Requirements Architecture and Engineering Specifications for the Media-to-Text Extraction System

The rapid proliferation of unstructured data in modern enterprise environments has necessitated the development of sophisticated automated systems capable of converting multifaceted media into searchable, actionable textual intelligence. In an era where organizations grapple with vast repositories of images, complex document formats, and voluminous video archives, the ability to derive structural meaning from these assets is no longer a luxury but a strategic imperative. The Media-to-Text Extraction System, envisioned as a Python-based infrastructure, represents a comprehensive response to this challenge. This initiative, spearheaded by project leadership under Somnath Shendage, aims to establish a robust, deployment-ready pipeline that bridges the divide between unstructured media and downstream analytical or artificial intelligence processing frameworks.[1]

Strategic Framework and Executive Project Overview

The core objective of the Media-to-Text Extraction System is to automate the transcription and recognition of textual content across three primary media vectors: static images, portable document formats (PDFs), and video recordings. This technical transformation is facilitated through a Python-centric ecosystem, leveraging state-of-the-art libraries such as Pytesseract for optical character recognition, pdfplumber for document parsing, and OpenAI's Whisper for high-fidelity audio-to-text conversion.[4] The overarching vision is to provide a seamless, web-accessible interface that allows diverse stakeholders to ingest media files and receive structured textual outputs suitable for reporting, archival, or integration into generative AI workflows.[7]

A successful implementation hinges on the alignment of technical capabilities with business goals. The primary goals of this project include providing an intuitive upload interface, ensuring high accuracy for both text-based and scanned PDFs, achieving reliable speech-to-text transcription even in suboptimal acoustic conditions, and maintaining a scalable architecture that can be deployed across internet-accessible cloud environments.[10] The system's utility is further enhanced by its ability to generate downloadable outputs, ensuring that users can transition from raw media to editable text with minimal friction.[1]

Multi-Dimensional Stakeholder Analysis and Responsibility Mapping

The success of an artificial intelligence-driven media project depends on the coordinated efforts of a multifaceted stakeholder group. Each role brings unique perspectives that shape the system's functional and non-functional requirements. The interaction between these roles creates a feedback loop that ensures the final product meets both the technical constraints and the business drivers.[14]

In the context of the Media-to-Text system, the Business Analyst serves as the primary architect of requirements, extracting structured insights from unstructured project goals. This role requires an understanding of how data point influence another; for example, a Project Owner's demand for high accuracy in video transcription directly influences the DevOps Engineer's need for larger GPU allocations and the Lead Developer's choice of the Whisper "Large" model over "Base".[29]

Functional Requirement Architecture: The Extraction Modules

The system's functional architecture is partitioned into four primary modules, each addressing a specific stage or type of media processing. These modules must be designed with modularity in mind, allowing for independent updates to the underlying recognition engines as technological standards evolve.[2]

Image-to-Text Module Engineering

The Image-to-Text module is primarily driven by Optical Character Recognition (OCR) technology. While the fundamental task is to recognize characters in an image, professional-grade systems require extensive preprocessing to achieve acceptable accuracy thresholds.[25] Preprocessing includes deskewing to fix alignment issues, noise removal to enhance clarity, and binarization to maximize the contrast between text and background.[25] The choice of Pytesseract as the primary engine provides a robust, open-source framework, though its performance is heavily dependent on Page Segmentation Modes (PSM) that define how the engine interprets the document layout.[6]

PSM Mode	Theoretical Application	Practical Impact on Extraction

PSM 3	Fully automatic page segmentation (no OSD).	Best for general, unstructured documents.[33]
PSM 6	Assume a single uniform block of text.	Ideal for book pages or single-column reports.[33]
PSM 11	Sparse text; find as much text as possible.	Useful for labels, sparse ads, or complex headers.[33]
PSM 12	Sparse text with columnar format.	Specifically designed for newspaper or magazine layouts.[33]

The system must support a variety of image formats, including .jpg, .png, and .tiff. For high-precision use cases, the module should aim for a Character Error Rate (CER) below 2% for clean printed scans. However, the system's effectiveness is often challenged by degraded scans, low resolution (below 300 DPI), or complex handwriting, which remains a persistent limitation for traditional OCR engines like Tesseract.[5]

PDF-to-Text Module Optimization

PDF documents represent a dual challenge because they can contain both natively encoded digital text and scanned images of text. The PDF-to-Text module must implement a logic-gate to determine the appropriate extraction method. For digital PDFs, the system should leverage pdfplumber or PDFMiner to extract selectable text with high fidelity and layout preservation.[6] If no selectable text is detected, the system must trigger a fallback mechanism that converts PDF pages into high-resolution images for OCR processing.[5]

This "Two-Step" process is computationally expensive. For instance, converting a single PDF page at 300 DPI for OCR can consume up to 25MB of RAM, making multi-page document processing a significant memory bottleneck on standard cloud dynos.[5] Therefore, the module requirements must specify a sequential or chunked processing approach to prevent memory overflow errors. The integration of layout-aware engines is also essential for extracting structured data from tables or multi-column forms, ensuring that the reading order is maintained during the transition from visual to textual formats.[5]

Video-to-Text Module and ASR Pipeline

The Video-to-Text module represents the most complex pipeline in the system. It involves

demuxing the video file to isolate the audio stream, followed by Automatic Speech

Recognition (ASR). OpenAI's Whisper model has emerged as the industry benchmark for this task due to its ability to handle multiple languages, accents, and background noise.[4] The module should support various models—Tiny, Base, Small, Medium, and Large—allowing the user to balance the trade-off between transcription speed and word accuracy.[30]

A critical feature of the video module is its ability to handle long-form content. The audio must be segmented into manageable chunks (e.g., 30-second intervals) to fit within the receptive window of transformer models like Whisper.[4] Furthermore, the system should provide time-aligned transcripts, allowing users to map specific text segments back to their original timestamp in the video. In more advanced configurations, the module may also trigger OCR on specific video frames—such as when a scene change detection algorithm identifies a new presentation slide—to capture visual text that may not be spoken.[13]

Web Interface and User Interaction Layer

The web interface serves as the gateway for all media ingestion and interaction. Developed using Python frameworks such as FastAPI or Flask, the interface must prioritize an intuitive user experience.[13] Key functionalities include multi-file upload support, real-time status indicators for ongoing extraction tasks, and a preview window for reviewing extracted text. For professional users, the interface must allow for locale selection to improve recognition accuracy and provide a mechanism for users to correct "hallucinations" or misrecognitions in the transcript.[23]

The interaction layer also encompasses the output delivery. The system must provide options to download the extracted text in various formats, such as .txt, .docx, or .json, which includes metadata like confidence scores and timestamps.[1] From a business analysis perspective, the interface is not just a visual layer but a feedback loop; by capturing user corrections, the system can provide data for future model fine-tuning.[34]

Non-Functional Requirements (NFRs) and Quality Attributes

NFRs define the system's operational constraints and quality thresholds. For a media extraction system, these requirements are often more critical than functional ones, as they determine the system's reliability in a production environment.[24]

Performance and Latency Metrics

Performance requirements are defined by the speed and responsiveness of the system under varying workloads. In speech-to-text, latency is the time elapsed between audio input and the appearance of the transcript. For real-time applications, this should be under 300ms.[29] For

batch processing, however, throughput—the total volume of media processed per unit of time—is the more relevant metric.

Metric	Business Requirement	Technical Significance
Real-Time Factor (RTF)	< 0.6 for Whisper Small	Indicates the system processes one minute of audio in less than 36 seconds.[29]
OCR Throughput	> 10 pages/minute	Measures efficiency in high-volume document digitization.[5]
API Response Time	< 2 seconds for log-in	Critical for maintaining user engagement in the web interface.[43]
VRAM Allocation	≥ 10 GB for Whisper Large	Dictates the choice of GPU hardware to avoid processing bottlenecks.[30]

Security, Privacy, and Redaction

Security requirements focus on data protection and regulatory compliance. The system must implement encryption standards such as TLS 1.3 for data in transit and AES-256 for data at rest.[28] A major requirement for this project is the identification and redaction of Personally Identifiable Information (PII) within transcripts. This includes names, addresses, social security numbers, and financial details.[27]

The implementation of PII redaction is a legal necessity under GDPR and HIPAA. The system should ideally offer both "redaction" (replacing sensitive text with placeholders like [PII]) and "identification" (flagging sensitive text for manual review).[46] However, stakeholders must be aware that ML-driven redaction is probabilistic; therefore, the NFR should specify a human-in-the-loop validation process for high-stakes domains like healthcare or law.[34]

Scalability and Resource Elasticity

The system must scale gracefully as the volume of media uploads increases. This involves implementing horizontal scaling, where the system automatically spins up additional worker instances as the processing queue grows.[41] For cloud-native deployments, the system should

support auto-scaling groups triggered by CPU utilization (e.g., >70%) or memory pressure.[26] Resource elasticity is particularly important for video processing, which requires significant GPU cycles compared to simple image OCR.[37]

Technical Architecture and Data Pipeline Orchestration

The system's technical foundation is a modular data pipeline that manages the flow of media from ingestion to final storage. An effective architecture ensures that each stage of the pipeline—collection, processing, integration, and storage—is decoupled and scalable.[2]

Pipeline Design Patterns: ETL vs. ELT

The traditional Extract, Transform, Load (ETL) approach is well-suited for structured data but can create bottlenecks in media processing because transformation (OCR/ASR) is the most time-consuming step.[48] Modern architectures increasingly favor Extract, Load, Transform (ELT), where raw media is immediately loaded into a data lake (like AWS S3 or a local object store) and then processed by dedicated microservices as resources become available.[13]

A "Zero ETL" approach is also emerging, which allows for point-to-point data movement without traditional staging steps, though this often requires the transactional database and processing engine to reside on the same cloud provider.[48] For this project, a hybrid streaming/batch pipeline is recommended: streaming for small images and real-time commands, and batch for multi-page PDFs and long-form video.[12]

Decoupled Microservices and Communication

By adopting a microservices pattern, the system can independently scale different components based on workload. The "Pipeline Manager" serves as the central orchestrator, using a message bus (such as Kafka or RabbitMQ) to coordinate tasks between specialized services.[13]

Component	Role in the Pipeline	Key Technologies
Ingestion Service	Consumes files from the UI; handles validation and initial storage.[12]	FastAPI, AWS S3
Media Demuxer	Extracts audio channels from video files; performs image resizing.[38]	MoviePy, ffmpeg

Transcription Service	Executes ASR using Whisper models; manages audio chunking.[4]	PyTorch, Whisper
OCR Service	Performs text recognition on images and PDF pages; handles layout analysis.[32]	Pytesseract, PaddleOCR
Redaction Engine	Scans text for PII entities and applies masking rules.[27]	Presidio, AWS Transcribe
Storage Engine	Stores raw media, time-aligned transcripts, and structured metadata.[13]	MongoDB, Vector DBs

This decoupled design is a strategic choice; it allows the system to remain adaptable. If a video has no speech, the system can skip the ASR service entirely. If the visual probe detects significant on-screen text, the OCR path is activated. This conditional logic maximizes efficiency and reduces operational costs.[13]

Infrastructure, Environment, and Deployment Constraints

Deploying a compute-heavy Python application requires navigating specific infrastructure limits, particularly regarding memory and binary dependencies.[21]

Cloud Deployment on Heroku and Render

Heroku represents a common deployment target for Python APIs, but it imposes strict "slug" and memory limits. A standard Heroku dyno provides 512MB to 1GB of RAM, while a Tesseract-based application may require significantly more during PDF-to-image conversion.[5] To overcome these limits, developers must utilize custom buildpacks to include the Tesseract binary and its associated libraries (libtiff, libjpeg).[52]

The configuration process involves creating an Aptfile to specify the Debian packages needed (e.g., tesseract-ocr, tesseract-ocr-eng) and setting environment variables like TESSDATA_PREFIX to point to the training data directory.[21] Furthermore, the Procfile must be configured to use a production-grade web server like Gunicorn with Uvicorn workers to handle the asynchronous nature of media extraction tasks.[21]

GPU Acceleration and VRAM Requirements

While the system can run on CPUs, performance for larger models like Whisper "Medium" or "Large" will be unacceptably slow without GPU acceleration.[30] Modern transformers require a CUDA-enabled GPU with sufficient VRAM to load the model weights and handle the attention mechanisms used in ASR.[4]

GPU Model	VRAM	Whisper Support	Performance Focus
NVIDIA A100	40-80 GB	All Models	Large-scale batch deployments.[49]
RTX 4090	24 GB	All Models	High-end consumer; real-time transcription.[49]
RTX 3060 Ti	8 GB	Medium / Large	Best price-to-performance for medium models.[49]
GTX 1650	4 GB	Tiny / Base	Entry-level; limited to smallest models.[49]

For cost-effective scaling, the system can utilize quantized models (INT4 or INT8). Quantization can reduce model size by up to 45% and latency by 19% while maintaining approximately 99% of the original model's accuracy, making high-performance transcription feasible on more modest hardware.[55]

API and Integration Requirements

The system must provide a robust RESTful API to allow for integration with external business systems. This API serves as the primary contract between the extraction engine and the client applications.[11]

Key API Endpoints and Payloads

- POST /upload: Accepts a multipart/form-data payload containing the media file and configuration parameters (e.g., target language, model size, PII redaction toggle).
- GET /status/{task_id}: Returns the current state of the processing job (e.g., queued, processing, completed, failed).
- GET /result/{task_id}: Retrieves the final textual output along with metadata such as confidence scores and timestamps.
- POST /webhook: Allows the system to push notifications to external listeners once a task is finished, reducing the need for constant polling.[12]

Custom Vocabularies and Locale Management

To improve accuracy in domain-specific contexts (e.g., medical, legal, or technical), the API must support "Custom Vocabularies." This allows users to provide a phrasal list of terms that are likely to appear in the media, such as product names or specialized jargon, which the ASR or OCR engine might otherwise misidentify.[29] Additionally, the API must handle "Input Locales" to ensure that the engine uses the correct language model for recognition. Mismatched locales are a primary cause of high Word Error Rates.[39]

Agile Execution: User Stories and Acceptance Criteria

User stories provide a narrative framework for defining functionality from the perspective of the end user. They ensure that the technical team understands the "why" behind each feature.[22]

Representative User Stories

- **As a Marketing Manager**, I want to upload 100 customer testimonial videos at once, so that I can automatically generate a sentiment analysis report from the transcripts.[9]
- **As a Legal Researcher**, I want to search through a digitized archive of scanned court documents for specific case names, so that I can find precedents without manually scanning thousands of pages.[5]
- **As a Healthcare Administrator**, I want the system to redact patient names and dates of birth from dictated clinical notes, so that we remain HIPAA compliant while using cloud-based transcription.[27]
- **As a Developer**, I want an API endpoint that returns JSON-formatted transcripts with word-level confidence scores, so that I can build custom validation interfaces for our users.[25]

Detailed Acceptance Criteria (AC)

Acceptance Criteria define the boundaries of a user story and provide the basis for testing. They must be binary (Pass/Fail) and measurable.[59]

1. **Given** a scanned PDF file with no selectable text, **When** the file is uploaded, **Then** the

system must successfully perform OCR on every page and return a searchable text file.[5]
2. **Given** an audio file with background noise, **When** transcribed with Whisper Large, **Then** the Word Error Rate must not exceed 10%.[29]
3. **Given** a video file, **When** the transcription is complete, **Then** the text must be returned with timestamps accurate to within 500ms of the spoken word.[38]
4. **Given** a request for PII redaction, **When** the transcript is generated, **Then** all detected physical addresses must be replaced with the tag ``.[46]

Future Horizons and System Enhancements

The evolution of media extraction is moving beyond simple "transcription" toward "intelligent content understanding." Future enhancements should focus on adding layers of semantic intelligence to the raw text output.[9]

Semantic Search and Vector Databases

A primary enhancement is the integration of Semantic Search. Traditional keyword search fails when users use synonyms or natural language queries. By converting extracted text into vector embeddings and storing them in a specialized vector database (such as Chroma, Pinecone, or Weaviate), the system can support "context-aware" searches.[50] This allows users to find content based on "meaning" rather than just character matching.[63]

Retrieval-Augmented Generation (RAG) and Agentic AI

By combining the extracted text with Large Language Models (LLMs) through a RAG pipeline, the system can answer complex questions about the media. For example, a user could ask, "What were the three main conclusions mentioned in the 45-minute boardroom video?" and the system would retrieve the relevant transcript segments and synthesize an accurate answer.[63] Furthermore, Agentic AI could automate follow-up actions, such as automatically drafting an email summary of a meeting and sending it to all participants.[9]

Blockchain for Document Verification

As digital manipulation of media becomes more sophisticated, the "provenance" of extracted text will become a critical requirement. Integrating blockchain technology could allow the system to create a tamper-proof link between the original media and the extracted text, ensuring that the digitized record has not been altered since its creation.[66] This is particularly relevant for the legal and financial sectors, where document integrity is paramount.

Assumptions, Constraints, and Risk Mitigation

Defining the assumptions and constraints is essential for managing stakeholder expectations and identifying potential project failures early.[67]

Key Project Assumptions

- The system assumes that input media files are in supported formats and are not corrupted.
- It is assumed that the cloud environment (Heroku or AWS) has sufficient internet bandwidth to handle large file uploads without timing out.
- The system assumes that users will provide correctly labeled locales for multilingual media to ensure optimal model selection.[39]

Technical and Operational Constraints

- **Memory Constraint**: Standard Heroku slugs are limited to 500MB, which may restrict the number of language models or binary libraries that can be bundled.[21]
- **Hardware Constraint**: Real-time video transcription is only feasible with GPU support; CPU-based transcription will be limited to asynchronous, background tasks.[36]
- **Accuracy Constraint**: OCR accuracy for handwriting is not guaranteed and will likely fall below 80% without specialized model training.[5]

Risk Assessment Matrix

A proactive risk analysis identifies threats to the project's success and defines strategies to reduce their likelihood or impact.[67]

Risk Event	Likelihood	Impact	Mitigation Strategy
Model "hallucinations" in transcripts	Moderate	Moderate	Implement confidence scores; display "uncertain" words to the user for verification.[39]
Privacy breach due to failed PII redaction	Low	Severe	Use a second-pass LLM-based redaction check; ensure human-in-the-loop review.[27]
System crashes during large file	High	Moderate	Implement robust chunking and a "retry" mechanism

processing			for failed tasks.[5]
High cost of GPU resources for video	Moderate	High	Use quantized models (INT8/INT4) to reduce compute requirements.[36]
Incompatible library updates (e.g., pdfplumber/pdfminer)	Low	Moderate	Use pinned versions in requirements.txt to ensure environmental stability.[35]

Project Deliverables and Implementation Roadmap

The final phase of the requirement gathering process is the definition of concrete deliverables. These artifacts ensure that the project is not just a theoretical model but a deployment-ready solution.[1]

1. **Complete Source Code**: A well-documented Python codebase including modules for Image, PDF, and Video extraction.
2. **Web Interface**: A fully functional web-based front-end for media ingestion and result visualization.
3. **Deployment Scripts**: Dockerfiles, Procfile, Aptfile, and environment configuration for cloud deployment (Heroku/AWS).
4. **API Documentation**: A comprehensive guide to the system's REST endpoints, including request/response schemas.
5. **Benchmarking Report**: A detailed analysis of CER/WER metrics across a sample set of the organization's media files.
6. **Security & Compliance Audit**: Documentation of encryption standards, data handling policies, and PII redaction effectiveness.

The Media-to-Text Extraction System is a pivotal technological step toward unlocking the value of unstructured media. By meticulously defining the functional requirements of each module, establishing rigorous non-functional quality attributes, and architecting a scalable, decoupled data pipeline, the project team—led by Somnath Shendage—can deliver a system that is both technically sophisticated and business-aligned. The inclusion of forward-looking features like semantic search and automated redaction ensures that the system will remain relevant as the broader artificial intelligence landscape continues to evolve.[1] Through this disciplined approach to requirement gathering, the organization transitions from merely

"having" data to actively "understanding" it, paving the way for a more intelligent, data-driven future.