive
- Yellow (40% ¡= score ¡ 70%): Moderate
- Red (score ¡ 40%): Weak

8.3 Temporal Evidence View

The Temporal Evidence View provides a chronological timeline of the interview with synchronized audio, video signals, and transcripts.

Key Features:

1. **Unified Timeline**: Visual representation of the entire interview with Q&A segments
2. **Synchronized Audio Timeline**: Clickable segments that show question/answer pairs with transcript
3. **Score Trajectory Chart**: Line graph showing relevance score progression
4. **Behavioral Signals Chart**: Toggleable graphs for:
   - Vocal Energy (RMS)
   - Speech Rate (words per minute)
   - Gaze Stability (0-1 scale)
5. **Segment Details**: Expandable panels showing metrics for each speaking segment

The Timeline Bar serves as the central navigation element:

- Horizontal bar representing full interview duration
- Color-coded segments for different activities (questions, answers, silence)
- clicking jumps Interactive - to that segment
- Shows relevance scores at Q&A checkpoints

The Score Trajectory Chart displays how the candidate's relevance score evolves throughout the interview:

- X-axis: Question number / checkpoint
- Y-axis: Relevance score (0.0-1.0)
- Line connects consecutive checkpoints
- Visual indicators for incremental verdicts (color-coded dots)

The Behavioral Signals Chart provides multi-series visualization:

- Toggle buttons to show/hide individual metrics
- Vocal Energy: Shows candidate's volume/assertiveness over time
- Speech Rate: Words per minute indicating fluency
- Gaze Stability: Eye contact quality throughout the interview

8.4 Analytics View

The Analytics View provides aggregate analysis and high-level insights into candidate performance through various visualizations.

Components:

1. **Question-wise Relevance Score Graph**: Bar chart showing scores for each Q&A pair
2. **Question-wise Breakdown**: Detailed list of questions with scores and matched keywords
3. **Competency Breakdown Graph**: Radar chart showing the five competency dimensions
4. **Keyword Match Heatmap**: Visual representation of technical terms matched from job description
5. **Behavioral Distributions**: Histograms/box plots of audio and video metrics
6. **Performance Trajectory**: Trend analysis showing candidate performance evolution

Question-wise Relevance Score:
Bar chart visualization where:

- Each bar represents one Q&A pair
- Bar height indicates relevance score (0.0-1.0)
- Color coding indicates verdict strength
- Hover shows question text and matched keywords

Competency Breakdown:
Radar chart with five axes:

1. **Technical Depth**: Understanding of technical concepts
2. **System Design**: Ability to design scalable systems
3. **Production Experience**: Real-world deployment knowledge
4. **Communication Clarity**: Ability to articulate ideas
5. **Problem Solving**: Approach to tackling technical challenges

The radar chart uses Chart.js:

```
const radarChart = new Chart(canvas, {
    type: 'radar',
    data: {
        labels: ['Technical Depth', 'System Design', 'Production Experience',
                 'Communication Clarity', 'Problem Solving'],
        datasets: [{
            label: 'Competency',
            data: values,
            backgroundColor: 'rgba(56,139,253,0.15)',
```

```
            borderColor: '#388bfd',
            borderWidth: 2,
            pointBackgroundColor: '#388bfd',
        }]
    },
    options: {
        scales: {
            r: {
                min: 0, max: 100,
                ticks: { stepSize: 25 },
                grid: { color: 'rgba(48,54,61,0.8)' }
            }
        }
    }
});
```

Keyword Match Heatmap:
Visual display of technical terms extracted from job description that appear in candidate responses:

- Grid layout with keywords as cells
- Cell size indicates frequency/importance
- Color intensity shows relevance to job requirements
- Grouped by technical categories

Behavioral Distributions:
Statistical distributions of behavioral metrics:

- **Pitch Distribution**: Histogram of fundamental frequency values
- **Energy Distribution**: Histogram of RMS energy values
- **Speech Rate Distribution**: Histogram of words per minute
- **Gaze Stability Distribution**: Histogram of eye contact quality scores
- **Face Presence Distribution**: Percentage of frames with detected face

8.5 Pipeline Execution View

The Pipeline Execution View provides transparency into the backend processing that generated the analysis results.

Information Displayed:

1. **Stage Status**: Each pipeline stage with execution status
2. **Processing Time**: Time taken by each stage to complete
3. **Output Files**: List of all generated JSON files with paths and sizes
4. **Execution Logs**: Timestamped log entries for debugging
5. **Dataset Information**: Metadata about the processed interview

This view is particularly useful for:

- Technical users and administrators
- Debugging processing issues
- Understanding data flow
- Verifying pipeline execution

8.6 Q&A Review View

The Q&A Review View provides a detailed, text-based exploration of each question-answer pair.

Features:

1. **Question List**: Collapsible list of all Q&A pairs
2. **Expanded Detail View**: When clicked, shows:
   - Full question text
   - Full answer text
   - Relevance score with justification
   - Matched keywords
   - Competency scores for that specific answer
   - Incremental verdict
3. **LLM Reasoning**: Shows the LLM's assessment for each answer
4. **Historical Context**: Shows how previous answers influenced current assessment

This view is designed for:

- Detailed manual review of specific responses
- Understanding the reasoning behind scores
- Identifying areas of strength and weakness
- Preparing for follow-up interviews

9 Dashboard Implementation Details

This section provides detailed technical documentation of the Dashboard implementation.

9.1 File Structure

```
dashboard/
|-- index.html              # Main HTML entry point
|-- css/
|   |-- main.css            # Core styles
|   |-- components.css      # Reusable UI components
|   |-- charts.css          # Chart-specific styles
|-- js/
    |-- main.js             # Application entry point
    |-- router.js           # SPA routing
    |-- dataLoader.js       # JSON file loading
    |-- config.js           # Configuration constants
    |-- pages/
        |-- hub.js          # Session Hub renderer
        |-- temporalEvidence.js
        |-- analytics.js
        |-- pipelineExecution.js
        |-- qaReview.js
```

9.2 Configuration

The Dashboard uses a centralized configuration in `config.js`:

```
export const CONFIG = {
    OUTPUT_DIR: '../backend/backend/output/',
    FILES: {
        timeline: 'timeline.json',
        qa_pairs: 'qa_pairs.json',
        checkpoints: 'candidate_score_timeline.json',
        final_verdict: 'candidate_score_timeline.json',
        behavior_metrics: 'candidate_behavior_metrics.json',
        speaking_segments: 'speaking_segments.json',
        relevance_scores: 'relevance_scores.json',
    },
    COMPETENCY_KEYS: [
        { key: 'technical_depth', label: 'Technical Depth' },
        { key: 'system_design', label: 'System Design' },
        { key: 'production_experience', label: 'Production Experience' },
        { key: 'communication_clarity', label: 'Communication Clarity' },
        { key: 'problem_solving', label: 'Problem Solving' },
    ],
    CHART_COLORS: {
        primary: '#388bfd',
        success: '#3fb950',
        warning: '#d29922',
        danger: '#f85149',
    }
};
```

9.3 Data Loading

The dataLoader.js module handles loading JSON files from the pipeline output directory:

```
export async function loadFromPath(path, fileConfig, onProgress) {
    const results = {};
    const files = Object.keys(fileConfig);
    let loaded = 0;

    for (const key of files) {
        const filename = fileConfig[key];
        try {
```

```
            const response = await fetch(`${path}/${filename}`);
            if (response.ok) {
                results[key] = await response.json();
            }
        } catch (e) {
            console.error(`Failed to load ${filename}:`, e);
        }
        loaded++;
        onProgress(loaded, files.length);
    }

    return results;
}
```

The data loader supports multiple loading modes:

1. **Path Mode**: Load files from a directory path via HTTP
2. **Folder Picker**: User selects a folder, files are matched by name
3. **Individual Files**: User selects each JSON file separately

9.4 Session Data Structure

After loading, the session data is available in a unified structure:

```
const sessionData = {
    timeline: { ... },           // from timeline.json
    qa_pairs: { ... },           // from qa_pairs.json
    checkpoints: [ ... ],        // from candidate_score_timeline.json
    final_verdict: { ... },      // extracted from last checkpoint
    behavior_metrics: { ... },  // from candidate_behavior_metrics.json
    speaking_segments: [ ... ],  // from speaking_segments.json
    relevance_scores: [ ... ],   // from relevance_scores.json
    errors: { ... },             // Track any loading errors
};
```

Helper functions provide convenient access to computed values:

```
export function totalSpeakingTime(segments) {
    return segments
        .filter(s => s.type === 'speaking')
        .reduce((sum, s) => sum + (s.end_time - s.start_time), 0);
}

export function avgCompetency(checkpoints, key) {
    if (!checkpoints.length) return 0;
    const sum = checkpoints.reduce((acc, cp) =>
        acc + (cp.competency_scores?.[key] || 0), 0);
    return sum / checkpoints.length;
}

export function scoreColor(score) {
    return score >= 0.7 ? '#3fb950' :
           score >= 0.4 ? '#d29922' : '#f85149';
}
```

9.5 Routing

The router.js module implements client-side SPA routing:

```
export function initRouter() {
    window.addEventListener('hashchange', handleRoute);
    handleRoute(); // Initial route
}

function handleRoute() {
    const hash = window.location.hash.replace('#', '') || 'hub';
    showPage(hash);
```

```
}

export function navigateTo(page) {
    window.location.hash = page;
}

function showPage(pageName) {
    // Hide all pages
    document.querySelectorAll('.page-view').forEach(el => {
        el.classList.remove('active');
    });

    // Show target page
    const pageEl = document.getElementById(`page-${pageName}`);
    if (pageEl) {
        pageEl.classList.add('active');
    }

    // Update nav tabs
    document.querySelectorAll('.nav-tab').forEach(tab => {
        tab.classList.toggle('active', tab.dataset.page === pageName);
    });
}
```

10 Dashboard Photo Documentation

This section provides visual documentation of the Dashboard interface with screenshots and descriptions.

10.1 Load Session Modal

Figure 3 shows the Load Session modal that appears when users click the "Load Session" button in the navigation bar. This modal provides multiple ways to load interview analysis data:

- **Folder Picker (Recommended)**: Browse to the output directory and select all JSON files
- **URL Path**: Enter the relative path to the output directory
- **Individual Files**: Select each JSON file separately

The modal displays helpful hints about where to find the output files and shows loading progress.

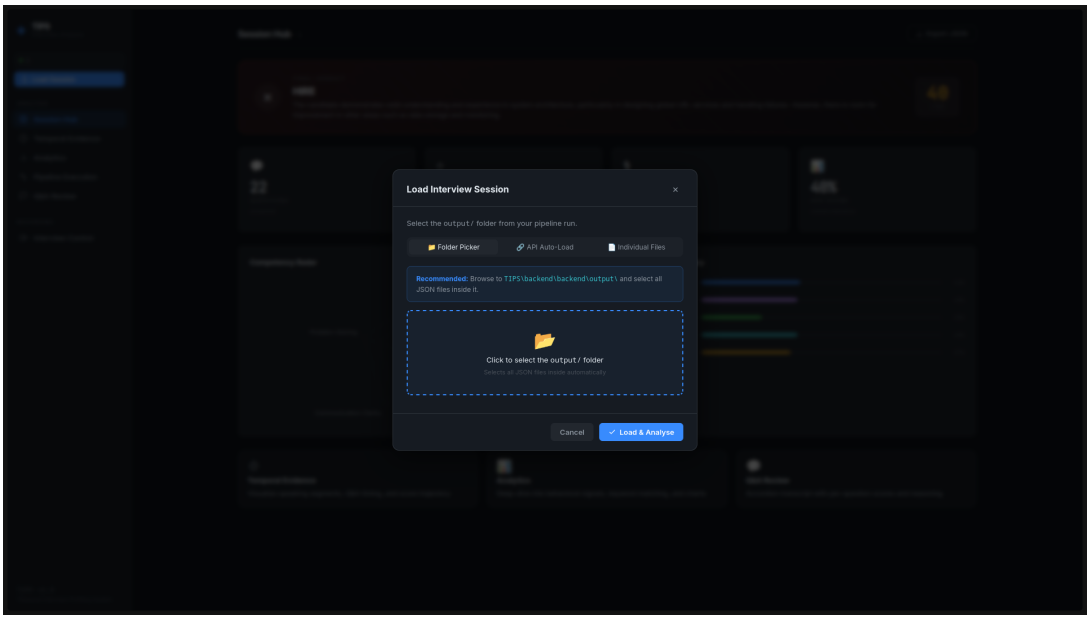

Figure 3: Load Session Modal - Initial State

Figure 4 shows the Load Session modal after selecting a folder. The file list displays all the JSON files found in the output directory, indicating which ones matched the expected file names:

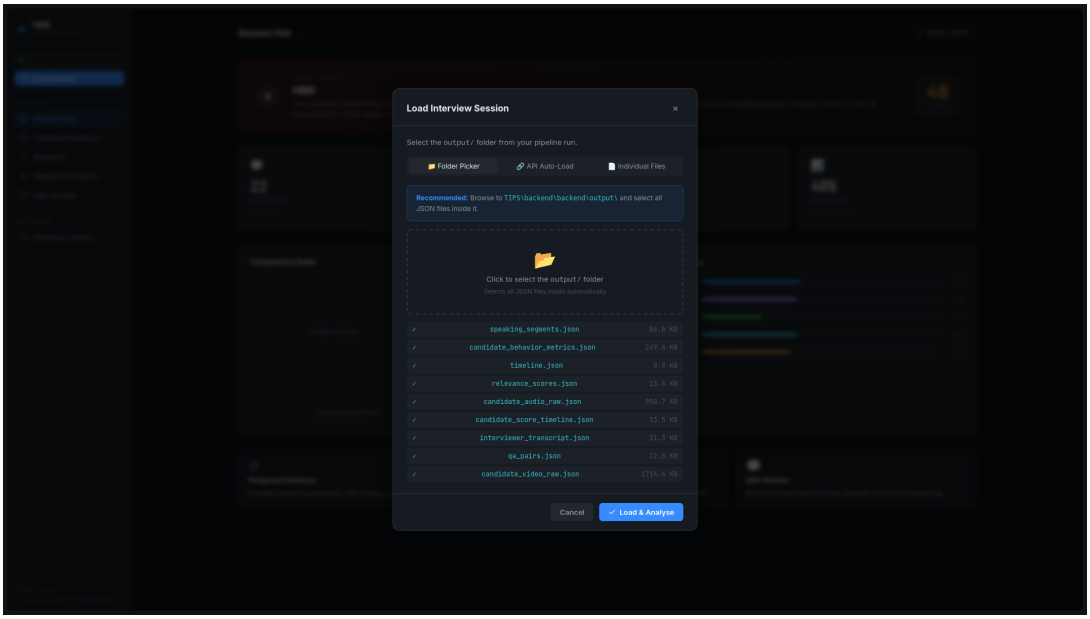

Figure 4: Load Session Modal - Files Selected

10.2 Session Hub

Figure 5 shows the Session Hub, the main landing page of the Dashboard after loading an interview session. The key components visible are:

1. **Verdict Banner**: Shows "STRONG_HIRE" with green theme, overall score (90%)
2. **Confidence Level**: "HIGH" confidence
3. **Statistics Cards**: Total Duration, Questions, Speaking Time, Avg Relevance
4. **Competency Radar Chart**: Visualizes the five competency dimensions
5. **Session Metadata**: Dataset ID, timebase, video FPS, frame count, sample rate
6. **Quick Navigation Cards**: Links to Temporal Evidence, Analytics, Pipeline Execution, and Q&A Review

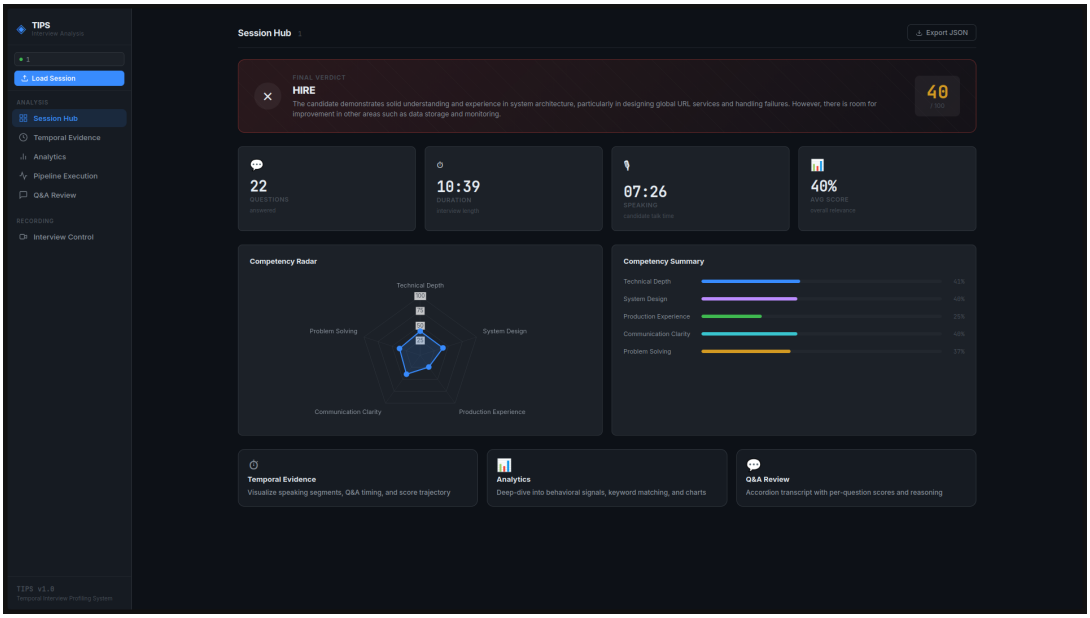

Figure 5: Session Hub - Main Dashboard Overview

10.3 Temporal Evidence View

Figure 6 shows the Temporal Evidence view with the interview timeline. This view provides:

1. **Timeline Bar**: Horizontal bar showing Q&A segments
2. **Score Overlay**: Shows relevance scores at each checkpoint
3. **Vertical Markers**: Indicating question boundaries
4. **Synchronization Controls**: Ability to play back video with transcripts

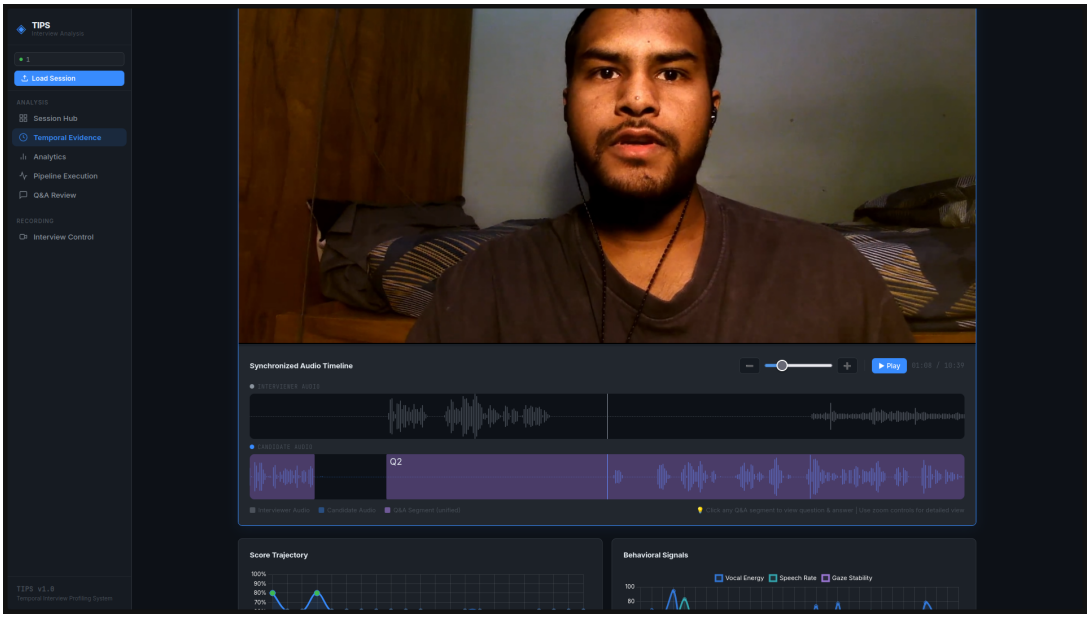

Figure 6: Temporal Evidence - Timeline Overview

Figure 7 shows the Synchronized Audio Timeline feature. When a user clicks on a Q&A segment, the transcript panel opens showing:

1. **Question Text**: The interviewer's question
2. **Answer Text**: The candidate's full response
3. **Question Start/End Times**: Timestamps for the question
4. **Answer Start/End Times**: Timestamps for the answer
5. **Video Navigation**: Click to jump to that segment in the video

This feature allows recruiters to quickly navigate to specific Q&A pairs and review responses in context.

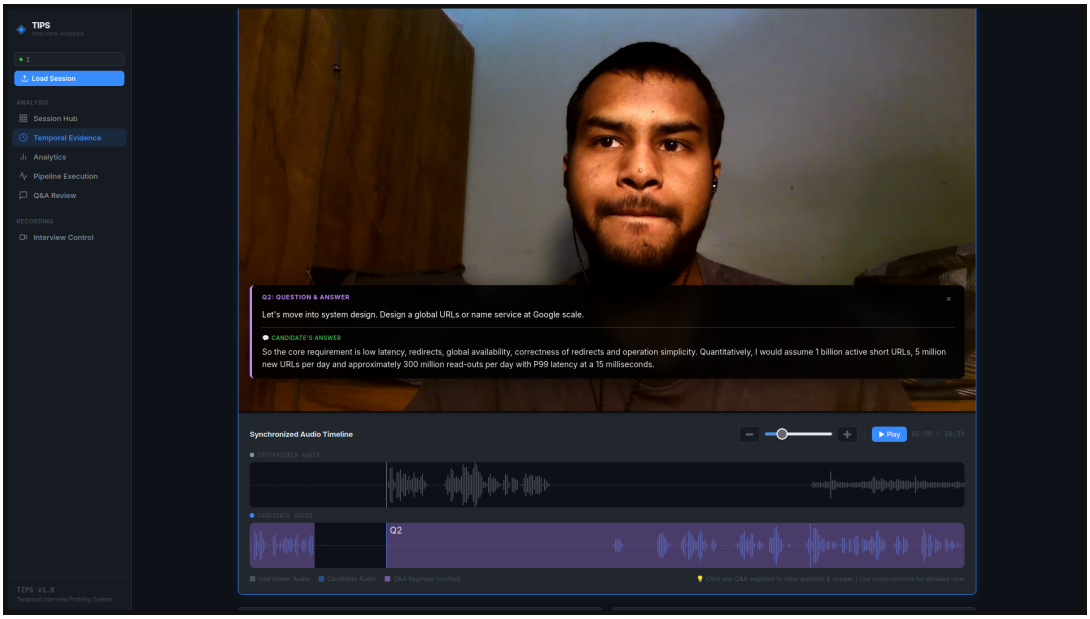

Figure 7: Temporal Evidence - Synchronized Transcript

Figure 8 shows the Score Trajectory and Behavioral Signals charts:

1. **Score Trajectory Chart**: Line graph showing relevance score progression across questions
2. **Behavioral Signals Chart**: Multi-series graph with toggles for:
   - Vocal Energy (RMS)
   - Speech Rate (words per minute)
   - Gaze Stability (0-1 scale)
3. **Toggle Buttons**: Enable/disable individual signal series

Figure 8: Temporal Evidence - Score Trajectory and Behavioral Signals

10.4 Analytics View

Figure 9 shows the Analytics view with the Question-wise Relevance Score graph:

1. **Bar Chart**: Each bar represents a Q&A pair
2. **Score Labels**: Shows the relevance score for each question
3. **Color Coding**: Visual indication of score strength

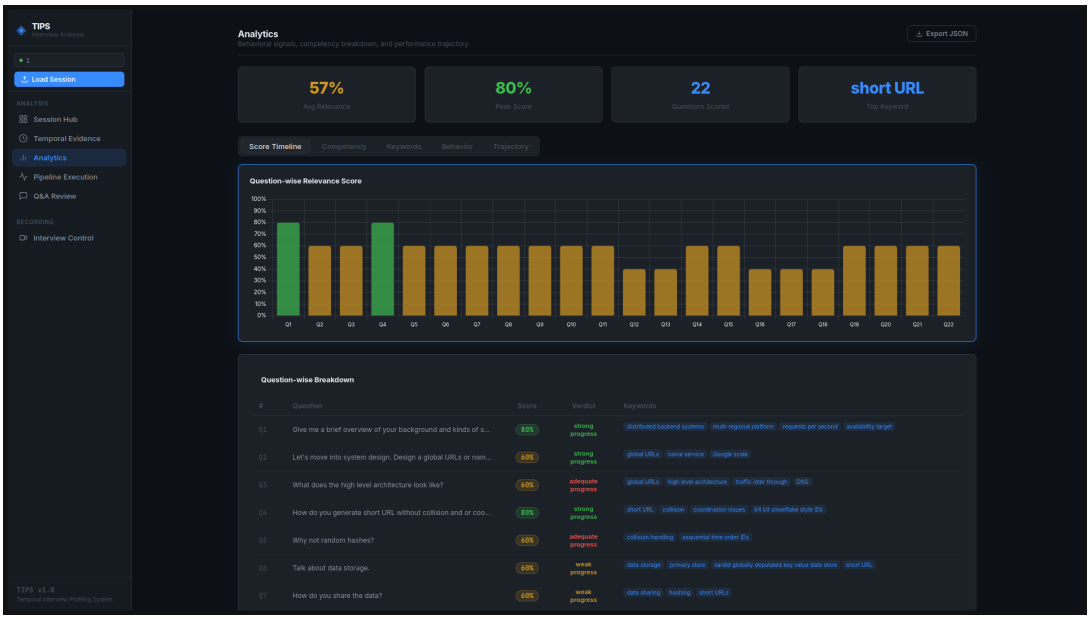

Figure 9: Analytics - Question-wise Relevance Score

Figure 10 shows the Question-wise Breakdown section:

1. **Question List**: Expandable list of all Q&A pairs
2. **Per-Question Details**: Score, matched keywords, and verdict for each question
3. **Expand/Collapse**: Click to see full answer text and reasoning

Figure 10: Analytics - Question-wise Breakdown

Figure 11 shows the Competency Breakdown radar chart:

1. **Radar Chart**: Five-axis visualization of competency dimensions
   - Technical Depth
   - System Design
   - Production Experience
   - Communication Clarity
   - Problem Solving
2. **Percentage Scores**: Values for each competency (e.g., 82%, 73%, 92%)
3. **Shape Analysis**: The shape of the radar indicates strengths and weaknesses

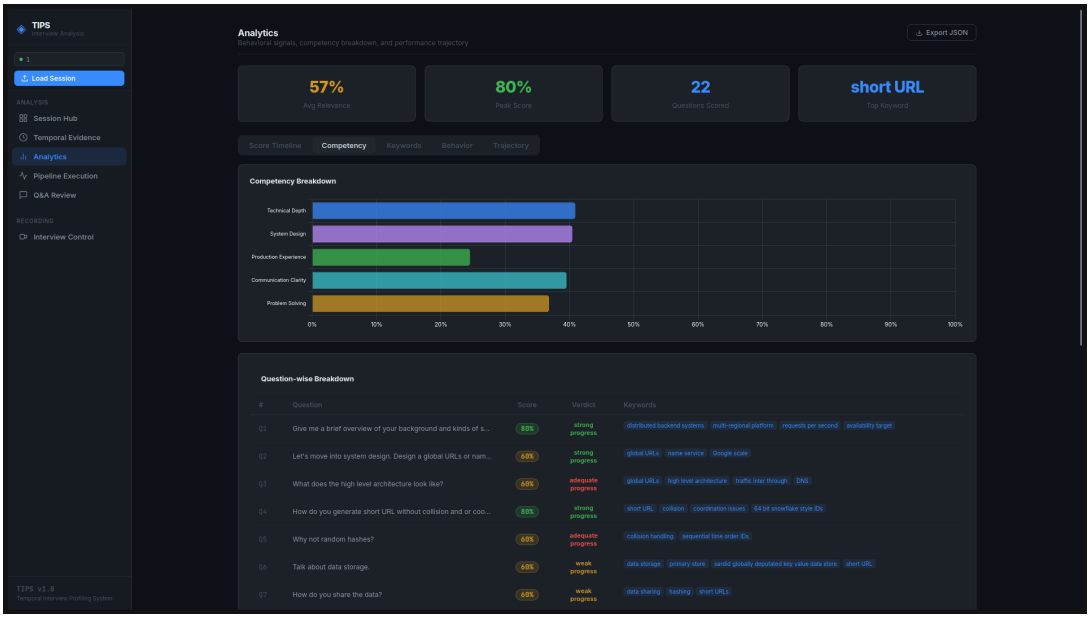

Figure 11: Analytics - Competency Breakdown Radar Chart

Figure 12 shows the Keyword Match Heatmap:

1. **Keyword Grid**: Visual display of technical terms
2. **Category Grouping**: Keywords organized by type
3. **Match Count**: Numbers indicating frequency of occurrence

This visualization helps quickly identify the candidate's technical strengths and alignment with job requirements.

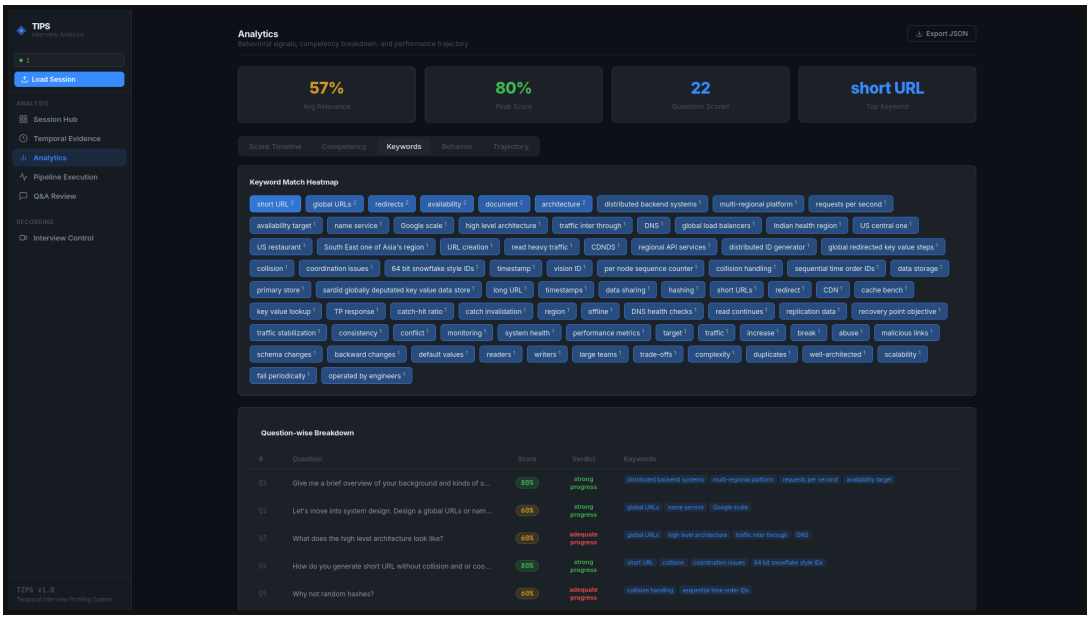

Figure 12: Analytics - Keyword Match Heatmap

Figure 13 shows the Behavioral Distributions:

1. **Audio Metrics**: Pitch, Energy, Speech Rate distributions
2. **Video Metrics**: Face Presence, Gaze Stability distributions
3. **Histograms**: Visual representation of metric value distributions
4. **Statistics**: Mean, median, and range values

Figure 13: Analytics - Behavioral Distributions

Figure 14 shows the Performance Trajectory:

1. **Trend Analysis**: Visual representation of how performance evolved
2. **Early vs Late Questions**: Comparison of performance at different interview stages
3. **Improvement Indicators**: Shows if candidate improved, stayed consistent, or declined

This view helps identify patterns in candidate performance over the course of the interview.

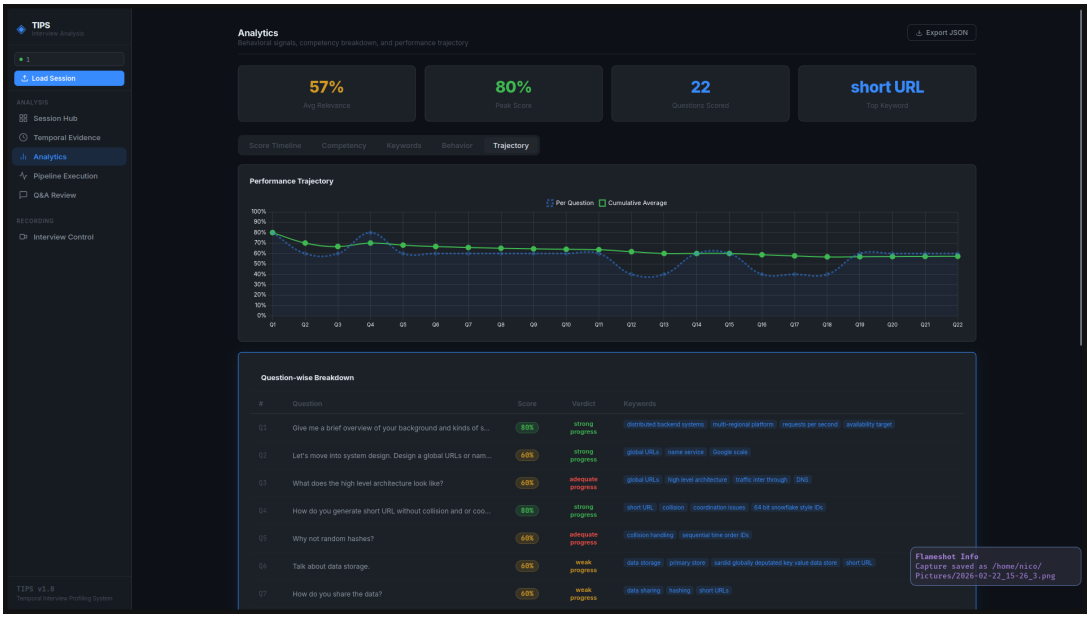

Figure 14: Analytics - Performance Trajectory

10.5 Pipeline Execution View

Figure 15 shows the Pipeline Execution view:

1. **Stage Status**: Read-only information about each pipeline stage
2. **Processing Metadata**: Dataset ID, processing timestamps
3. **Output Summary**: Statistics about the generated output files

This view provides transparency into how the analysis was generated and helps with debugging if needed.

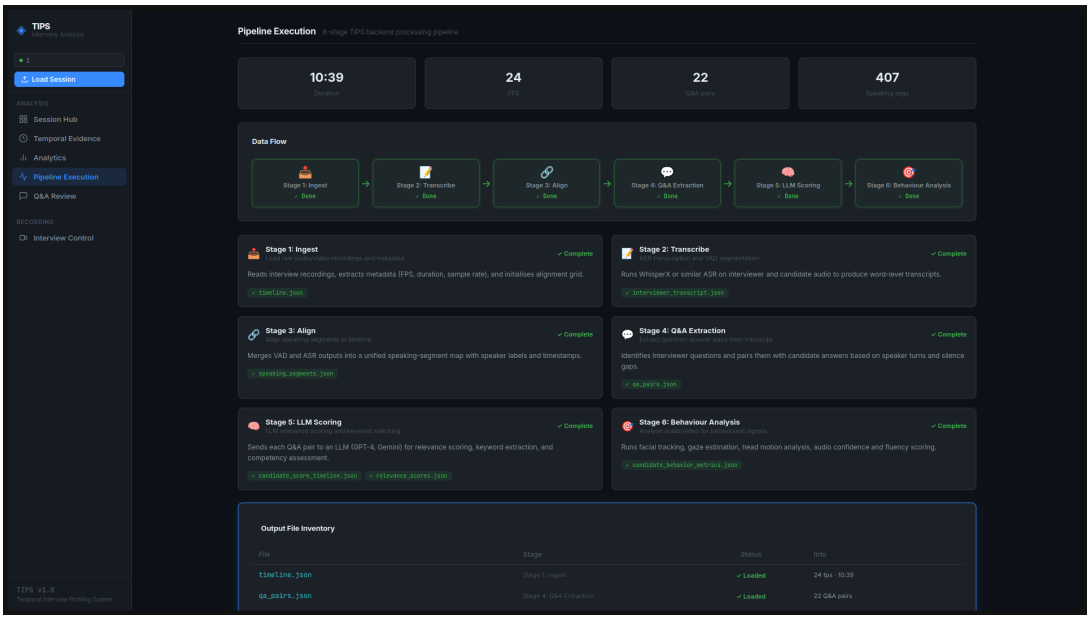

Figure 15: Pipeline Execution - Processing Status

10.6 Q&A Review View

Figure 16 shows the Q&A Review overview:

1. **Question List**: Scrollable list of all Q&A pairs
2. **Score Summary**: Quick view of scores for each question
3. **Verdict Indicators**: Color-coded status for each response

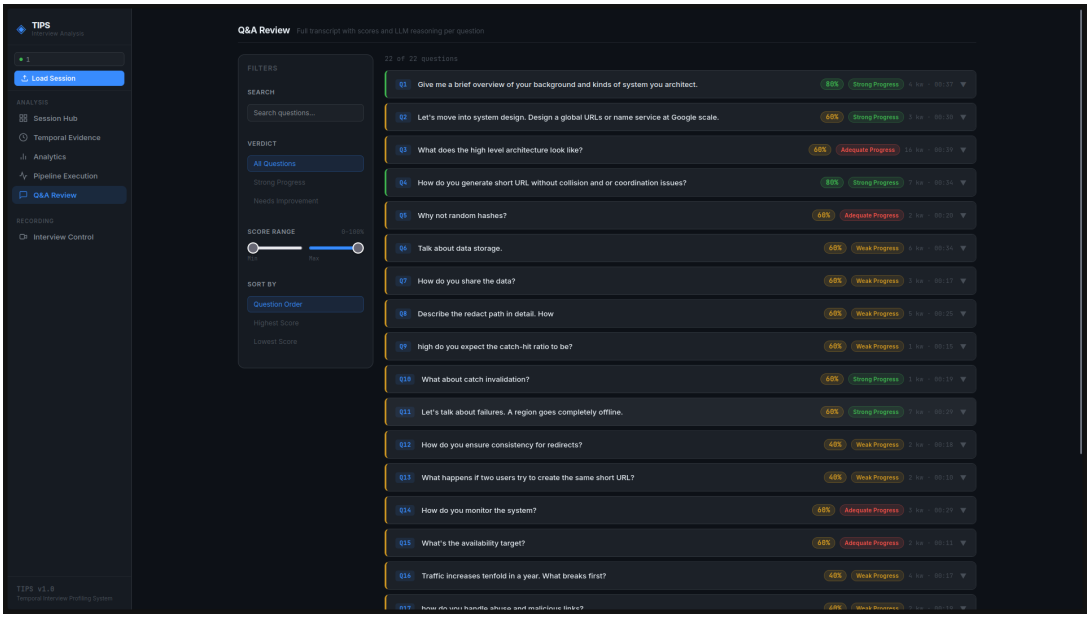

Figure 16: Q&A Review - Overview

Figure 17 shows the expanded Q&A pair detail view:

1. **Full Question Text**: Complete question from interviewer
2. **Full Answer Text**: Complete candidate response
3. **Relevance Score**: Numerical score with justification
4. **Matched Keywords**: Technical terms identified in the answer
5. **Competency Scores**: Five-dimensional scoring for this specific answer
6. **Incremental Verdict**: Assessment at this checkpoint
7. **LLM Reasoning**: Explanation of the scoring

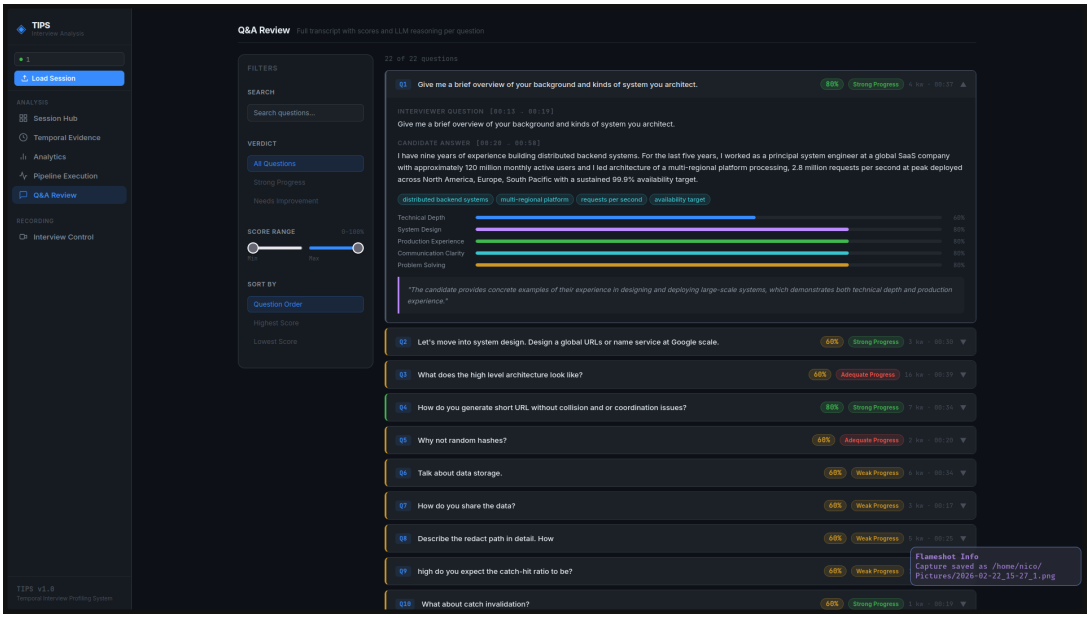

Figure 17: Q&A Review - Expanded Pair Details

11 Interview Recording UI Documentation

This section provides visual documentation of the Interview Recording UI.
Figure 18 shows the Interview Recording UI front page:

1. **Room Creation**: Interface for creating a new interview room
2. **Room ID Display**: Shows the room ID to share with the candidate
3. **Connection Status**: Visual indication of participant connections
4. **Video Preview**: Local video preview for both interviewer and candidate
5. **Recording Controls**: Start/Stop recording buttons

The Interview Recording UI provides a simple, intuitive interface for conducting video interviews with synchronized recording capabilities.

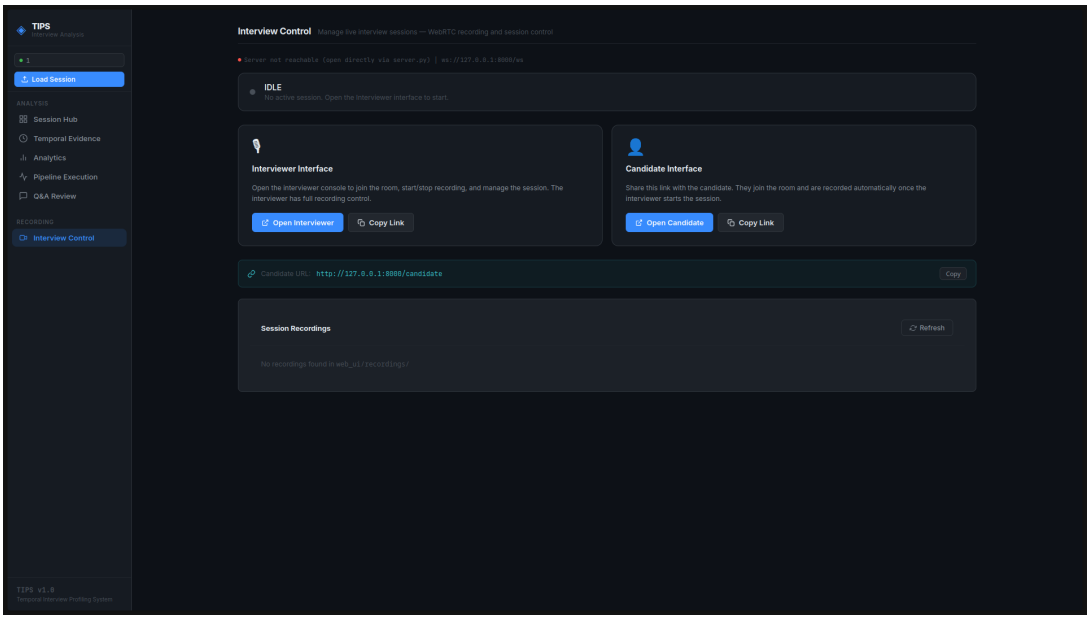

Figure 18: Interview Recording UI - Main Interface

12 Limitations

The current implementation of TIPS has several limitations:

1. **Batch Processing**: The system processes recorded interviews only; analysis is available after post-processing completes.
2. **Language Support**: Optimized for English-language interviews only.
3. **Video Quality**: Poor lighting or camera angles may affect face detection and pose estimation accuracy.
4. **Audio Quality**: Background noise impacts transcription accuracy and VAD performance.
5. **LLM Constraints**: The quantized model may not capture all nuances; token limits may require truncating long answers.
6. **Bias**: The scoring system reflects biases present in the LLM training data and job description.
7. **Cross-Domain Generalization**: The scoring model may not generalize well across different job roles without fine-tuning.
8. **Interviewer Analysis**: Only transcription is performed; interviewer audio is not analyzed for behavioral cues.
9. **Single Camera**: Only single-view video analysis is supported.
10. **Browser Compatibility**: Interview Recording UI requires WebRTC-enabled browsers.
11. **Dashboard Deployment**: Requires a web server to serve static files and load JSON outputs.

13 Scope for Future Enhancement

Several enhancements can be made to improve the TIPS system:

13.1 Enhanced Analysis

1. Add real-time processing capability for live interview support
2. Implement more sophisticated behavioral metrics
3. Add support for multi-party interviews (panel discussions)
4. Integrate additional LLM models for specialized domains
5. Implement multi-language support for transcription and evaluation
6. Add sentiment analysis for emotional tone assessment

13.2 Dashboard Enhancements

1. Add interactive elements for exploring the timeline
2. Include export functionality (PDF reports, CSV data)
3. Add multi-language support for dashboard interfaces
4. Implement comparison views for multiple candidates
5. Add candidate ranking across multiple interviews
6. Implement role-based access controls

13.3 Infrastructure

1. Containerize the pipeline for easier deployment
2. Add API endpoints for integration with existing HR systems
3. Implement caching for frequently processed profiles
4. Add support for distributed processing
5. Create Docker Compose setup for easy deployment
6. Implement authentication and authorization

14 Conclusion

The Temporal Interview Profiling System (TIPS) represents a comprehensive approach to automated interview analysis. By combining multiple signal processing techniques with Large Language Model-based semantic evaluation, TIPS provides a multi-dimensional assessment of candidate performance.

The six-stage pipeline architecture enables modular processing of interview data, from timebase establishment through behavioral metrics computation to LLM-powered semantic scoring. The system produces time-evolving candidate scores that track performance throughout the interview, culminating in a final hiring recommendation with supporting evidence.

Key achievements of this project include:

1. Implementation of a complete end-to-end processing pipeline
2. Integration of state-of-the-art speech recognition (Faster-Whisper) and LLM technology (Qwen2.5-3B-Instruct)
3. Design of a parallel processing architecture that maximizes throughput
4. Creation of comprehensive JSON output schemas for analysis results
5. Implementation of a fully functional Dashboard with five interactive views
6. Development of a WebRTC-based Interview Recording UI
7. Integration of all components into a unified system

The system's design as a backend addon ensures compatibility with existing interview platforms, allowing organizations to leverage TIPS's analytical capabilities without replacing their current infrastructure. The Interview Recording UI provides a complete solution for conducting and recording interviews in the required format.

The Dashboard transforms complex analysis results into intuitive visualizations, enabling hiring teams to quickly assess candidates and drill down into specific details when needed. The five different views (Hub, Temporal Evidence, Analytics, Pipeline Execution, Q&A Review) cater to different user needs and levels of technical expertise.

While the current implementation has limitations—particularly the batch processing model and dependency on English-language processing—the foundation is solid for future enhancements. The modular architecture allows for individual components to be upgraded as technology advances.

In conclusion, TIPS demonstrates the potential of combining traditional signal processing with modern AI to create tools that can assist, rather than replace, human decision-making in the hiring process.

15 References

1. Faster-Whisper: Faster Whisper implementation with CTranslate2. https://github.com/SYSTRAN/faster-whisper
2. Qwen2.5-Instruct. https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
3. MediaPipe: Google's framework for building media processing pipelines. https://mediapipe.dev/
4. WebRTC VAD: Voice Activity Detection. https://github.com/wiseman/py-webrtcvad
5. Librosa: Python library for audio analysis. https://librosa.org/
6. OpenCV: Open Source Computer Vision Library. https://opencv.org/
7. aiortc: Implementation of WebRTC for Python. https://aiortc.readthedocs.io/
8. FastAPI: Modern Python web framework. https://fastapi.tiangolo.com/
9. Transformers: Hugging Face library for state-of-the-art NLP. https://huggingface.co/transformers/
10. BitsAndBytes: Efficient quantization for LLMs. https://github.com/TimDettmers/bitsandbytes
11. Chart.js: Simple yet flexible JavaScript charting library. https://www.chartjs.org/
12. DAF: A Framework for Defining Automated Interview Systems. ACM Computing Surveys.
13. Behavioral Signal Processing: Deriving Behavioral Profiles from Audio-Video Data. IEEE Signal Processing Magazine.
14. End-to-End Speech Recognition: A Review of the Transformer Architecture. Computer Speech and Language.