Backend Processing Pipeline: Interview Intelligence System

MCA Minor Project

January 2024

Contents

1 Introduction

1.1 Processing Philosophy

The backend implements an offline processing approach that prioritizes accuracy and traceability over real-time performance. Media artifacts from the interview interface are processed through deterministic pipeline stages, with each stage producing structured outputs for subsequent analysis. This methodology ensures reproducible results with explicit uncertainty quantification, essential for academic evaluation systems.

1.2 System Scope

The backend processing pipeline is responsible for:

- Audio-to-text transcription with word-level timing accuracy
- Conversation reconstruction and temporal alignment
- Question-answer pair extraction with confidence scoring
- Job description processing and skill weighting
- Incremental candidate profiling with Bayesian updating
- Behavioral signal integration from video analysis
- Final verdict generation with uncertainty quantification

The pipeline does not handle:

- Real-time media streaming or live capture
- Interactive inference during interview sessions
- Model training or fine-tuning from collected data
- User interface rendering or client-side interactions

1.3 Technical Constraints

- Processing performed offline after interview completion
- Deterministic algorithms for reproducible results
- Explicit confidence intervals for all numerical outputs
- Maximum session duration: 2 hours of audio/video
- Processing time: under 30 minutes for 1-hour session

2 Django Architecture

2.1 Application Structure

The backend is organized as a Django project with modular applications:

```
interview_backend/
|-- processing/
|   |-- models.py      # Core data models
|   |-- views.py       # Processing orchestration
|   |-- tasks.py       # Async processing tasks
|   `-- pipeline.py    # Main processing pipeline
|-- transcripts/
|   |-- models.py      # Transcription data
|   |-- asr.py         # Speech recognition
|   `-- alignment.py   # Forced alignment
|-- profiling/
|   |-- models.py      # Profile data
|   |-- profiler.py    # Candidate evaluation
|   `-- verdict.py     # Final evaluation
`-- dashboard/
    |-- views.py       # Read-only data access
    `-- serializers.py # API data formatting
```

2.2 Data Management Strategy

- PostgreSQL with JSONB fields for flexible score storage
- File system storage for media artifacts
- Redis for caching and temporary state management
- Celery for asynchronous task processing and monitoring
- Database connections with connection pooling

2.3 Processing Orchestration

Processing triggered through Django signals and Celery tasks:

1. Media file completion detected by file system watcher
2. Django signal initiates processing pipeline
3. Celery creates task with session identifier
4. Sequential stage execution with dependency tracking
5. Progress monitoring through task state updates
6. Error handling with retry mechanisms

3 Data Models

3.1 Core Entity Models

3.1.1 InterviewSession Model

```
session_id: CharField(unique=True)
start_time: DateTimeField()
end_time: DateTimeField()
jd_text: TextField()
jd_processed: JSONField(default=dict)
status: CharField(choices=['processing', 'completed', 'error'])
created_at: DateTimeField(auto_now_add=True)
updated_at: DateTimeField(auto_now=True)
metadata: JSONField(default=dict)
```

3.1.2 MediaAsset Model

```
session: ForeignKey(InterviewSession)
asset_type: CharField(choices=['candidate_audio',
                               'interviewer_audio',
                               'candidate_video'])
file_path: CharField(max_length=500)
duration: FloatField()
sample_rate: IntegerField(null=True)
bit_rate: IntegerField(null=True)
quality_score: FloatField(default=0.0)
```

3.1.3 TranscriptWord Model

```
session: ForeignKey(InterviewSession)
speaker: CharField(choices=['candidate', 'interviewer'])
word: CharField(max_length=100)
start_time: FloatField()
end_time: FloatField()
confidence: FloatField()
speaker_confidence: FloatField()
```

3.1.4 ScriptTurn Model

```
session: ForeignKey(InterviewSession)
speaker: CharField(choices=['candidate', 'interviewer'])
start_time: FloatField()
end_time: FloatField()
text: TextField()
word_count: IntegerField()
average_confidence: FloatField()
emotion_score: FloatField(null=True)
```

3.1.5 QuestionAnswer Model

```
session: ForeignKey(InterviewSession)
question: TextField()
answer: TextField()
question_start: FloatField()
answer_start: FloatField()
question_end: FloatField()
answer_end: FloatField()
confidence: FloatField()
semantic_similarity: FloatField()
question_type: CharField(choices=['technical',
                          'behavioral',
                          'situational'])
```

3.1.6 CandidateProfileSnapshot Model

```
session: ForeignKey(InterviewSession)
timestamp: FloatField()
technical_depth: FloatField()
jd_relevance: FloatField()
clarity: FloatField()
consistency: FloatField()
confidence: FloatField()
risk_flags: JSONField(default=list)
evidence_summary: JSONField(default=dict)
```

3.1.7 FinalVerdict Model

```
session: OneToOneField(InterviewSession)
hire_probability: FloatField()
confidence: FloatField()
risk_flags: JSONField(default=list)
jd_alignment_score: FloatField()
final_score: FloatField()
evidence_weights: JSONField(default=dict)
processing_time: FloatField()
created_at: DateTimeField(auto_now_add=True)
```

4 Audio Processing Pipeline

4.1 Preprocessing Stage

Audio files undergo standardized preprocessing to ensure quality:

1. Format validation (WAV, mono, 48kHz)
2. Duration verification and synchronization check
3. Silence detection and removal (segments below 0.1 amplitude)

4. Normalization to -3dB peak amplitude
5. Quality assessment for signal-to-noise ratio

4.2 Quality Metrics Implementation

- Signal-to-noise ratio threshold: minimum 20dB for inclusion
- Peak amplitude normalization to -3dB target
- Silent segment removal: gaps longer than 2 seconds
- Artifact detection for clipping and distortion
- Sample rate consistency verification

4.3 ASR with Word Timestamps

Whisper model implementation with forced alignment:

- Model selection: Whisper-base for accuracy/speed balance
- Forced alignment using Montreal Forced Aligner principles
- Confidence scoring calculated per word and turn
- Language auto-detection with English as default

4.4 Processing Parameters

- Chunk size: 30 seconds segments for optimal accuracy
- Overlap between chunks: 2 seconds for boundary continuity
- Beam size: 5 for transcription quality
- Temperature: 0.0 for deterministic output
- Word confidence threshold: 0.3 for inclusion

4.5 Output Format Specification

Each transcribed word generates a TranscriptWord record with:

- Word text and speaker identification
- Precise start and end timestamps (milliseconds)
- Confidence score (0.0 to 1.0)
- Speaker confidence for diarization validation

4.6 Forced Alignment Implementation

Word boundaries refined using duration models:

- Hidden Markov Model alignment for phoneme timing
- Silence gap detection for word boundary detection
- Prosody analysis for sentence boundary inference
- Context-dependent duration models

5 Natural Language Processing

5.1 Sentence Framing Algorithm

Sentences inferred using deterministic rule-based approach:

1. Primary rule: Silence gaps longer than 0.8 seconds
2. Secondary rule: Question intonation pattern detection
3. Tertiary rule: Punctuation prediction using language models
4. Constraint: Maximum duration limit of 15 seconds per segment
5. Minimum word count: 3 words per segment for inclusion

5.2 Quality Assurance Metrics

- Confidence threshold: 0.7 for segment inclusion
- Maximum segment duration: 15 seconds enforced
- Minimum word count: 3 words per segment
- Overlap resolution: prefer shorter segments

5.3 Conversation Script Construction

Script turns assembled from word sequences using:

1. Sort all words by chronological start time
2. Group consecutive words by same speaker within 2-second gaps
3. Merge overlapping segments with speaker priority rules
4. Assign turn boundaries using silence detection
5. Calculate turn-level confidence as average of constituent words

6 Question-Answer Extraction

6.1 Question Detection Algorithm

Interviewer turns classified as questions using multi-factor analysis:

- Interrogative word detection (what, how, why, when, where, who)
- Sentence structure analysis (subject-verb-object patterns)
- Semantic intent classification using transformer models
- Intonation pattern analysis from audio prosody
- Confidence threshold: 0.75 for classification

6.2 Answer Matching Algorithm

Candidate answers linked to questions using scoring function:

$$\text{AnswerScore} = \alpha \cdot \text{TempSim} + \beta \cdot \text{SemSim} + \gamma \cdot \text{ContSim}$$

where:

- TempSim = Temporal proximity score (within 60 seconds)
- SemSim = Semantic similarity using BERT embeddings
- ContSim = Content relevance using cosine similarity
- $\alpha = 0.4$, $\beta = 0.35$, $\gamma = 0.25$

6.3 Confidence Calculation

Q/A pair confidence computed as weighted combination:

- Question detection confidence: 40% weight
- Semantic similarity score: 35% weight
- Temporal alignment score: 25% weight

7 Job Description Processing

7.1 JD Ingestion Pipeline

Raw job description processed through text analysis stages:

1. Text cleaning and normalization
2. Section identification (requirements, responsibilities, qualifications)
3. Skill keyword extraction using Named Entity Recognition
4. Experience level detection (entry, mid, senior, expert)
5. Tool/technology requirement extraction

7.2 Skill Extraction Methods

Skills identified using multiple approaches:

- NER models trained on technical job descriptions
- Common skill dictionary matching with 5000+ entries
- Context-aware classification using sentence embeddings
- Industry-specific terminology recognition

7.3 Weight Assignment Algorithm

Skill importance weights calculated using:

- Frequency analysis within JD text
- Position importance (requirements section weighted higher)
- Industry standard skill classification
- Experience level alignment

7.4 Weight Distribution Example


```
{
  "skills": {
    "distributed systems": 0.30,
    "python": 0.25,
    "system design": 0.20,
    "communication": 0.15,
    "problem solving": 0.10
  },
  "experience_level": "mid-senior",
  "total_weight": 1.0,
  "processing_confidence": 0.82
}
```


8 Candidate Profiling System

8.1 Profile Dimensions Definition

Five key dimensions evaluated throughout interview:

1. **Technical Depth**: Domain knowledge and expertise level
2. **JD Relevance**: Alignment with job requirements
3. **Clarity**: Communication effectiveness and structure
4. **Consistency**: Logical coherence across multiple answers
5. **Confidence**: Linguistic certainty and assertiveness

8.2 Incremental Update Framework

Profile updates implemented using Bayesian updating:

$$P(\theta|D) = \frac{P(D|\theta) \cdot P(\theta)}{P(D)}$$

where θ represents the candidate's true capability vector and D is observed evidence from answers.

8.3 Update Rules Implementation

After each answer, profiles updated using:

1. Compute answer-specific feature vector f_i
2. Calculate likelihood $P(f_i|\theta_{current})$
3. Apply Bayesian update with decay factor $\lambda = 0.95$
4. Store timestamped snapshot with uncertainty bounds

8.4 Decay Mechanism

Old evidence receives exponential decay to prioritize recent performance:

$$w_{old} = w_{initial} \cdot e^{-t/\tau}$$

where t is time since observation and τ is decay constant (default: 300 seconds).

8.5 Feature Extraction Methods

8.5.1 Technical Depth Features

- Technical terminology usage frequency compared to baseline
- Domain-specific concept accuracy validation
- Problem-solving methodology quality assessment
- Depth vs. breadth analysis in explanations
- Technology stack familiarity indicators

8.5.2 Clarity Features

- Sentence complexity metrics using Flesch-Kincaid readability
- Filler word frequency analysis (um, uh, like, etc.)
- Response structure organization evaluation
- Explanation coherence and logical flow assessment

9 Video Analysis Pipeline

9.1 Frame Extraction Strategy

Video processing implemented with efficient sampling:

- Sampling rate: 1 frame per second for analysis
- Face detection using Multi-task Cascaded Convolutional Networks
- 68-point facial landmark extraction for feature calculation
- GPU acceleration when available for batch processing

9.2 Behavioral Signal Detection

9.2.1 Stress Indicators

- Facial muscle tension patterns around eyes and mouth
- Eye contact frequency and duration measurement
- Head movement variance and speed analysis
- Speaking rhythm disruption detection
- Micro-expression analysis for emotional states

9.2.2 Engagement Metrics

- Gaze direction consistency estimation
- Nod frequency and timing in response to questions
- Responsive gesture pattern detection
- Attention span duration measurement
- Posture and position stability analysis

9.3 Signal Integration Protocol

Behavioral signals weighted as supporting evidence:

- Overall weight: 20% of total profile score
- Confidence adjustment factor: ± 0.1 max deviation
- Risk flag threshold: stress indicator > 0.7 triggers flag
- Engagement threshold: engagement < 0.4 reduces clarity score

10 Fusion and Verdict Engine

10.1 Feature Aggregation Method

All evidence sources combined using weighted linear combination:

$$S_{final} = \sum_{i=1}^{n} w_i \cdot s_i + \epsilon$$

where s_i are normalized scores, w_i are calibrated weights, and ϵ is noise term.

10.2 Evidence Weights

- Q/A quality scores: 40% weight
- JD alignment metrics: 30% weight
- Profile trajectory: 20% weight
- Behavioral signals: 10% weight

10.3 Uncertainty Quantification

Final verdict includes comprehensive uncertainty metrics:

- Epistemic uncertainty: Model confidence limits and parameters
- Aleatoric uncertainty: Data variability and measurement noise
- Total confidence: Combined uncertainty bounds using interval arithmetic
- Calibration: Reliability diagram validation against expert judgments

10.4 Verdict Generation Algorithm

10.4.1 Hire Probability Calculation

$$P_{hire} = \sigma\left(\sum_{i} w_i \cdot s_i - \beta\right)$$

where σ is sigmoid function, s_i are normalized scores, w_i are weights, and β is calibrated threshold.

10.4.2 Risk Flag Detection

Risk conditions trigger specific flags using threshold analysis:

- Inconsistency: Contradictory statements detected across answers
- Overclaiming: Unrealistic skill assertions compared to experience
- Communication issues: Clarity score below 0.5 consistently
- Behavioral concerns: Stress indicator above 0.7 for extended periods

10.4.3 Output Format Specification

Final verdict includes comprehensive evaluation data:

```
{
  "hire_probability": 0.63,
  "confidence": 0.58,
  "confidence_interval": [0.51, 0.75],
  "risk_flags": ["overgeneralization", "inconsistency"],
  "jd_alignment_score": 0.71,
  "final_score": 0.66,
  "evidence_summary": {
    "qa_quality": 0.68,
    "profile_stability": 0.72,
    "behavioral_consistency": 0.51,
    "technical_accuracy": 0.74
  },
  "processing_metadata": {
    "total_processing_time": 1247.5,
    "stages_completed": 8,
    "error_count": 0
  }
}
```

11 API Design and Data Access

11.1 Internal Processing APIs

Processing pipeline exposed through Django REST endpoints:

- `POST /api/processing/start`: Trigger pipeline for session
- `GET /api/processing/status/{session_id}`: Check pipeline progress
- `POST /api/processing/retry`: Restart failed processing stages
- `GET /api/processing/logs/{session_id}`: Access processing logs

11.2 Dashboard Data APIs

Read-only data access for dashboard consumption:

- `GET /api/session/{id}/script`: Conversation timeline data
- `GET /api/session/{id}/qa`: Question-answer pairs with metadata
- `GET /api/session/{id}/profile`: Profile evolution over time
- `GET /api/session/{id}/verdict`: Final evaluation results

11.3 Data Serialization Standards

All API responses include standardized metadata:

- Timestamp for data freshness indication
- Confidence intervals where applicable
- Links to related data entities using HATEOAS principles
- Metadata for data lineage and traceability
- Processing status and completion percentages

12 Implementation Sequence and Dependencies

12.1 Module Dependency Graph

Processing stages depend on specific prerequisites:

```
Audio Preprocessing (independent)
|-- ASR and Alignment (depends on preprocessing)
|   `-- Script Construction (depends on ASR)
|       `-- Q/A Extraction (depends on script)
|-- JD Processing (independent)
|   `-- Profiling (depends on Q/A and JD)
`-- Video Analysis (independent)
    `-- Verdict Generation (depends on all above)
```

12.2 Testing Strategy

Comprehensive testing approach implemented:

- Unit tests for individual processing functions (pytest framework)
- Integration tests for pipeline stage interactions
- End-to-end tests with complete sample interviews
- Performance tests with large datasets (benchmarking)
- Validation tests against expert-labeled ground truth

12.3 Performance Optimization

- Whisper model loading with memory optimization
- Database query optimization using indexes and connection pooling
- Caching for repeated computations using Redis
- Resource monitoring during processing for bottleneck identification
- Parallel processing where pipeline independence allows

13 Quality Assurance and Validation

13.1 Validation Criteria

Each processing stage includes comprehensive validation:

- ASR confidence > 0.7 for word inclusion
- Q/A pair confidence > 0.6 for analysis inclusion
- Profile dimension bounds enforced: [0.0, 1.0] range
- Final verdict uncertainty < 0.3 for high-confidence decisions
- Consistency checks across processing stages

13.2 Error Handling Strategy

- Graceful degradation for partial processing failures
- Automatic retry mechanisms for transient errors with exponential backoff
- Manual override capabilities for edge cases
- Comprehensive logging with structured error information
- Recovery procedures for common failure scenarios

13.3 Data Integrity Measures

- Input validation for all data processing stages
- Output validation to ensure data format consistency
- Checksum verification for large file processing
- Atomic operations for database transactions
- Rollback capabilities for failed processing stages

14 Conclusion

The backend processing pipeline implements a comprehensive and explainable approach to interview intelligence analysis. By using deterministic algorithms with explicit uncertainty quantification, the system provides defensible evaluations suitable for academic and professional contexts. The modular architecture enables independent development and testing of components while maintaining clear data flow and dependency management.

The offline processing philosophy ensures accuracy and reproducibility, critical for evaluation systems where decision transparency is paramount. Bayesian updating for candidate profiles and comprehensive uncertainty quantification provide statistically sound foundations for evaluation outcomes. The system successfully balances technical complexity with operational reliability, establishing a robust foundation for interview intelligence applications.