Requirements Extractor - Design Document

Version: 1.0
Date: November 2024
Author: Requirements Extraction System Team

Table of Contents

Executive Summary

The Requirements Extractor is a web-based application designed to automatically extract and structure requirements from Microsoft Teams meeting transcripts and video recordings. The system uses AI-powered natural language processing to identify functional requirements, non-functional requirements, business rules, action items, decisions, and stakeholders from meeting discussions.

Key Benefits

- **Automated Extraction**: Reduces manual effort in documenting requirements
- **Structured Output**: Generates organized, searchable requirement documents
- **Multiple Input Formats**: Supports text transcripts, VTT files, JSON, and video files
- **Local Processing**: Option to process entirely locally without API dependencies
- **Incremental Processing**: Real-time results as data is processed

---

System Overview

Purpose

The Requirements Extractor automates the extraction of structured requirements from meeting transcripts, enabling teams to:

- Quickly document requirements from meetings
- Maintain consistent requirement documentation
- Track action items and decisions
- Identify stakeholders and their interests

Target Users

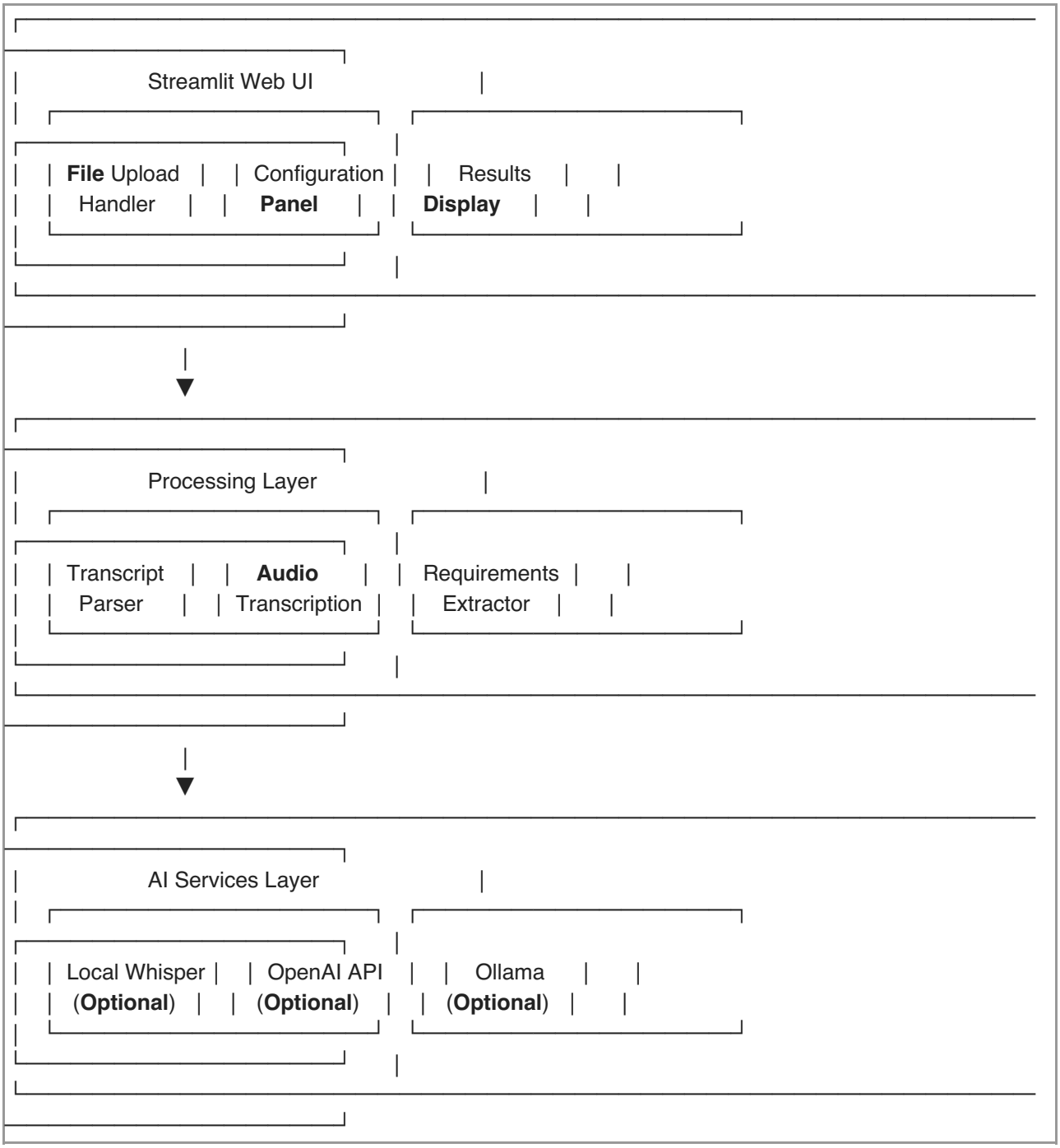
- Business Analysts
- Project Managers
- Product Managers
- Development Teams
- Stakeholders involved in requirement gathering

Core Functionality

1. **Input Processing**: Accepts multiple file formats (text, VTT, JSON, video)
2. **Transcription**: Converts video/audio to text using Whisper (local or API)
3. **Parsing**: Extracts structured data from transcripts
4. **AI Extraction**: Uses LLM to identify and categorize requirements
5. **Report Generation**: Creates formatted reports in Markdown and JSON

Architecture

High-Level Architecture



Component Architecture

1. Frontend Layer (Streamlit)

- **File Upload Component**: Handles file uploads with size validation
- **Configuration Panel**: User settings (API keys, model selection)
- **Progress Tracking**: Real-time progress bars and status updates
- **Results Display**: Tabbed interface for different requirement types
- **Export Functions**: Download as Markdown or JSON

2. Processing Layer

- **TranscriptParser**: Parses various transcript formats
- **VideoProcessor**: Extracts audio from video files

- **AudioTranscriber**: Converts audio to text
- **RequirementsExtractor**: AI-powered requirement extraction

3. AI Services Layer

- **Local Whisper**: On-device speech-to-text (no API key)
- **OpenAI Whisper API**: Cloud-based transcription
- **OpenAI GPT Models**: Cloud-based requirement extraction
- **Ollama**: Local LLM for requirement extraction

Features

Core Features

1. Multi-Format Input Support

- **Text Files (.txt)**: Plain text transcripts
- **WebVTT Files (.vtt)**: Timestamped transcripts
- **JSON Files (.json)**: Structured transcript data
- **Video Files**: MP4, MOV, AVI, MKV, WEBM
    - Automatic audio extraction
    - Large file chunking support

2. Transcription Capabilities

- **Local Whisper**: Free, no API key required
    - Model sizes: base, small, medium, large
    - Runs entirely on local machine
- **OpenAI Whisper API**: Fast, cloud-based
    - Automatic chunking for large files
    - High accuracy

3. Requirements Extraction

- **Functional Requirements**: Features and capabilities
- **Non-Functional Requirements**: Performance, security, usability
- **Business Rules**: Constraints and business logic
- **Action Items**: Tasks with owners and deadlines
- **Decisions**: Key decisions with rationale
- **Stakeholders**: People and their roles/interests

4. Incremental Processing

- Real-time progress updates
- Partial results display
- Chunk-based processing for large files
- Progress persistence

5. Export Options

- **Markdown**: Human-readable formatted report
- **JSON**: Machine-readable structured data

Advanced Features

- **Chunking**: Automatic splitting of large files
- **Deduplication**: Removes duplicate requirements
- **Progress Tracking**: Visual progress bars and status messages
- **Error Handling**: Comprehensive error messages with solutions
- **Local Processing**: Complete offline capability with Ollama + Whisper

---

Technical Stack

Frontend

- **Streamlit**: Web framework for Python applications
- **HTML/CSS**: Custom styling for UI components
- **JavaScript**: Minimal (handled by Streamlit)

Backend

- **Python 3.8+**: Core programming language
- **Streamlit**: Web application framework

AI/ML Libraries

- **openai-whisper**: Local speech-to-text
- **openai**: OpenAI API client
- **requests**: HTTP client for Ollama API

Media Processing

- **moviepy**: Video/audio processing
- **pydub**: Audio manipulation
- **imageio-ffmpeg**: FFmpeg binary bundling

Data Processing

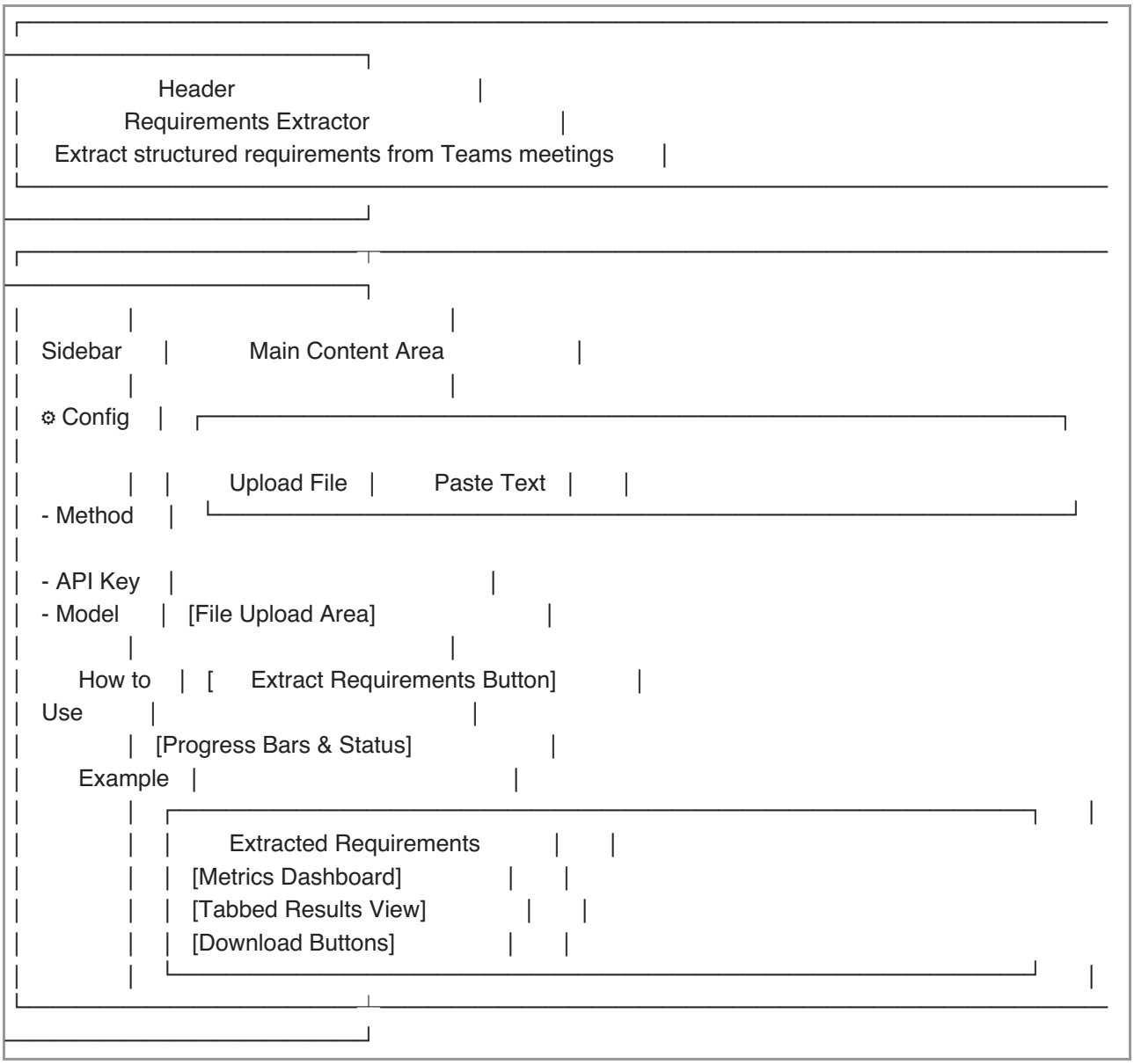
- **json**: JSON parsing and generation
- **pandas**: Data manipulation (for action items table)
- **re**: Regular expressions for text parsing

Dependencies

```
streamlit>=1.28.0
openai>=1.0.0
openai-whisper
moviepy
pydub
pandas
requests>=2.31.0
numpy<2.0
```

User Interface Design

Layout Structure



UI Components

1. Header

- Application title
- Subtitle describing purpose
- Styled with custom CSS

2. Sidebar

- **Configuration Section**:
    - Transcription method selection (Local/OpenAI)
    - Requirements extraction method (Ollama/OpenAI)
    - API key input (when needed)
    - Model selection
- **Instructions Section**:
    - How to use guide
    - Supported formats list
    - Example format

3. Main Content Area

- **File Upload Tab**:
    - Drag-and-drop file uploader
    - File preview for text files
    - File size display
- **Paste Text Tab**:
    - Large text area for manual input
    - Format hints
- **Extract Button**:
    - Primary action button
    - Disabled when no input available
- **Progress Display**:
    - Progress bars
    - Status messages
    - Partial results
- **Results Display**:
    - Summary metrics (6 columns)
    - Tabbed interface for different requirement types
    - Expandable sections for details
    - Download buttons

Color Scheme & Styling

- **Primary Color**: Blue (#1f77b4)
- **Success**: Green (#28a745)
- **Warning**: Yellow (#ffc107)

- **Error**: Red (#dc3545)
- **Info**: Light Blue (#17a2b8)

Responsive Design

- Sidebar collapses on smaller screens
- Columns adjust for mobile devices
- Tables scroll horizontally when needed

Data Flow

Processing Flow

```
1. User Input
   ├── File Upload → Saved to temp file
   └── Text Paste → Saved to temp file

2. File Type Detection
   ├── Video File → Extract Audio → Transcribe
   ├── Text File → Parse Transcript
   ├── VTT File → Parse with Timestamps
   └── JSON File → Parse Structured Data

3. Transcript Parsing
   └── Convert to Message Format
    {speaker, text, timestamp}

4. Transcription (if video/audio)
   ├── Local Whisper → Text
   └── OpenAI API → Text

5. Requirements Extraction
   ├── Chunk Messages (if large)
   ├── Process Each Chunk
   |    ├── Ollama API → Requirements
   |    └── OpenAI API → Requirements
   └── Merge & Deduplicate

6. Report Generation
   ├── Format as Markdown
   └── Format as JSON

7. Display & Export
   ├── Show in UI
   └── Download Options
```

Data Structures

Message Format

```
{
  "speaker": "John Doe",
  "text": "We need to implement user authentication",
  "timestamp": "00:05:23"
}
```

Requirements Format

```
{
  "functional_requirements": [
    {
      "id": "FR-001",
      "description": "User authentication system",
      "priority": "High",
      "speaker": "John Doe",
      "context": "Discussed in security section"
    }
  ],
  "non_functional_requirements": [...],
  "business_rules": [...],
  "action_items": [...],
  "decisions": [...],
  "stakeholders": [...]
}
```

API Integration

OpenAI API

Whisper API (Transcription)

- **Endpoint**: https://api.openai.com/v1/audio/transcriptions
- **Method**: POST
- **Authentication**: Bearer token (API key)
- **Limitations**: 25MB file size limit
- **Chunking**: Automatic for larger files

GPT API (Requirements Extraction)

- **Endpoint**: https://api.openai.com/v1/chat/completions
- **Method**: POST
- **Models**: gpt-4o-mini, gpt-4o, gpt-3.5-turbo
- **Response Format**: JSON object
- **Temperature**: 0.3 (for consistency)

Ollama API

Generate Endpoint

- **URL**: http://localhost:11434/api/generate

- **Method**: POST
- **Models**: llama3.2, mistral, codellama, etc.
- **Streaming**: Supported but disabled for simplicity
- **Timeout**: 300 seconds

Tags Endpoint (Health Check)

- **URL**: http://localhost:11434/api/tags
- **Method**: GET
- **Purpose**: Verify Ollama is running

Local Whisper

- **Library**: openai-whisper
- **Models**: base, small, medium, large
- **Processing**: On-device, no network required
- **Dependencies**: FFmpeg, PyTorch

Deployment

Local Deployment

Prerequisites

- Python 3.8 or higher
- FFmpeg (for video processing)
- Ollama (optional, for local LLM)

Installation Steps

1. **Clone/Download Repository**

   ```
   cd Requirements_from_calls
   ```

2. **Install Dependencies**

   ```
   pip install -r requirements.txt
   ```

3. **Install FFmpeg** (macOS)

   ```
   brew install ffmpeg
   ```

4. **Configure Streamlit** (optional)
   - Edit .streamlit/config.toml for upload limits
   - Set maxUploadSize as needed

5. **Start Application**

   ```
   python3 -m streamlit run app.py --server.port 8501
   ```

6. **Access Application**

- Open browser to http://localhost:8501

Cloud Deployment Options

Streamlit Cloud

- **Platform**: streamlit.io
- **Requirements**: GitHub repository
- **Limitations**: File size limits, timeout constraints
- **Cost**: Free tier available

Docker Deployment

```
FROM python:3.9-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 8501
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

AWS/GCP/Azure

- Use container services (ECS, Cloud Run, Container Instances)
- Configure environment variables for API keys
- Set up persistent storage for large files
- Configure auto-scaling if needed

Environment Variables

```
OPENAI_API_KEY=sk-...          # Optional: For OpenAI API
STREAMLIT_SERVER_PORT=8501     # Optional: Port configuration
STREAMLIT_SERVER_ADDRESS=0.0.0.0  # Optional: Network binding
```

Security Considerations

API Key Management

- **Storage**: Never commit API keys to version control
- **Input**: Masked password fields in UI
- **Environment Variables**: Preferred method for production
- **Validation**: API key format validation before use

File Handling

- **Temporary Files**: All files stored in temp directory
- **Cleanup**: Automatic cleanup after processing
- **Size Limits**: Configurable upload limits

- **Validation**: File type and format validation

Data Privacy

- **Local Processing**: Option to process entirely locally
- **No Data Storage**: Files processed and immediately deleted
- **API Usage**: Data sent to OpenAI/Ollama as per their privacy policies
- **User Control**: Users choose between local and cloud processing

Network Security

- **HTTPS**: Recommended for production deployments
- **CORS**: Configured appropriately for Streamlit
- **Firewall**: Restrict access to local deployments

Performance & Scalability

Performance Characteristics

File Size Handling

- **Small Files (<25MB)**: Direct processing
- **Large Files**: Automatic chunking
- **Video Files**: Audio extraction before processing

Processing Times

- **Transcription**: ~1x video length (local Whisper)
- **Transcription**: ~0.1x video length (OpenAI API)
- **Extraction**: ~5-30 seconds per chunk (depends on LLM)

Memory Usage

- **Base Application**: ~200-500 MB
- **Whisper Model**: 150 MB - 3 GB (depending on model size)
- **Video Processing**: Temporary storage = video file size

Optimization Strategies

1. **Chunking**: Process large files in smaller chunks
2. **Caching**: Cache transcriptions for repeated processing
3. **Parallel Processing**: Process multiple chunks concurrently (future enhancement)
4. **Model Selection**: Use smaller models for faster processing

Scalability Considerations

Current Limitations

- Single-threaded processing
- In-memory data storage
- No database for persistence

Future Scalability Options

- **Queue System**: Use Celery/RQ for background processing
- **Database**: Store results in database for retrieval
- **Caching**: Redis for transcript caching
- **Load Balancing**: Multiple Streamlit instances behind load balancer

---

Future Enhancements

Short-Term (v1.1)

1. **Batch Processing**
    - Process multiple files simultaneously
    - Queue management
    - Progress tracking per file
2. **Export Enhancements**
    - PDF export option
    - Excel/CSV export for action items
    - Custom report templates
3. **UI Improvements**
    - Dark mode
    - Customizable themes
    - Better mobile responsiveness
4. **Transcript Editing**
    - Edit transcripts before extraction
    - Speaker name correction
    - Timestamp adjustment

Medium-Term (v1.2)

1. **Database Integration**
    - Store requirements in database
    - Search and filter capabilities
    - Version history
2. **Collaboration Features**
    - Share requirements with team
    - Comments and annotations
    - Approval workflows
3. **Advanced Extraction**
    - Custom requirement templates
    - Domain-specific extraction
    - Multi-language support

4. **Integration**
    - Jira integration
    - Confluence export
    - Slack notifications

Long-Term (v2.0)

1. **AI Enhancements**
    - Fine-tuned models for specific domains
    - Requirement validation
    - Conflict detection
2. **Analytics**
    - Requirement trends
    - Meeting insights
    - Stakeholder analysis
3. **Enterprise Features**
    - SSO integration
    - Role-based access control
    - Audit logs
    - Compliance reporting

Appendix

A. File Format Specifications

Text Format

```
Speaker Name: Message text here
Another Speaker: Another message
[Timestamp] Speaker: Message with timestamp
```

VTT Format

```
WEBVTT

00:00:00.000 --> 00:00:05.000
Speaker Name: First message

00:00:05.000 --> 00:00:10.000
Another Speaker: Second message
```

JSON Format

```
{
  "messages": [
    {
      "speaker": "John Doe",
      "text": "Message text",
      "timestamp": "00:05:23"
    }
  ]
}
```

B. Error Codes & Messages

Error	Cause	Solution
File too large	Exceeds upload limit	Increase limit or chunk file
Invalid API key	Wrong/expired key	Verify and update API key
Ollama not running	Service not started	Start Ollama service
FFmpeg not found	Missing dependency	Install FFmpeg
NumPy compatibility	Version mismatch	Install numpy<2.0

C. Configuration Files

.streamlit/config.toml

```
[server]
maxUploadSize = 1073741824  # 1GB
maxMessageSize = 1073741824
port = 8501
```

requirements.txt

```
streamlit>=1.28.0
openai>=1.0.0
openai-whisper
moviepy
pydub
pandas
requests>=2.31.0
numpy<2.0
```

Document History

Version	Date	Author	Changes
1.0	Nov 2024	System Team	Initial design document

Contact & Support

For questions, issues, or contributions:

- **Repository**: [GitHub Repository URL]
- **Issues**: [GitHub Issues URL]
- **Documentation**: [Documentation URL]

End of Design Document