Problem Statement 9: Document Intelligence and Processing Platform

GenAI Hackathon 2025 - AI-Powered Document Processing and Intelligence System

Problem Overview

Develop an AI-powered document intelligence platform that can automatically process, analyze, and extract insights from various document types including PDFs, Word documents, images, scanned documents, forms, contracts, invoices, and unstructured text. The platform should provide intelligent document classification, content extraction, entity recognition, sentiment analysis, summarization, and automated workflow processing.

Key Requirements

Core AI/ML Capabilities

- **Multi-format Document Processing**: Support for PDF, DOCX, images (JPG, PNG), scanned documents, forms, and text files
- **Optical Character Recognition (OCR)**: Advanced OCR with 98%+ accuracy for printed and handwritten text
- **Document Classification**: Automatic categorization of documents by type, purpose, and content
- **Information Extraction**: Extract key entities, dates, amounts, signatures, tables, and structured data
- **Natural Language Processing**: Sentiment analysis, summarization, and content understanding
- **Form Processing**: Intelligent form field detection and data extraction
- **Contract Analysis**: Legal document analysis with clause identification and risk assessment
- **Invoice Processing**: Automated invoice data extraction and validation

Integration Requirements

- **Cloud Storage Integration**: AWS S3, Azure Blob, Google Cloud Storage, SharePoint, Dropbox
- **Enterprise Systems**: ERP systems, CRM platforms, document management systems
- **API-First Architecture**: RESTful APIs for seamless integration with existing workflows
- **Workflow Automation**: Integration with workflow engines and business process management systems
- **Real-time Processing**: Support for real-time document processing and batch operations

Data Requirements

- **Training Datasets**: Large-scale document datasets for various document types and languages
- **OCR Training Data**: Diverse text recognition datasets including handwritten text
- **NLP Models**: Pre-trained language models fine-tuned for document understanding
- **Classification Models**: Document type classification with industry-specific categories
- **Entity Recognition**: Named entity recognition models for legal, financial, and business documents

Technical Themes

- **Computer Vision**: Advanced OCR, layout analysis, and image processing
- **Natural Language Processing**: Text analysis, entity extraction, and document understanding
- **Machine Learning**: Document classification, information extraction, and pattern recognition
- **Cloud Computing**: Scalable processing infrastructure and storage solutions
- **API Development**: Comprehensive API ecosystem for document processing workflows

Expected Business Outcomes

- **90% reduction** in manual document processing time
- **95% accuracy** in data extraction and document classification
- **80% cost savings** in document processing operations
- **Real-time processing** capabilities for time-sensitive documents
- **Compliance automation** for regulatory document requirements

Technical Approach

Architecture Strategy

- **Microservices Architecture**: Scalable, modular design with independent processing services
- **Event-Driven Processing**: Asynchronous document processing with queue management
- **AI/ML Pipeline**: Integrated ML models for OCR, classification, and extraction
- **Cloud-Native Deployment**: Containerized services with auto-scaling capabilities
- **API Gateway**: Centralized API management with authentication and rate limiting

AI/ML Implementation

- **Multi-Modal Models**: Combined vision and language models for document understanding
- **Transfer Learning**: Fine-tuned models for specific document types and industries
- **Ensemble Methods**: Multiple model approaches for improved accuracy and reliability
- **Active Learning**: Continuous model improvement through user feedback and corrections
- **Model Versioning**: MLOps practices for model deployment and management

Data Processing Pipeline

- **Document Ingestion**: Multi-format document upload and preprocessing
- **OCR Processing**: Text extraction with confidence scoring and quality assessment
- **Layout Analysis**: Document structure understanding and content segmentation
- **Information Extraction**: Entity recognition, table extraction, and data validation
- **Post-Processing**: Data cleansing, formatting, and quality assurance

Implementation Strategy

Phase 1: Core Document Processing (Months 1-3)

- Basic OCR and text extraction capabilities
- Document format support (PDF, DOCX, images)
- Simple document classification
- RESTful API development
- Cloud storage integration

Phase 2: Advanced AI Features (Months 4-6)

- Advanced OCR with handwriting recognition
- Intelligent document classification
- Entity recognition and information extraction
- Form processing capabilities
- Workflow integration APIs

Phase 3: Specialized Processing (Months 7-9)

- Contract analysis and legal document processing
- Invoice and financial document processing
- Multi-language support
- Advanced NLP features (summarization, sentiment analysis)
- Real-time processing capabilities

Phase 4: Enterprise Features (Months 10-12)

- Advanced workflow automation
- Enterprise system integrations
- Compliance and audit features
- Advanced analytics and reporting
- Performance optimization and scaling

Success Metrics

- **Processing Accuracy**: >95% accuracy in data extraction across document types
- **Processing Speed**: <30 seconds average processing time per document
- **System Availability**: 99.9% uptime with disaster recovery capabilities
- **User Adoption**: 80% user satisfaction score and active platform usage
- **Cost Efficiency**: 80% reduction in document processing operational costs

This document intelligence platform will revolutionize how organizations handle document processing, enabling automated workflows, improved accuracy, and significant cost savings through AI-powered document understanding and processing capabilities. # Product Requirements Document (PRD) ## Document Intelligence and Processing Platform - AI-Powered Document Processing System

Building upon README foundation for comprehensive product specification

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… Problem statement analysis identifying document processing challenges and AI/ML opportunities
- âœ… Market research on document intelligence platforms and competitive landscape
- âœ… Stakeholder requirements gathering from business users, IT teams, and compliance officers

TASK

Develop comprehensive product requirements defining business objectives, market positioning, user personas, success metrics, core features, technical requirements, and implementation roadmap for an AI-powered document intelligence platform that automates document processing workflows with 95% accuracy and 90% time reduction.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Business objectives aligned with organizational document processing goals and ROI targets - [ ] Market analysis validated through competitive research and industry benchmarking - [ ] User personas validated through stakeholder interviews and user research - [ ] Success metrics defined with measurable KPIs and baseline measurements - [ ] Feature requirements validated against user workflows and business processes - [ ] Technical requirements assessed for feasibility and scalability

Validation Criteria: - [ ] Product requirements validated with business stakeholders and executive sponsors - [ ] Technical feasibility validated with engineering teams and AI/ML experts - [ ] Market positioning validated through competitive analysis and customer feedback - [ ] Success metrics validated with business analysts and performance management teams - [ ] Implementation roadmap validated with project management and resource planning teams

EXIT CRITERIA

- âœ… Complete product requirements specification ready for functional requirements development
- âœ… Business case established with clear ROI projections and success metrics
- âœ… User personas and workflows documented for design and development guidance
- âœ… Technical requirements framework established for architecture and implementation
- âœ… Foundation prepared for detailed functional requirements specification

---

Reference to Previous Documents

This PRD builds upon **README** foundations: - **Problem Overview** â†' Business objectives and market opportunity - **Key Requirements** â†' Core features and technical capabilities - **Technical Approach** â†' Product architecture and implementation strategy - **Expected Outcomes** â†' Success metrics and business value proposition

1. Business Objectives

1.1 Primary Business Goals

- **Operational Efficiency**: Achieve 90% reduction in manual document processing time across all document types
- **Cost Optimization**: Deliver 80% cost savings in document processing operations within 12 months
- **Accuracy Improvement**: Maintain 95%+ accuracy in data extraction and document classification
- **Compliance Automation**: Automate 85% of regulatory document processing and compliance workflows
- **Scalability Achievement**: Support processing of 1M+ documents per month with auto-scaling capabilities

1.2 Strategic Business Outcomes

- **Digital Transformation**: Enable paperless operations and automated document workflows
- **Competitive Advantage**: Provide faster, more accurate document processing than manual methods
- **Revenue Growth**: Enable new business opportunities through improved document processing capabilities
- **Risk Mitigation**: Reduce human error in critical document processing and compliance activities
- **Customer Satisfaction**: Improve service delivery through faster document processing and response times

1.3 Return on Investment (ROI) Projections

- **Year 1**: 300% ROI through operational efficiency gains and cost reduction
- **Year 2**: 500% ROI with expanded document processing capabilities and user adoption
- **Year 3**: 750% ROI through advanced AI features and enterprise-wide deployment
- **Break-even Point**: 8 months from initial deployment
- **Total Cost of Ownership**: 60% lower than traditional document processing solutions

2. Market Analysis

2.1 Market Opportunity

- **Total Addressable Market (TAM)**: $8.1B global document processing market

- **Serviceable Addressable Market (SAM)**: $2.3B AI-powered document intelligence segment
- **Serviceable Obtainable Market (SOM)**: $115M target market for enterprise solutions
- **Market Growth Rate**: 23.7% CAGR in AI document processing market
- **Market Drivers**: Digital transformation, compliance requirements, operational efficiency

2.2 Competitive Landscape

Direct Competitors: - **Microsoft Form Recognizer**: Strong cloud integration, limited customization - **Google Document AI**: Advanced ML models, complex pricing structure - **Amazon Textract**: Comprehensive OCR, limited workflow integration - **ABBYY FlexiCapture**: Mature platform, high implementation complexity - **Kofax TotalAgility**: Enterprise focus, legacy technology constraints

Competitive Advantages: - **Superior Accuracy**: 95%+ extraction accuracy vs.Â 85-90% industry average - **Unified Platform**: Single solution for all document types vs.Â specialized tools - **Real-time Processing**: <30 second processing vs.Â minutes for competitors - **API-First Design**: Seamless integration vs.Â complex implementation requirements - **Cost Efficiency**: 40% lower TCO than enterprise alternatives

2.3 Market Positioning

- **Primary Position**: â€œThe most accurate and fastest AI-powered document intelligence platformâ€
- **Secondary Position**: â€œComplete document processing automation for enterprise workflowsâ€
- **Value Proposition**: â€œTransform document processing with 95% accuracy, 90% time reduction, and seamless integrationâ€
- **Target Differentiation**: Superior accuracy, speed, and ease of integration
- **Brand Promise**: â€œIntelligent document processing that just worksâ€

3. User Personas and Stakeholders

3.1 Primary User Personas

3.1.1 Document Processing Specialist (Primary User)

Demographics: - Role: Document Processing Manager, Operations Specialist - Experience: 5-10 years in document management and business operations - Industry: Financial services, healthcare, legal, manufacturing - Team Size: 10-50 document processing staff

Goals and Motivations: - Reduce manual document processing workload by 80% - Improve accuracy and consistency in data extraction - Streamline document workflows and approval processes - Meet compliance requirements and audit standards - Demonstrate operational efficiency improvements

Pain Points: - High volume of manual document processing tasks - Inconsistent data extraction accuracy and quality - Time-consuming document classification and routing - Compliance and audit trail requirements - Integration challenges with existing systems

Success Criteria: - 90% reduction in manual processing time - 95%+ accuracy in data extraction - Seamless integration with existing workflows - Complete audit trail and compliance reporting - Intuitive user interface requiring minimal training

3.1.2 IT Administrator (Technical User)

Demographics: - Role: IT Manager, Systems Administrator, Integration Specialist - Experience: 7-15 years in enterprise IT and system integration - Industry: Cross-industry enterprise IT departments - Team Size: 5-20 IT professionals

Goals and Motivations: - Deploy scalable document processing solution - Ensure secure and compliant data handling - Integrate with existing enterprise systems - Maintain system performance and availability - Minimize operational overhead and maintenance

Pain Points: - Complex integration requirements with legacy systems - Security and compliance concerns with document data - Scalability challenges with increasing document volumes - Limited technical resources for implementation and maintenance - Need for comprehensive monitoring and management tools

Success Criteria: - Seamless API integration with existing systems - Enterprise-grade security and compliance features - Auto-scaling capabilities for variable workloads - Comprehensive monitoring and alerting - Minimal maintenance and operational overhead

3.1.3 Business Analyst (Strategic User)

Demographics: - Role: Business Analyst, Process Improvement Manager, Operations Director - Experience: 8-15 years in business analysis and process optimization - Industry: Cross-industry business operations and strategy - Team Size: 3-15 business analysts and process specialists

Goals and Motivations: - Optimize document processing workflows and efficiency - Generate insights from document processing data - Measure and report on operational improvements - Identify opportunities for further automation - Support strategic decision-making with data analytics

Pain Points: - Limited visibility into document processing performance - Difficulty measuring ROI and operational improvements - Lack of analytics and reporting capabilities - Manual effort required for process optimization - Challenges in identifying bottlenecks and inefficiencies

Success Criteria: - Comprehensive analytics and reporting dashboard - Real-time visibility into processing performance - Measurable ROI and operational improvements - Automated workflow optimization recommendations - Integration with business intelligence tools

3.2 Secondary Stakeholders

3.2.1 Compliance Officer

- **Primary Concern**: Regulatory compliance and audit requirements
- **Key Requirements**: Complete audit trails, data privacy, retention policies
- **Success Metrics**: 100% compliance with industry regulations

3.2.2 Executive Sponsor

- **Primary Concern**: Business value and ROI achievement
- **Key Requirements**: Clear ROI metrics, strategic alignment, competitive advantage
- **Success Metrics**: Measurable business impact and cost savings

3.2.3 End Users (Document Processors)

- **Primary Concern**: Ease of use and workflow efficiency
- **Key Requirements**: Intuitive interface, minimal training, error handling
- **Success Metrics**: User adoption rate and satisfaction scores

4. Success Metrics and KPIs

4.1 Operational Performance Metrics

- **Processing Speed**: Average processing time <30 seconds per document
- **Accuracy Rate**: >95% accuracy in data extraction across all document types
- **Throughput**: Support for 1M+ documents processed per month
- **System Availability**: 99.9% uptime with <1 hour mean time to recovery
- **Error Rate**: <2% error rate in document classification and extraction

4.2 Business Impact Metrics

- **Time Reduction**: 90% reduction in manual document processing time

- **Cost Savings**: 80% reduction in document processing operational costs
- **ROI Achievement**: 300% ROI within first year of deployment
- **Productivity Improvement**: 5x increase in documents processed per FTE
- **Compliance Rate**: 100% compliance with regulatory requirements

4.3 User Experience Metrics

- **User Adoption**: 85% active user adoption within 6 months
- **User Satisfaction**: >4.5/5 user satisfaction score
- **Training Time**: <4 hours average time to user proficiency
- **Support Tickets**: <5% of users requiring support per month
- **Feature Utilization**: >70% utilization of core platform features

4.4 Technical Performance Metrics

- **API Response Time**: <2 seconds average API response time
- **Scalability**: Auto-scaling to handle 10x peak load increases
- **Integration Success**: <2 weeks average integration time with enterprise systems
- **Data Security**: Zero security incidents or data breaches
- **System Reliability**: <0.1% data loss rate with complete backup recovery

5. Core Features and Capabilities

5.1 Document Processing Engine

Feature Description: Advanced AI-powered document processing with multi-format support **Business Value**: Eliminates manual document processing and improves accuracy **User Stories**: - As a document processor, I want to upload multiple document formats so that I can process all document types in one platform - As an IT administrator, I want batch processing capabilities so that I can handle large document volumes efficiently - As a business analyst, I want processing analytics so that I can monitor and optimize document workflows

Acceptance Criteria: - Support for PDF, DOCX, images (JPG, PNG), scanned documents, and text files - Batch processing of up to 1000 documents simultaneously - Real-time processing status and progress tracking - Automatic format detection and optimization - Processing history and audit trail maintenance

5.2 Optical Character Recognition (OCR)

Feature Description: Advanced OCR with 98%+ accuracy for printed and handwritten text **Business Value**: Converts unstructured documents into searchable, processable data **User Stories**: - As a document processor, I want accurate text extraction so that I can digitize paper documents - As a compliance officer, I want handwriting recognition so that I can process handwritten forms and signatures - As a business analyst, I want OCR confidence scoring so that I can assess data quality

Acceptance Criteria: - 98%+ accuracy for printed text recognition - 90%+ accuracy for handwritten text recognition - Multi-language support for 25+ languages - Confidence scoring for all extracted text - Support for complex layouts and formatting

5.3 Intelligent Document Classification

Feature Description: Automatic document categorization using AI/ML models **Business Value**: Eliminates manual document sorting and enables automated routing **User Stories**: - As a document processor, I want automatic document classification so that I can route documents to appropriate workflows - As an IT administrator, I want custom classification models so that I can adapt to organization-specific document types - As a business analyst, I want classification analytics so that I can understand document patterns

Acceptance Criteria: - Pre-trained models for 50+ common document types - Custom model training for organization-specific documents - 95%+ accuracy in document classification - Confidence scoring and manual override capabilities - Real-time classification with <5 second response time

5.4 Information Extraction and Entity Recognition

Feature Description: Extract key entities, dates, amounts, and structured data from documents **Business Value**: Automates data entry and reduces manual extraction errors **User Stories**: - As a document processor, I want automatic data extraction so that I can populate forms and databases - As a compliance officer, I want entity recognition so that I can identify sensitive information - As a business analyst, I want structured data output so that I can integrate with business systems

Acceptance Criteria: - Named entity recognition for people, organizations, locations, dates, amounts - Custom entity extraction for industry-specific data fields - Table and form data extraction with structure preservation - Data validation and quality scoring - Multiple output formats (JSON, XML, CSV, database integration)

5.5 Contract and Legal Document Analysis

Feature Description: Specialized processing for legal documents with clause identification **Business Value**: Accelerates legal document review and risk assessment **User Stories**: - As a legal professional, I want clause identification so that I can quickly review contract terms - As a compliance officer, I want risk assessment so that I can identify potential legal issues - As a business analyst, I want contract analytics so that I can track contract performance

Acceptance Criteria: - Pre-trained models for common contract types and clauses - Risk scoring and flagging of potentially problematic terms - Key date and obligation extraction - Comparison capabilities for contract versions - Integration with legal document management systems

6. Technical Requirements

6.1 Performance Requirements

- **Processing Speed**: <30 seconds average processing time per document
- **Concurrent Users**: Support for 1000+ concurrent users
- **Throughput**: 1M+ documents processed per month capacity
- **API Performance**: <2 seconds average API response time
- **Scalability**: Auto-scaling to handle 10x load increases

6.2 Integration Requirements

- **API Standards**: RESTful APIs with OpenAPI 3.0 specification
- **Authentication**: OAuth 2.0, SAML 2.0, and enterprise SSO support
- **Data Formats**: JSON, XML, CSV input/output with schema validation
- **Cloud Storage**: AWS S3, Azure Blob, Google Cloud Storage, SharePoint integration
- **Enterprise Systems**: ERP, CRM, document management system connectors

6.3 Security Requirements

- **Data Encryption**: AES-256 encryption at rest, TLS 1.3 in transit
- **Access Control**: Role-based access control with fine-grained permissions
- **Audit Logging**: Complete audit trail for all document processing activities
- **Compliance**: SOC 2, GDPR, HIPAA, PCI DSS compliance support
- **Data Privacy**: Data isolation, retention policies, and right-to-delete capabilities

6.4 Reliability Requirements

- **System Availability**: 99.9% uptime with disaster recovery capabilities
- **Data Integrity**: <0.1% data loss rate with automated backup and recovery

- **Error Handling**: Graceful error handling with retry mechanisms
- **Monitoring**: Comprehensive system monitoring and alerting
- **Backup and Recovery**: Automated backup with <4 hour recovery time objective

7. Business Constraints and Assumptions

7.1 Business Constraints

- **Budget Allocation**: $2.5M development budget over 12 months
- **Timeline**: 12-month development and deployment timeline
- **Resource Availability**: 15-person development team with AI/ML expertise
- **Compliance Requirements**: Must meet industry-specific regulatory requirements
- **Integration Complexity**: Must integrate with existing enterprise systems

7.2 Technical Assumptions

- **Cloud Infrastructure**: Deployment on major cloud platforms (AWS, Azure, GCP)
- **AI/ML Models**: Availability of pre-trained models and training datasets
- **Processing Power**: Sufficient computational resources for AI/ML processing
- **Network Connectivity**: Reliable high-speed internet connectivity for cloud services
- **Third-party Services**: Availability and reliability of external API services

7.3 Market Assumptions

- **Market Demand**: Continued growth in document processing automation demand
- **Technology Adoption**: Enterprise readiness for AI-powered document processing
- **Competitive Landscape**: Stable competitive environment without major disruptions
- **Regulatory Environment**: Stable regulatory requirements for document processing
- **Economic Conditions**: Stable economic conditions supporting technology investments

8. Risk Assessment and Mitigation

8.1 Technical Risks

Risk: AI/ML model accuracy below target thresholds - **Probability**: Medium - **Impact**: High - **Mitigation**: Extensive model training, ensemble methods, continuous learning, fallback to manual processing

Risk: Integration complexity with legacy systems - **Probability**: High - **Impact**: Medium - **Mitigation**: Comprehensive API design, integration testing, phased rollout, dedicated integration team

Risk: Performance degradation under high load - **Probability**: Medium - **Impact**: High - **Mitigation**: Load testing, auto-scaling architecture, performance monitoring, capacity planning

8.2 Business Risks

Risk: User adoption slower than projected - **Probability**: Medium - **Impact**: Medium - **Mitigation**: User training programs, change management, phased rollout, user feedback incorporation

Risk: Competitive pressure from established players - **Probability**: High - **Impact**: Medium - **Mitigation**: Differentiated features, superior performance, competitive pricing, strong customer relationships

Risk: Regulatory compliance challenges - **Probability**: Low - **Impact**: High - **Mitigation**: Early compliance assessment, legal review, compliance-by-design, regular audits

8.3 Operational Risks

Risk: Data security and privacy breaches - **Probability**: Low - **Impact**: Very High - **Mitigation**: Security-by-design, regular security audits, encryption, access controls, incident response plan

Risk: Vendor dependency and service availability - **Probability**: Medium - **Impact**: Medium - **Mitigation**: Multi-vendor strategy, service level agreements, backup services, vendor relationship management

This PRD establishes the foundation for developing a comprehensive AI-powered document intelligence platform that delivers significant business value through automated document processing, superior accuracy, and seamless integration capabilities. # Functional Requirements Document (FRD) ## Document Intelligence and Processing Platform - AI-Powered Document Processing System

Building upon README and PRD foundations for detailed functional specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… PRD completed with business objectives, user personas, success metrics, and core features

TASK

Develop comprehensive functional requirements specifying detailed system behaviors, user interactions, data processing workflows, AI/ML capabilities, integration interfaces, and acceptance criteria for the document intelligence platform.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All functional requirements mapped to PRD features and business objectives - [ ] User workflows documented with step-by-step interaction specifications - [ ] AI/ML processing requirements defined with accuracy and performance criteria - [ ] Integration requirements specified with detailed API and data exchange protocols

Validation Criteria: - [ ] Functional requirements validated with user personas and workflow analysis - [ ] Processing workflows validated with document processing specialists - [ ] AI/ML requirements validated with data scientists and ML engineers - [ ] Integration requirements validated with IT administrators

EXIT CRITERIA

- âœ… Complete functional requirements specification ready for non-functional requirements development
- âœ… Detailed system behaviors and user interactions documented
- âœ… AI/ML processing workflows specified with accuracy requirements
- âœ… Integration interfaces defined with comprehensive API specifications

---

1. Document Processing Engine Module

1.1 Multi-Format Document Ingestion

Requirement ID: FRD-DPE-001 **Priority**: Critical **User Story**: As a document processor, I want to upload multiple document formats so that I can process all document types in one platform

Functional Specification: The system SHALL provide document ingestion capabilities supporting multiple input formats with automatic format detection and preprocessing optimization.

Detailed Requirements: - Support PDF, DOCX, DOC, RTF, TXT, JPG, PNG, TIFF, BMP formats - Provide drag-and-drop interface for single and multiple document upload - Support batch upload of up to 1000 documents simultaneously - Automatically detect document format and apply appropriate preprocessing - Validate document integrity and provide error messages for corrupted files - Support cloud storage integration (AWS S3, Azure Blob, Google Drive, SharePoint) - Provide API endpoints for programmatic document submission - Generate unique document identifiers and maintain processing history

Acceptance Criteria: - System accepts all specified formats with 100% success rate - Batch upload completes within 30 seconds for 100 documents - Document format detection accuracy >99% - API endpoints respond within 2 seconds

1.2 Document Preprocessing and Optimization

Requirement ID: FRD-DPE-002 **Priority**: High **Functional Specification**: The system SHALL provide intelligent document preprocessing to optimize image quality, correct orientation, and prepare documents for accurate OCR processing.

Detailed Requirements: - Automatically detect and correct document orientation (0Â°, 90Â°, 180Â°, 270Â°) - Apply image enhancement algorithms (noise reduction, contrast adjustment) - Detect and correct skewed documents with automatic deskewing - Remove background noise and artifacts from scanned documents - Optimize image resolution for OCR processing (minimum 300 DPI) - Detect and separate multi-page documents automatically - Preserve original document metadata and properties - Generate preprocessing quality scores and confidence metrics

Acceptance Criteria: - Orientation detection accuracy >98% - Image enhancement improves OCR accuracy by minimum 15% - Preprocessing completes within 10 seconds per document

2. Optical Character Recognition (OCR) Module

2.1 Advanced Text Recognition Engine

Requirement ID: FRD-OCR-001 **Priority**: Critical **User Story**: As a document processor, I want accurate text extraction so that I can digitize paper documents with high fidelity

Functional Specification: The system SHALL provide advanced OCR capabilities with 98%+ accuracy for printed text and 90%+ accuracy for handwritten text across multiple languages.

Detailed Requirements: - Achieve 98%+ accuracy for printed text recognition - Achieve 90%+ accuracy for handwritten text recognition - Support 25+ languages including English, Spanish, French, German, Chinese, Japanese, Arabic - Handle complex document layouts with multiple columns, tables, and mixed content - Preserve text formatting including fonts, sizes, bold, italic, underline - Detect and maintain document structure (headings, paragraphs, lists, tables) - Process mathematical formulas and special characters accurately - Generate confidence scores for all extracted text elements

Acceptance Criteria: - Printed text accuracy exceeds 98% on standardized datasets - Handwritten text accuracy exceeds 90% on diverse samples - Multi-language support validated for all 25 specified languages - Complex layout preservation accuracy >95%

2.2 Intelligent Layout Analysis

Requirement ID: FRD-OCR-002 **Priority**: High **Functional Specification**: The system SHALL provide intelligent layout analysis to identify document structure, regions, and content organization for accurate text extraction.

Detailed Requirements: - Automatically detect document regions (headers, footers, body text, sidebars, images) - Identify and extract table structures with row and column relationships - Recognize form fields and associate labels with input areas - Detect reading order for multi-column and complex layouts - Identify and separate text from graphics and images - Recognize signatures, stamps, and handwritten annotations - Maintain spatial relationships between document elements - Generate structured output preserving document hierarchy

Acceptance Criteria: - Document region detection accuracy >95% - Table structure extraction maintains 100% row-column relationships - Form field association accuracy >90% - Reading order detection accuracy >95%

3. Document Classification Module

3.1 Intelligent Document Type Classification

Requirement ID: FRD-CLS-001 **Priority**: Critical **User Story**: As a document processor, I want automatic document classification so that documents are routed to appropriate workflows

Functional Specification: The system SHALL provide intelligent document classification using AI/ML models to automatically categorize documents with 95%+ accuracy.

Detailed Requirements: - Classify documents into 50+ pre-trained categories (invoices, contracts, forms, reports) - Achieve 95%+ accuracy in document type classification - Support custom classification models for organization-specific document types - Provide confidence scores for all classification decisions - Enable manual override and correction of classification results - Support hierarchical classification with main categories and subcategories - Process classification within 5 seconds per document - Continuously improve models through user feedback

Acceptance Criteria: - Classification accuracy exceeds 95% on diverse document sets - Custom model training completes within 24 hours - Confidence scores accurately predict classification reliability - Manual override interface allows immediate correction

3.2 Content-Based Document Routing

Requirement ID: FRD-CLS-002 **Priority**: High **Functional Specification**: The system SHALL provide intelligent document routing based on classification results to automatically direct documents to appropriate workflows.

Detailed Requirements: - Define routing rules based on document type, content, and metadata - Support conditional routing with multiple criteria and logic operators - Integrate with workflow management systems and business process engines - Provide real-time routing notifications and status updates - Enable manual routing override and exception handling - Maintain routing history and audit trail for compliance - Support priority-based routing for urgent documents - Generate routing analytics and performance reports

Acceptance Criteria: - Routing rules execute within 3 seconds of classification - Conditional routing supports complex business logic with 100% accuracy - Workflow integration maintains document context and metadata

4. Information Extraction Module

4.1 Named Entity Recognition and Extraction

Requirement ID: FRD-EXT-001 **Priority**: Critical **User Story**: As a document processor, I want automatic data extraction so that I can populate forms and databases without manual entry

Functional Specification: The system SHALL provide comprehensive named entity recognition to identify and extract key information with 95%+ accuracy.

Detailed Requirements: - Extract standard entities (persons, organizations, locations, dates, monetary amounts) - Support custom entity extraction for industry-specific data fields - Achieve 95%+ accuracy in entity recognition and extraction - Provide confidence scores for all extracted entities - Maintain entity relationships and context within documents - Support multi-language entity extraction for 25+ languages - Extract entities from tables, forms, and structured content - Generate structured output in JSON, XML, and CSV formats

Acceptance Criteria: - Entity extraction accuracy exceeds 95% on diverse documents - Custom entity training completes within 12 hours - Confidence scores predict extraction accuracy within 5% margin - Multi-language extraction maintains accuracy across supported languages

4.2 Table and Form Data Extraction

Requirement ID: FRD-EXT-002 **Priority**: High **Functional Specification**: The system SHALL provide intelligent table and form data extraction to identify, extract, and structure tabular data while preserving relationships.

Detailed Requirements: - Detect and extract table structures with headers, rows, and columns - Identify form fields and associate labels with corresponding values - Preserve table relationships and data hierarchy - Handle complex table layouts including merged cells and nested tables - Extract checkbox, radio button, and signature field values - Validate extracted data against expected formats and constraints - Support table extraction from multi-page documents - Generate structured output maintaining original table organization

Acceptance Criteria: - Table structure detection accuracy >90% for standard layouts - Form field association accuracy >95% for common form types - Data relationships preserved with 100% accuracy in structured output

5. Integration and API Module

5.1 RESTful API Services

Requirement ID: FRD-API-001 **Priority**: Critical **User Story**: As an IT administrator, I want comprehensive APIs so that I can integrate document processing with existing systems

Functional Specification: The system SHALL provide comprehensive RESTful API services following OpenAPI 3.0 standards for seamless integration.

Detailed Requirements: - Implement RESTful APIs following OpenAPI 3.0 specification - Provide APIs for document upload, processing, status checking, and result retrieval - Support synchronous and asynchronous processing modes - Implement comprehensive error handling with detailed error codes - Provide API rate limiting and throttling capabilities - Support API versioning and backward compatibility - Generate comprehensive API documentation with examples - Implement API monitoring and analytics

Acceptance Criteria: - All APIs respond within 2 seconds for standard operations - API documentation provides complete integration examples - Error handling covers all failure scenarios with appropriate HTTP status codes - Rate limiting prevents system overload while maintaining service availability

5.2 Enterprise System Integration

Requirement ID: FRD-API-002 **Priority**: High **Functional Specification**: The system SHALL provide pre-built connectors for common enterprise systems including ERP, CRM, and document management systems.

Detailed Requirements: - Provide connectors for major ERP systems (SAP, Oracle, Microsoft Dynamics) - Support CRM integration (Salesforce, HubSpot, Microsoft CRM) - Integrate with document management systems (SharePoint, Box, Dropbox) - Connect with workflow engines (Microsoft Power Automate, Zapier) - Support database integration with JDBC and ODBC connectivity - Provide message queue integration (Apache Kafka, RabbitMQ, Azure Service Bus) - Enable webhook notifications for real-time event processing - Maintain integration monitoring and error handling

Acceptance Criteria: - Enterprise connectors support standard authentication methods - Integration setup completes within 4 hours for standard configurations - Data synchronization maintains consistency with <1% error rate - Webhook notifications deliver within 5 seconds of event occurrence

This FRD establishes 42 detailed functional requirements across 5 core modules, providing the foundation for developing a comprehensive AI-powered document intelligence platform that meets all business objectives and user needs specified in the README and PRD documents. # Non-Functional Requirements Document (NFRD) ## Document Intelligence and Processing Platform - AI-Powered Document Processing System

Building upon README, PRD, and FRD foundations for comprehensive non-functional specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… PRD completed with business objectives, user personas, success metrics, and core features
- âœ… FRD completed with 42 detailed functional requirements across 5 system modules
- âœ… Performance targets established (<30s processing, 95% accuracy, 99.9% uptime)
- âœ… Integration requirements defined for enterprise systems and cloud platforms

TASK

Develop comprehensive non-functional requirements addressing performance, scalability, reliability, security, usability, compliance, and operational requirements that enable the document intelligence platform to achieve 95% accuracy, process 1M+ documents monthly, and maintain 99.9% availability.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Performance requirements aligned with PRD success metrics and FRD processing specifications - [ ] Scalability requirements support projected user growth and document volume increases - [ ] Security requirements address data protection, privacy, and compliance obligations - [ ] Reliability requirements ensure business continuity and disaster recovery capabilities - [ ] Usability requirements validated against user personas and accessibility standards - [ ] Compliance requirements cover industry regulations and data protection laws

Validation Criteria: - [ ] Performance requirements validated with system architects and performance engineers - [ ] Scalability requirements validated with infrastructure teams and capacity planners - [ ] Security requirements validated with cybersecurity experts and compliance officers - [ ] Reliability requirements validated with operations teams and business continuity planners - [ ] Usability requirements validated with UX designers and accessibility experts - [ ] Compliance requirements validated with legal teams and regulatory specialists

EXIT CRITERIA

- âœ… Complete non-functional requirements specification ready for architecture design
- âœ… Performance benchmarks and scalability targets defined for system design
- âœ… Security framework established for data protection and access control
- âœ… Reliability standards defined for business continuity and disaster recovery
- âœ… Foundation prepared for system architecture and technical design specifications

---

Reference to Previous Documents

This NFRD builds upon **README**, **PRD**, and **FRD** foundations: - **README Expected Outcomes** â†’ Non-functional requirements supporting 90% time reduction and 95% accuracy - **PRD Success Metrics** â†’ Performance requirements enabling 300% ROI and operational efficiency - **PRD User Personas** â†’ Usability requirements tailored for document processors, IT administrators, and business analysts - **FRD Processing Requirements** â†’ Performance specifications for OCR accuracy, classification speed, and API response times - **FRD Integration Requirements** â†’ Scalability and reliability requirements for enterprise system connectivity

1. Performance Requirements

1.1 Document Processing Performance

Requirement Category: Processing Speed and Throughput **Business Impact**: Critical for user productivity and operational efficiency

Performance Specifications: - **Document Processing Time**: Average processing time <30 seconds per document across all formats and sizes - **OCR Processing Speed**: Text extraction completes within 15 seconds for standard documents (up to 10 pages) - **Classification Response Time**: Document classification results delivered within 5 seconds - **Batch Processing Throughput**: Process minimum 1000 documents per hour in batch mode - **Concurrent Processing**: Support 500+ concurrent document processing operations - **Peak Load Handling**: Maintain performance during 3x normal load spikes - **Large Document Processing**: Documents up to 100MB processed within 2 minutes - **Multi-language Processing**: No performance degradation for non-English languages

Measurement Criteria: - 95th percentile processing time meets specified targets - Throughput measured under sustained load conditions - Performance monitoring with real-time metrics and alerting - Load testing validates peak capacity handling

1.2 API and System Response Performance

Requirement Category: System Responsiveness and User Experience **Business Impact**: Critical for integration success and user satisfaction

Performance Specifications: - **API Response Time**: REST API responses within 2 seconds for standard operations - **Database Query Performance**: Database queries complete within 500ms for 95% of requests - **User Interface Responsiveness**: Web interface actions complete within 3 seconds - **Search and Retrieval**: Document search results returned within 5 seconds - **Real-time Status Updates**: Processing status updates delivered within 2 seconds - **File Upload Performance**: Document upload completes at minimum 10MB/second - **Report Generation**: Analytics reports generated within 30 seconds - **System Startup Time**: Application services start within 60 seconds

Measurement Criteria: - Response time monitoring with percentile-based SLAs - Performance testing under various network conditions - User experience metrics tracking and optimization - Continuous performance monitoring and alerting

1.3 AI/ML Model Performance

Requirement Category: Machine Learning Accuracy and Speed **Business Impact**: Critical for business value and competitive advantage

Performance Specifications: - **OCR Accuracy**: 98%+ accuracy for printed text, 90%+ for handwritten text - **Classification Accuracy**: 95%+ accuracy in document type classification - **Entity Extraction Accuracy**: 95%+ accuracy in named entity recognition - **Model Inference Time**: ML model predictions delivered within 3 seconds - **Confidence Score Reliability**: Confidence scores predict actual accuracy within 5% margin - **Multi-language Accuracy**: Maintain accuracy targets across all supported languages - **Model Training Time**: Custom model training completes within 24 hours - **Model Update Frequency**: Models updated and deployed within 4 hours

Measurement Criteria: - Accuracy testing on standardized datasets and real-world samples - Performance benchmarking against industry standards - Continuous model monitoring and drift detection - A/B testing for model improvements and validation

2. Scalability Requirements

2.1 User and Concurrent Access Scalability

Requirement Category: User Load and Concurrent Operations **Business Impact**: Essential for enterprise adoption and growth

Scalability Specifications: - **Concurrent Users**: Support 1000+ concurrent active users - **User Growth**: Scale to 10,000+ registered users without performance degradation - **Session Management**: Handle 5000+ concurrent user sessions - **Multi-tenant Architecture**: Support 100+ organizational tenants with data isolation - **Geographic Distribution**: Support users across multiple time zones and regions - **Peak Usage Handling**: Auto-scale during business hours and peak periods - **Load Balancing**: Distribute load across multiple application instances - **Resource Allocation**: Dynamic resource allocation based on usage patterns

Measurement Criteria: - Load testing with realistic user behavior patterns - Scalability testing with gradual user increase scenarios - Performance monitoring during peak usage periods - Auto-scaling effectiveness and response time measurement

2.2 Data and Storage Scalability

Requirement Category: Data Volume and Storage Growth **Business Impact**: Critical for long-term system sustainability

Scalability Specifications: - **Document Volume**: Process and store 1M+ documents per month - **Data Growth**: Support 100% annual data growth without performance impact - **Storage Capacity**: Scale to petabyte-level document storage - **Database Performance**: Maintain query performance with billions of records - **Archive Management**: Automated data archiving and retrieval capabilities - **Backup Scalability**: Backup operations scale with data volume growth - **Search Index Scaling**: Search performance maintained with growing document corpus - **Metadata Management**: Efficient metadata storage and retrieval at scale

Measurement Criteria: - Storage performance testing with large datasets - Database scalability testing with projected data volumes - Search performance validation with growing document collections - Backup and recovery testing at scale

2.3 Infrastructure and Resource Scalability

Requirement Category: Computing Resources and Infrastructure **Business Impact**: Essential for cost-effective operations and performance

Scalability Specifications: - **Auto-scaling**: Automatic horizontal scaling based on demand (CPU, memory, queue depth) - **Resource Elasticity**: Scale up/down within 5 minutes of demand changes - **Multi-cloud Support**: Deploy across multiple cloud providers for redundancy - **Container Orchestration**: Kubernetes-based scaling with pod auto-scaling - **Database Scaling**: Read replicas and sharding for database scalability - **CDN Integration**: Content delivery network for global performance optimization - **Queue Management**: Message queue scaling for asynchronous processing - **Monitoring Scalability**: Monitoring and logging systems scale with infrastructure

Measurement Criteria: - Auto-scaling response time and effectiveness testing - Multi-cloud deployment and failover testing - Container orchestration performance validation - Infrastructure cost optimization analysis

3. Reliability and Availability Requirements

3.1 System Availability and Uptime

Requirement Category: Service Availability and Business Continuity **Business Impact**: Critical for business operations and customer trust

Reliability Specifications: - **System Uptime**: 99.9% availability (maximum 8.76 hours downtime per year) - **Planned Maintenance**: Scheduled maintenance windows <2 hours monthly - **Service Recovery**: Mean Time to Recovery (MTTR) <1 hour for critical issues - **Fault Tolerance**: System continues operation with single component failures - **Redundancy**: No single points of failure in critical system components - **Health Monitoring**: Continuous health checks with automated alerting - **Graceful Degradation**: Reduced functionality during partial system failures - **Service Level Agreement**: 99.9% SLA with financial penalties for breaches

Measurement Criteria: - Uptime monitoring with third-party validation - Failure simulation and recovery testing - Mean Time Between Failures (MTBF) tracking - Service level agreement compliance reporting

3.2 Data Integrity and Consistency

Requirement Category: Data Protection and Consistency **Business Impact**: Critical for business trust and regulatory compliance

Reliability Specifications: - **Data Loss Prevention**: Zero data loss tolerance for processed documents - **Data Consistency**: ACID compliance for all database transactions - **Backup Reliability**: 99.99% backup success rate with verification - **Data Corruption Detection**: Automated detection and correction of data corruption - **Version Control**: Complete audit trail for all data modifications - **Replication Consistency**: Data consistency across all replicated systems - **Transaction Integrity**: All processing operations are atomic and recoverable - **Data Validation**: Continuous data integrity validation and reporting

Measurement Criteria: - Data integrity testing with corruption simulation - Backup and recovery validation testing - Transaction consistency verification - Audit trail completeness and accuracy validation

3.3 Disaster Recovery and Business Continuity

Requirement Category: Disaster Recovery and Emergency Response **Business Impact**: Essential for business continuity and risk mitigation

Reliability Specifications: - **Recovery Time Objective (RTO)**: System recovery within 4 hours of disaster - **Recovery Point Objective (RPO)**: Maximum 1 hour of data loss in disaster scenarios - **Geographic Redundancy**: Multi-region deployment with automatic failover - **Backup Strategy**: Daily incremental backups with weekly full backups - **Disaster Recovery Testing**: Quarterly disaster recovery drills and validation - **Emergency Response**: 24/7 emergency response team and procedures - **Communication Plan**: Stakeholder communication during disaster events - **Business Impact Assessment**: Regular assessment and update of recovery procedures

Measurement Criteria: - Disaster recovery testing and validation - RTO and RPO compliance measurement - Failover testing and performance validation - Emergency response procedure effectiveness assessment

4. Security Requirements

4.1 Data Protection and Encryption

Requirement Category: Data Security and Privacy Protection **Business Impact**: Critical for regulatory compliance and customer trust

Security Specifications: - **Encryption at Rest**: AES-256 encryption for all stored data and documents - **Encryption in Transit**: TLS 1.3 encryption for all data transmission - **Key Management**: Hardware Security Module (HSM) for encryption key management - **Data Masking**: Sensitive data masking in non-production environments - **Secure Document Storage**: Encrypted document storage with access logging - **Database Encryption**: Transparent database encryption for all data stores - **Backup Encryption**: Encrypted backups with separate key management - **End-to-End Encryption**: Optional end-to-end encryption for sensitive documents

Measurement Criteria: - Encryption implementation validation and testing - Key management security audit and compliance - Data protection effectiveness assessment - Security vulnerability scanning and penetration testing

4.2 Access Control and Authentication

Requirement Category: User Access and Identity Management **Business Impact**: Critical for data security and regulatory compliance

Security Specifications: - **Multi-Factor Authentication**: MFA required for all user accounts - **Single Sign-On (SSO)**: Enterprise SSO integration (SAML 2.0, OAuth 2.0) - **Role-Based Access Control**: Granular permissions based on user roles and responsibilities - **Principle of Least Privilege**: Users granted minimum necessary permissions - **Session Management**: Secure session handling with automatic timeout - **Account Lockout**: Automatic account lockout after failed login attempts - **Privileged Access Management**: Enhanced security for administrative accounts - **API Authentication**: Secure API authentication with token-based access

Measurement Criteria: - Access control testing and validation - Authentication system security assessment - Privilege escalation testing and prevention - API security testing and vulnerability assessment

4.3 Compliance and Audit Requirements

Requirement Category: Regulatory Compliance and Audit Trail **Business Impact**: Essential for regulatory compliance and legal protection

Security Specifications: - **Audit Logging**: Complete audit trail for all system activities and data access - **Compliance Frameworks**: SOC 2 Type II, ISO 27001, GDPR, HIPAA, PCI DSS compliance - **Data Retention**: Configurable data retention policies with automated enforcement - **Right to Delete**: GDPR-compliant data deletion capabilities - **Privacy Controls**: Data privacy controls and consent management - **Regulatory Reporting**: Automated compliance reporting and documentation - **Security Monitoring**: 24/7 security monitoring and incident response - **Vulnerability Management**: Regular security assessments and vulnerability remediation

Measurement Criteria: - Compliance audit results and certification maintenance - Audit log completeness and integrity validation - Privacy control effectiveness assessment - Security incident response time and effectiveness

5. Usability and User Experience Requirements

5.1 User Interface and Experience Design

Requirement Category: User Interface Design and Usability **Business Impact**: Critical for user adoption and productivity

Usability Specifications: - **Intuitive Design**: User interface follows established UX design principles - **Learning Curve**: New users achieve proficiency within 4 hours of training - **Task Efficiency**: Common tasks completed 50% faster than manual processes - **Error Prevention**: Interface design prevents common user errors - **Responsive Design**: Optimal experience across desktop, tablet, and mobile devices - **Accessibility Compliance**: WCAG 2.1 AA accessibility standards compliance - **Multi-language Support**: User interface available in 10+ languages - **Customization**: Configurable dashboards and workflow preferences

Measurement Criteria: - User experience testing and feedback collection - Task completion time measurement and optimization - Accessibility testing and compliance validation - User satisfaction surveys and Net Promoter Score tracking

5.2 Documentation and Support

Requirement Category: User Support and Documentation **Business Impact**: Essential for successful user adoption and system utilization

Usability Specifications: - **Comprehensive Documentation**: Complete user guides, API documentation, and tutorials - **Interactive Help**: Context-sensitive help and guided tutorials - **Video Training**: Video tutorials for all major system functions - **Knowledge Base**: Searchable knowledge base with FAQs and troubleshooting - **Support Channels**: Multiple support channels (chat, email, phone, ticketing) - **Response Times**: Support ticket response within 4 hours for standard issues - **Self-Service**: Self-service capabilities for common tasks and configurations - **Community Support**: User community forums and peer-to-peer support

Measurement Criteria: - Documentation completeness and accuracy assessment - Support ticket resolution time and satisfaction measurement - Self-service utilization and effectiveness tracking - Training effectiveness and user proficiency measurement

6. Operational Requirements

6.1 Monitoring and Observability

Requirement Category: System Monitoring and Operations **Business Impact**: Critical for system reliability and performance optimization

Operational Specifications: - **Application Performance Monitoring**: Real-time monitoring of all system components - **Infrastructure Monitoring**: Server, network, and resource utilization monitoring - **Business Metrics**: Key business metrics tracking and alerting - **Log Management**: Centralized logging with search and analysis capabilities - **Distributed Tracing**: End-to-end request tracing for performance optimization - **Custom Dashboards**: Configurable monitoring dashboards for different stakeholders - **Automated Alerting**: Intelligent alerting with escalation procedures - **Capacity Planning**: Predictive analytics for capacity planning and optimization

Measurement Criteria: - Monitoring coverage and effectiveness assessment - Alert accuracy and false positive rate measurement - Dashboard utilization and effectiveness tracking - Capacity planning accuracy and optimization results

6.2 Maintenance and Updates

Requirement Category: System Maintenance and Lifecycle Management **Business Impact**: Essential for system security and continuous improvement

Operational Specifications: - **Zero-Downtime Deployments**: Blue-green deployment strategy for updates - **Automated Testing**: Comprehensive automated testing for all deployments - **Rollback Capabilities**: Immediate rollback capability for failed deployments - **Security Patching**: Automated security patch management and deployment - **Database Migrations**: Safe database schema migrations with rollback support - **Configuration Management**: Version-controlled configuration management - **Environment Consistency**: Consistent environments across development, staging, and production - **Change Management**: Formal change management process for all system modifications

Measurement Criteria: - Deployment success rate and rollback frequency tracking - Security patch deployment time and coverage measurement - Environment consistency validation and drift detection - Change management process effectiveness assessment

This NFRD establishes comprehensive non-functional requirements that ensure the document intelligence platform delivers enterprise-grade performance, security, and reliability while maintaining excellent user experience and operational efficiency. # Architecture Diagram (AD) ## Document Intelligence and

Processing Platform - AI-Powered Document Processing System

Building upon README, PRD, FRD, and NFRD foundations for comprehensive system architecture

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… PRD completed with business objectives, user personas, success metrics, and core features
- âœ… FRD completed with 42 detailed functional requirements across 5 system modules
- âœ… NFRD completed with performance (<30s processing), scalability (1M+ documents/month), security (AES-256), reliability (99.9% uptime)

TASK

Design comprehensive system architecture including microservices design, AI/ML pipeline, data layer, integration patterns, security framework, and cloud-native deployment.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] Architecture supports all functional requirements from FRD - [ ] Performance requirements from NFRD addressed through scalable design - [ ] Security requirements implemented through comprehensive security architecture - [ ] AI/ML pipeline designed for accuracy targets and continuous learning

Validation Criteria: - [ ] Architecture validated with system architects and technical leads - [ ] Microservices design validated with development teams - [ ] AI/ML pipeline validated with data scientists and ML engineers - [ ] Security architecture validated with cybersecurity experts

EXIT CRITERIA

- âœ… Complete system architecture ready for high-level design development
- âœ… Microservices architecture with clear service boundaries
- âœ… AI/ML pipeline designed for accuracy and performance
- âœ… Security framework established for data protection

---

1. System Architecture Overview

1.1 High-Level Architecture

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                           PRESENTATION LAYER                               â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Web Portal â”‚ Mobile App â”‚ IDE Plugins â”‚ CLI Tools â”‚ Admin Console         â”‚
â”‚ (React.js) â”‚ (React     â”‚ (VS Code)   â”‚ (Node.js) â”‚ (Angular)             â”‚
â”‚            â”‚ Native)    â”‚             â”‚           â”‚                       â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                    â”‚
                                    â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                            API GATEWAY                                     â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚              Kong API Gateway + Ambassador Edge Stack                      â”‚
â”‚ Authentication â€¢ Rate Limiting â€¢ Load Balancing â€¢ Monitoring           â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                    â”‚
                                    â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                         MICROSERVICES LAYER                                â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Document     â”‚ OCR         â”‚ Classification â”‚ Information  â”‚ Integration   â”‚
â”‚ Ingestion    â”‚ Processing  â”‚ Service        â”‚ Extraction   â”‚ Manager       â”‚
â”‚ Service      â”‚ Service     â”‚ (Python/       â”‚ Service      â”‚ (Node.js/     â”‚
â”‚ (Node.js)    â”‚ (Python)    â”‚ FastAPI)       â”‚ (Python)     â”‚ Express)      â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Workflow     â”‚ Notificationâ”‚ Analytics      â”‚ User         â”‚ Configuration â”‚
â”‚ Orchestrator â”‚ Service     â”‚ Service        â”‚ Management   â”‚ Service       â”‚
â”‚ (Temporal)   â”‚ (Node.js)   â”‚ (Python)       â”‚ (Node.js)    â”‚ (Node.js)     â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                    â”‚
                                    â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                           AI/ML PIPELINE                                   â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Model        â”‚ Model       â”‚ Feature       â”‚ ML           â”‚ Model         â”‚
â”‚ Training     â”‚ Serving     â”‚ Store         â”‚ Monitoring   â”‚ Registry      â”‚
â”‚ (Kubeflow)   â”‚ (TorchServe)â”‚ (Feast)       â”‚ (MLflow)     â”‚ (MLflow)      â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ OCR Models   â”‚ Classificationâ”‚ NER Models   â”‚ Layout       â”‚ Quality       â”‚
â”‚ (Tesseract,  â”‚ Models        â”‚ (spaCy,      â”‚ Analysis     â”‚ Assessment    â”‚
â”‚ PaddleOCR)   â”‚ (BERT)        â”‚ Transformers)â”‚ (LayoutLM)   â”‚ Models        â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                    â”‚
                                    â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                             DATA LAYER                                     â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ PostgreSQL   â”‚ Elasticsearchâ”‚ Redis Cache  â”‚ MongoDB      â”‚ MinIO         â”‚
â”‚ (Metadata,   â”‚ (Search,      â”‚ (Session,    â”‚ (Documents,  â”‚ (Object       â”‚
â”‚ Users)       â”‚ Analytics)    â”‚ Cache)       â”‚ Results)     â”‚ Storage)      â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ InfluxDB     â”‚ Apache Kafka â”‚ RabbitMQ     â”‚ Apache       â”‚ Backup        â”‚
â”‚ (Metrics)    â”‚ (Events)     â”‚ (Tasks)      â”‚ Airflow      â”‚ Solutions     â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                    â”‚
                                    â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                        INTEGRATION LAYER                                   â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Cloud Storageâ”‚ Enterprise   â”‚ Workflow     â”‚ Notification â”‚ External APIs â”‚
â”‚ (AWS S3,     â”‚ Systems      â”‚ Systems      â”‚ Systems      â”‚ (Third-party  â”‚
â”‚ Azure Blob)  â”‚ (SAP, Oracle)â”‚ (Power       â”‚ (Email,      â”‚ Services)     â”‚
â”‚              â”‚              â”‚ Automate)    â”‚ Slack)       â”‚               â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
                                    â”‚
                                    â–¼
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                      INFRASTRUCTURE LAYER                                  â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚                    Kubernetes Cluster (Multi-Cloud)                        â”‚
â”‚ â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â” â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â” â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â” â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€
â”‚ â”‚   AWS EKS   â”‚ â”‚  Azure AKS  â”‚ â”‚ Google GKE  â”‚ â”‚On-Premises  â”‚         â”‚
â”‚ â”‚ (Primary)   â”‚ â”‚(Secondary)  â”‚ â”‚ (Tertiary)  â”‚ â”‚  (Hybrid)   â”‚         â”‚
â”‚ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜ â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€
```

```
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
â”‚ Monitoring   â”‚ Logging      â”‚ Security     â”‚ CI/CD        â”‚ Service Mesh  â”‚
â”‚ (Prometheus, â”‚ (ELK Stack)  â”‚ (Istio, OPA) â”‚ (GitLab CI)  â”‚ (Istio)       â”‚
â”‚ Grafana)     â”‚              â”‚              â”‚              â”‚               â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â
```

2. Microservices Architecture

2.1 Core Processing Services

Document Ingestion Service (Node.js/Express)

- Multi-format document upload and validation
- Document preprocessing and optimization
- Cloud storage integration
- Batch processing coordination

OCR Processing Service (Python/FastAPI)

- Advanced optical character recognition
- Multi-language text extraction
- Layout analysis and structure detection
- Quality assessment and confidence scoring

Document Classification Service (Python/FastAPI)

- Intelligent document type classification
- Content-based categorization
- Custom model training and deployment
- Classification confidence scoring

Information Extraction Service (Python/FastAPI)

- Named entity recognition and extraction
- Table and form data extraction
- Custom entity extraction
- Data validation and quality assessment

Integration Manager Service (Node.js/Express)

- Enterprise system integration
- API gateway management
- Data transformation and mapping
- Connection monitoring and health checks

2.2 Supporting Services

Workflow Orchestrator (Temporal)

- Document processing workflow coordination
- Business process automation
- Error handling and retry logic
- Custom workflow definition

User Management Service (Node.js/Express)

- Authentication and authorization
- Role-based access control
- User profile management
- Session management

Analytics Service (Python/FastAPI)

- Processing performance analytics
- Business intelligence reporting
- Real-time dashboards
- Custom report generation

3. AI/ML Pipeline Architecture

3.1 Model Training Pipeline

- Data ingestion and preprocessing
- Distributed training with Kubeflow
- Model validation and testing
- Automated deployment and versioning

3.2 Model Serving Infrastructure

- High-performance inference with TorchServe
- Auto-scaling based on demand
- Model ensemble serving
- GPU acceleration support

3.3 Feature Store (Feast)

- Centralized feature management
- Real-time and batch feature serving
- Feature versioning and lineage
- Data quality monitoring

4. Data Architecture

4.1 Multi-Database Strategy

- **PostgreSQL**: User data, metadata, audit logs
- **Elasticsearch**: Search, analytics, content indexing
- **MongoDB**: Document content, processing results
- **Redis**: Caching, session management
- **InfluxDB**: Time-series metrics and monitoring

4.2 Data Pipeline

- **Apache Kafka**: Event streaming and messaging
- **RabbitMQ**: Task queues and job processing
- **Apache Airflow**: Batch processing orchestration

5. Security Architecture

5.1 Zero-Trust Security Model

- Multi-factor authentication (MFA)
- Role-based access control (RBAC)
- API security with OAuth 2.0
- Network segmentation and micro-segmentation

5.2 Data Protection

- AES-256 encryption at rest
- TLS 1.3 encryption in transit
- Hardware Security Module (HSM) for key management
- Data classification and handling policies

5.3 Compliance Framework

- SOC 2 Type II compliance
- GDPR compliance for EU data
- HIPAA compliance for healthcare
- Comprehensive audit logging

6. Integration Patterns

6.1 API Gateway (Kong)

- API versioning and lifecycle management
- Rate limiting and throttling
- Authentication and authorization
- Request/response transformation

6.2 Enterprise Connectors

- ERP systems (SAP, Oracle, Microsoft Dynamics)
- CRM platforms (Salesforce, HubSpot)
- Document management (SharePoint, Box)
- Workflow engines (Power Automate, Zapier)

7. Deployment Architecture

7.1 Kubernetes Deployment

- Multi-cloud deployment (AWS, Azure, GCP)
- Auto-scaling with HPA and VPA
- Service mesh with Istio
- GitOps with ArgoCD

7.2 Monitoring and Observability

- **Prometheus/Grafana**: Metrics and monitoring
- **ELK Stack**: Centralized logging
- **Jaeger**: Distributed tracing
- **AlertManager**: Intelligent alerting

This architecture provides a scalable, secure, and reliable foundation for the document intelligence platform, supporting all functional and non-functional requirements while enabling continuous improvement and enterprise integration. # High Level Design (HLD) ## Document Intelligence and Processing Platform - AI-Powered Document Processing System

Building upon README, PRD, FRD, NFRD, and AD foundations for detailed component specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… PRD completed with business objectives, user personas, success metrics, and core features
- âœ… FRD completed with 42 detailed functional requirements across 5 system modules
- âœ… NFRD completed with performance (<30s processing), scalability (1M+ documents/month), security (AES-256), reliability (99.9% uptime)
- âœ… AD completed with microservices architecture, AI/ML pipeline, data layer, integration patterns, and deployment strategy

TASK

Develop detailed high-level design specifications including component interfaces, data models, API specifications, processing workflows, AI/ML model architectures, and system integration patterns that enable implementation of all functional requirements with specified performance and reliability targets.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All AD components detailed with interfaces, dependencies, and data flows - [ ] API specifications cover all FRD functional requirements with request/response models - [ ] Data models support all document types and processing workflows - [ ] AI/ML components designed for 95% accuracy targets and continuous learning - [ ] Processing workflows handle all document types with <30s processing time - [ ] Integration patterns support all enterprise systems and cloud platforms

Validation Criteria: - [ ] Component specifications validated with software architects and development teams - [ ] API designs validated through contract testing and integration scenarios - [ ] Data models validated with database administrators and data engineers - [ ] AI/ML architectures validated with data scientists and ML engineers - [ ] Processing workflows validated with business analysts and domain experts - [ ] Integration patterns validated with enterprise architects and IT administrators

EXIT CRITERIA

- âœ… Complete high-level design ready for low-level implementation design
- âœ… Detailed component specifications with interfaces and dependencies
- âœ… Comprehensive API specifications for all service interactions
- âœ… Data models supporting all functional and performance requirements
- âœ… Foundation prepared for detailed implementation design and development

---

Reference to Previous Documents

This HLD builds upon **README**, **PRD**, **FRD**, **NFRD**, and **AD** foundations: - **README Technical Approach** â†' Detailed component design implementing microservices and AI/ML pipeline - **PRD Success Metrics** â†' Component specifications supporting 300% ROI and 95% accuracy targets - **PRD User Personas** â†' Interface designs tailored for document processors, IT administrators, and business analysts - **FRD Functional Requirements** â†' Detailed implementation of all 42 functional requirements across components - **NFRD Performance Requirements** â†' Component design meeting <30s processing, 1000+ concurrent users, 99.9% uptime - **AD System Architecture** â†' Detailed specification of all architectural components and their interactions

1. Component Architecture and Interfaces

1.1 Document Ingestion Service

1.1.1 Component Overview

Technology Stack: Node.js 18+, Express.js 4.18+, Multer 1.4+, Sharp 0.32+ **Primary Responsibilities**: - Multi-format document upload handling with validation - Document preprocessing and optimization - Metadata extraction and enrichment - Cloud storage integration and management - Batch processing coordination and status tracking

1.1.2 Service Interface Specification

```
// Document Ingestion Service API Interface
interface DocumentIngestionAPI {
  // Document upload endpoints
  uploadDocument(request: DocumentUploadRequest): Promise<DocumentUploadResponse>;
  uploadBatch(request: BatchUploadRequest): Promise<BatchUploadResponse>;

  // Document management endpoints
  getDocumentStatus(documentId: string): Promise<DocumentStatusResponse>;
  getDocumentMetadata(documentId: string): Promise<DocumentMetadataResponse>;
  deleteDocument(documentId: string): Promise<DeleteDocumentResponse>;

  // Preprocessing endpoints
  preprocessDocument(documentId: string, options: PreprocessingOptions): Promise<PreprocessingResponse>;
  getPreprocessingStatus(documentId: string): Promise<PreprocessingStatusResponse>;
}

// Request/Response Models
interface DocumentUploadRequest {
  file: Buffer | Stream;
  filename: string;
  contentType: string;
  metadata?: DocumentMetadata;
  processingOptions?: ProcessingOptions;
  organizationId: string;
  userId: string;
}

interface DocumentUploadResponse {
  documentId: string;
  status: 'uploaded' | 'validating' | 'preprocessing' | 'ready';
  uploadUrl?: string;
  estimatedProcessingTime: number;
  supportedFormats: string[];
}

interface DocumentMetadata {
  title?: string;
  description?: string;
  tags?: string[];
  category?: string;
  confidentialityLevel: 'public' | 'internal' | 'confidential' | 'restricted';
  retentionPeriod?: number;
  customFields?: Record<string, any>;
}
```

1.1.3 Internal Component Architecture

```
// Core Components
class DocumentValidator {
  validateFormat(file: Buffer, contentType: string): ValidationResult;
  validateSize(file: Buffer, maxSize: number): ValidationResult;
  validateIntegrity(file: Buffer): ValidationResult;
  scanForMalware(file: Buffer): Promise<SecurityScanResult>;
}

class DocumentPreprocessor {
  optimizeImage(image: Buffer): Promise<Buffer>;
  correctOrientation(image: Buffer): Promise<Buffer>;
  enhanceQuality(image: Buffer): Promise<Buffer>;
  extractMetadata(file: Buffer): Promise<FileMetadata>;
}

class CloudStorageManager {
  uploadToStorage(file: Buffer, path: string): Promise<StorageResult>;
  generateSignedUrl(path: string, expiration: number): Promise<string>;
  deleteFromStorage(path: string): Promise<boolean>;
  copyBetweenStorages(sourcePath: string, destPath: string): Promise<boolean>;
}
```

1.1.4 Data Flow and Processing Workflow

```
Document Upload Workflow:
  1. Request Validation:
     - Authenticate user and validate permissions
     - Validate file format and size constraints
     - Check organization quotas and limits

  2. File Processing:
     - Virus scanning and security validation
     - File integrity verification
     - Metadata extraction and enrichment

  3. Preprocessing:
     - Image optimization and enhancement
     - Orientation correction and deskewing
     - Format conversion if required

  4. Storage and Indexing:
     - Upload to cloud storage with encryption
     - Create database records with metadata
     - Generate search index entries

  5. Event Publication:
     - Publish document.uploaded event
     - Trigger downstream processing workflows
     - Send status notifications to users
```

1.2 OCR Processing Service

1.2.1 Component Overview

Technology Stack: Python 3.11+, FastAPI 0.104+, Tesseract 5.3+, PaddleOCR 2.7+, TrOCR **Primary Responsibilities**: - Advanced optical character recognition with multiple engines - Multi-language text extraction and recognition - Layout analysis and document structure detection - Handwriting recognition and processing - Quality assessment and confidence scoring

1.2.2 Service Interface Specification

```
# OCR Processing Service API Interface
from typing import List, Dict, Optional, Union
from pydantic import BaseModel, Field
from enum import Enum

class OCREngine(str, Enum):
    TESSERACT = "tesseract"
    PADDLEOCR = "paddleocr"
    TROCR = "trocr"
    ENSEMBLE = "ensemble"

class OCRRequest(BaseModel):
    document_id: str = Field(..., description="Document identifier")
    pages: Optional[List[int]] = Field(None, description="Specific pages to process")
    language: str = Field("auto", description="Language code or 'auto' for detection")
    engine: OCREngine = Field(OCREngine.ENSEMBLE, description="OCR engine to use")
    options: Optional[Dict[str, any]] = Field(None, description="Engine-specific options")

class OCRResponse(BaseModel):
    document_id: str
    processing_time: float
    total_pages: int
    pages_processed: int
    overall_confidence: float
    text_blocks: List[TextBlock]
    layout_analysis: LayoutAnalysis
    quality_metrics: QualityMetrics

class TextBlock(BaseModel):
    page_number: int
    block_id: str
    text: str
    confidence: float
    bounding_box: BoundingBox
    language: str
    formatting: TextFormatting

class LayoutAnalysis(BaseModel):
    document_type: str
    reading_order: List[str]
    regions: List[DocumentRegion]
    tables: List[TableStructure]
    images: List[ImageRegion]

# Core OCR Processing Classes
class OCREngineManager:
    def __init__(self):
        self.engines = {
            'tesseract': TesseractEngine(),
            'paddleocr': PaddleOCREngine(),
            'trocr': TrOCREngine()
        }

    async def process_document(self, request: OCRRequest) -> OCRResponse:
        # Engine selection and processing logic
        pass

    async def ensemble_processing(self, document: bytes, options: Dict) -> OCRResponse:
        # Multi-engine ensemble processing
        pass

class LayoutAnalyzer:
    async def analyze_layout(self, image: bytes) -> LayoutAnalysis:
        # Document layout analysis using LayoutLM
        pass

    async def detect_tables(self, image: bytes) -> List[TableStructure]:
        # Table detection and structure analysis
        pass

    async def extract_reading_order(self, layout: LayoutAnalysis) -> List[str]:
        # Determine optimal reading order
        pass
```

1.2.3 AI/ML Model Integration

```
# OCR Model Management
class OCRModelManager:
    def __init__(self):
        self.models = {
            'tesseract': self.load_tesseract_models(),
            'paddleocr': self.load_paddleocr_models(),
            'trocr': self.load_trocr_models(),
            'layout_analysis': self.load_layout_models()
        }

    def load_tesseract_models(self) -> Dict[str, any]:
        # Load Tesseract language models
        return {
            'eng': tesseract.load_model('eng'),
            'spa': tesseract.load_model('spa'),
            'fra': tesseract.load_model('fra'),
            'deu': tesseract.load_model('deu'),
            'chi_sim': tesseract.load_model('chi_sim'),
            'jpn': tesseract.load_model('jpn'),
            'ara': tesseract.load_model('ara')
        }

    def load_trocr_models(self) -> Dict[str, any]:
        # Load TrOCR transformer models
        return {
            'base': transformers.TrOCRProcessor.from_pretrained('microsoft/trocr-base-printed'),
            'large': transformers.TrOCRProcessor.from_pretrained('microsoft/trocr-large-printed'),
            'handwritten': transformers.TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
        }

# Quality Assessment Component
class OCRQualityAssessor:
    def assess_text_quality(self, text: str, confidence_scores: List[float]) -> QualityMetrics:
        # Assess OCR output quality
        return QualityMetrics(
            character_accuracy=self.calculate_character_accuracy(text, confidence_scores),
            word_accuracy=self.calculate_word_accuracy(text, confidence_scores),
            line_accuracy=self.calculate_line_accuracy(text, confidence_scores),
            overall_confidence=statistics.mean(confidence_scores),
            quality_score=self.calculate_quality_score(text, confidence_scores)
        )

    def detect_potential_errors(self, text: str, confidence_scores: List[float]) -> List[PotentialError]:
        # Identify likely OCR errors for manual review
        pass
```

1.3 Document Classification Service

1.3.1 Component Overview

Technology Stack: Python 3.11+, FastAPI 0.104+, Transformers 4.35+, scikit-learn 1.3+ **Primary Responsibilities**: - Intelligent document type classification using ML models - Content-based categorization and labeling - Custom model training and deployment - Hierarchical classification support - Classification confidence scoring and validation

1.3.2 Service Interface Specification

```
# Document Classification Service API
class ClassificationRequest(BaseModel):
    document_id: str = Field(..., description="Document identifier")
    text_content: str = Field(..., description="Extracted text content")
    metadata: Optional[DocumentMetadata] = Field(None, description="Document metadata")
    classification_type: str = Field("standard", description="Classification model type")
    confidence_threshold: float = Field(0.8, description="Minimum confidence threshold")

class ClassificationResponse(BaseModel):
    document_id: str
    predictions: List[ClassificationPrediction]
    processing_time: float
    model_version: str
    confidence_distribution: Dict[str, float]

class ClassificationPrediction(BaseModel):
    category: str
    subcategory: Optional[str]
    confidence: float
    probability_distribution: Dict[str, float]
    explanation: ClassificationExplanation

class ClassificationExplanation(BaseModel):
    key_features: List[str]
    decision_factors: Dict[str, float]
    similar_documents: List[str]
    confidence_factors: List[str]

# Core Classification Components
class DocumentClassifier:
    def __init__(self):
        self.models = {
            'bert_classifier': self.load_bert_model(),
            'roberta_classifier': self.load_roberta_model(),
            'custom_models': self.load_custom_models()
        }
        self.feature_extractor = FeatureExtractor()
        self.ensemble_manager = EnsembleManager()

    async def classify_document(self, request: ClassificationRequest) -> ClassificationResponse:
        # Extract features from document content
        features = await self.feature_extractor.extract_features(
            request.text_content,
            request.metadata
        )

        # Run ensemble classification
        predictions = await self.ensemble_manager.predict(features, request.classification_type)

        # Generate explanations
        explanations = await self.generate_explanations(features, predictions)

        return ClassificationResponse(
            document_id=request.document_id,
            predictions=predictions,
            processing_time=time.time() - start_time,
            model_version=self.get_model_version(),
            confidence_distribution=self.calculate_confidence_distribution(predictions)
        )

class FeatureExtractor:
    def __init__(self):
        self.text_vectorizer = TfidfVectorizer(max_features=10000)
        self.bert_tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
        self.bert_model = AutoModel.from_pretrained('bert-base-uncased')

    async def extract_features(self, text: str, metadata: Optional[DocumentMetadata]) -> FeatureVector:
        # Extract comprehensive features for classification
        features = {
            'text_features': self.extract_text_features(text),
            'semantic_features': await self.extract_semantic_features(text),
            'metadata_features': self.extract_metadata_features(metadata),
            'structural_features': self.extract_structural_features(text)
        }
        return FeatureVector(**features)
```

1.4 Information Extraction Service

1.4.1 Component Overview

Technology Stack: Python 3.11+, FastAPI 0.104+, spaCy 3.7+, Transformers 4.35+ **Primary Responsibilities**: - Named entity recognition and extraction - Table and form data extraction - Custom entity extraction and training - Data validation and quality assessment - Structured output generation and formatting

1.4.2 Service Interface Specification

```
# Information Extraction Service API
```

```
class ExtractionRequest(BaseModel):
    document_id: str = Field(..., description="Document identifier")
    text_content: str = Field(..., description="OCR extracted text")
    document_type: str = Field(..., description="Classified document type")
    layout_analysis: LayoutAnalysis = Field(..., description="Document layout information")
    extraction_config: ExtractionConfig = Field(..., description="Extraction configuration")

class ExtractionResponse(BaseModel):
    document_id: str
    extracted_entities: List[ExtractedEntity]
    extracted_tables: List[ExtractedTable]
    extracted_forms: List[ExtractedForm]
    validation_results: ValidationResults
    processing_time: float
    confidence_metrics: ConfidenceMetrics

class ExtractedEntity(BaseModel):
    entity_type: str
    entity_value: str
    confidence: float
    location: BoundingBox
    context: str
    validation_status: str
    normalized_value: Optional[str]

class ExtractedTable(BaseModel):
    table_id: str
    headers: List[str]
    rows: List[List[str]]
    confidence: float
    location: BoundingBox
    table_type: str
    validation_status: str

# Core Extraction Components
class EntityExtractor:
    def __init__(self):
        self.nlp_models = {
            'en': spacy.load('en_core_web_lg'),
            'es': spacy.load('es_core_news_lg'),
            'fr': spacy.load('fr_core_news_lg'),
            'de': spacy.load('de_core_news_lg')
        }
        self.custom_models = self.load_custom_models()
        self.transformers_pipeline = pipeline('ner', model='dbmdz/bert-large-cased-finetuned-conll03-english')

    async def extract_entities(self, text: str, document_type: str, language: str = 'en') -> List[ExtractedEntity]:
        # Multi-model entity extraction
        spacy_entities = self.extract_with_spacy(text, language)
        transformer_entities = self.extract_with_transformers(text)
        custom_entities = await self.extract_custom_entities(text, document_type)

        # Merge and deduplicate entities
        merged_entities = self.merge_entity_results(
            spacy_entities,
            transformer_entities,
            custom_entities
        )

        return merged_entities

class TableExtractor:
    def __init__(self):
        self.table_detection_model = self.load_table_detection_model()
        self.table_structure_model = self.load_table_structure_model()

    async def extract_tables(self, image: bytes, layout_analysis: LayoutAnalysis) -> List[ExtractedTable]:
        # Detect table regions
        table_regions = await self.detect_table_regions(image, layout_analysis)

        extracted_tables = []
        for region in table_regions:
            # Extract table structure and content
            table_structure = await self.extract_table_structure(region)
            table_content = await self.extract_table_content(region, table_structure)

            extracted_table = ExtractedTable(
                table_id=self.generate_table_id(),
                headers=table_structure.headers,
                rows=table_content.rows,
                confidence=table_structure.confidence,
                location=region.bounding_box,
                table_type=self.classify_table_type(table_structure),
                validation_status='pending'
            )

            extracted_tables.append(extracted_table)

        return extracted_tables
```

2. API Specifications

2.1 RESTful API Design Standards

2.1.1 API Gateway Configuration

```
# Kong API Gateway Configuration
api_gateway:
  version: "3.4"
  services:
    - name: document-ingestion
      url: http://document-ingestion-service:3000
      routes:
        - name: document-upload
          paths: ["/api/v1/documents"]
          methods: ["POST", "GET", "DELETE"]
        - name: batch-upload
          paths: ["/api/v1/documents/batch"]
          methods: ["POST"]

    - name: ocr-processing
      url: http://ocr-processing-service:8000
      routes:
        - name: ocr-process
          paths: ["/api/v1/ocr"]
          methods: ["POST"]
    - name: classification
      url: http://classification-service:8000
      routes:
        - name: classify
          paths: ["/api/v1/classification"]
          methods: ["POST"]

    - name: extraction
      url: http://extraction-service:8000
      routes:
        - name: extract
          paths: ["/api/v1/extraction"]
          methods: ["POST"]

  plugins:
    - name: rate-limiting
      config:
        minute: 100
        hour: 1000
        day: 10000

    - name: authentication
      config:
        type: oauth2
        scopes: ["read", "write", "admin"]

    - name: cors
      config:
        origins: ["*"]
        methods: ["GET", "POST", "PUT", "DELETE"]
        headers: ["Accept", "Content-Type", "Authorization"]
```

2.1.2 API Response Standards

```
// Standard API Response Format
interface APIResponse<T> {
  success: boolean;
  data?: T;
  error?: APIError;
  metadata: ResponseMetadata;
}

interface APIError {
  code: string;
  message: string;
  details?: Record<string, any>;
  timestamp: string;
  request_id: string;
}

interface ResponseMetadata {
  request_id: string;
  timestamp: string;
  processing_time: number;
  api_version: string;
  rate_limit: RateLimitInfo;
}

// Error Handling Standards
enum ErrorCodes {
  VALIDATION_ERROR = "VALIDATION_ERROR",
  AUTHENTICATION_ERROR = "AUTHENTICATION_ERROR",
  AUTHORIZATION_ERROR = "AUTHORIZATION_ERROR",
  PROCESSING_ERROR = "PROCESSING_ERROR",
  SYSTEM_ERROR = "SYSTEM_ERROR",
  RATE_LIMIT_EXCEEDED = "RATE_LIMIT_EXCEEDED"
}
```

3. Data Models and Schema Design

3.1 Core Data Models

3.1.1 Document Data Model

```
-- PostgreSQL Schema for Document Management
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL,
    user_id UUID NOT NULL,
    filename VARCHAR(255) NOT NULL,
    original_filename VARCHAR(255) NOT NULL,
    content_type VARCHAR(100) NOT NULL,
    file_size BIGINT NOT NULL,
    file_hash VARCHAR(64) NOT NULL UNIQUE,

    -- Document metadata
    title VARCHAR(500),
    description TEXT,
    tags JSONB DEFAULT '[]',
    category VARCHAR(100),
    confidentiality_level VARCHAR(20) NOT NULL DEFAULT 'internal',
    retention_period INTEGER,
    custom_fields JSONB DEFAULT '{}',

    -- Processing status
    processing_status VARCHAR(50) NOT NULL DEFAULT 'uploaded',
    processing_progress INTEGER DEFAULT 0,
    error_message TEXT,

    -- Storage information
    storage_provider VARCHAR(50) NOT NULL,
    storage_path VARCHAR(1000) NOT NULL,
    storage_region VARCHAR(50),

    -- Audit fields
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    processed_at TIMESTAMP WITH TIME ZONE,
    deleted_at TIMESTAMP WITH TIME ZONE,

    -- Constraints
    CONSTRAINT fk_documents_organization FOREIGN KEY (organization_id) REFERENCES organizations(id),
    CONSTRAINT fk_documents_user FOREIGN KEY (user_id) REFERENCES users(id),
    CONSTRAINT chk_confidentiality_level CHECK (confidentiality_level IN ('public', 'internal', 'confidential', 'restricted')),
    CONSTRAINT chk_processing_status CHECK (processing_status IN ('uploaded', 'preprocessing', 'processing', 'completed', 'failed'))
);
```

```
-- Indexes for performance
CREATE INDEX idx_documents_organization ON documents(organization_id, created_at DESC);
CREATE INDEX idx_documents_user ON documents(user_id, created_at DESC);
CREATE INDEX idx_documents_status ON documents(processing_status, created_at DESC);
CREATE INDEX idx_documents_category ON documents(category, created_at DESC);
CREATE INDEX idx_documents_tags ON documents USING GIN(tags);
CREATE INDEX idx_documents_custom_fields ON documents USING GIN(custom_fields);
```

3.1.2 Processing Results Data Model

```
-- OCR Results Table
CREATE TABLE ocr_results (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID NOT NULL,
    processing_engine VARCHAR(50) NOT NULL,
    processing_time DECIMAL(8,3) NOT NULL,

    -- OCR metrics
    total_pages INTEGER NOT NULL,
    pages_processed INTEGER NOT NULL,
    overall_confidence DECIMAL(5,4) NOT NULL,
    language_detected VARCHAR(10),

    -- Extracted content
    full_text TEXT NOT NULL,
    text_blocks JSONB NOT NULL DEFAULT '[]',
    layout_analysis JSONB NOT NULL DEFAULT '{}',

    -- Quality metrics
    quality_score DECIMAL(5,4),
    potential_errors JSONB DEFAULT '[]',

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_ocr_results_document FOREIGN KEY (document_id) REFERENCES documents(id) ON DELETE CASCADE
);

-- Classification Results Table
CREATE TABLE classification_results (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID NOT NULL,
    model_version VARCHAR(50) NOT NULL,
    processing_time DECIMAL(8,3) NOT NULL,

    -- Classification results
    primary_category VARCHAR(100) NOT NULL,
    subcategory VARCHAR(100),
    confidence DECIMAL(5,4) NOT NULL,
    probability_distribution JSONB NOT NULL,

    -- Explanation data
    key_features JSONB DEFAULT '[]',
    decision_factors JSONB DEFAULT '{}',

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_classification_results_document FOREIGN KEY (document_id) REFERENCES documents(id) ON DELETE CASCADE
);

-- Extraction Results Table
CREATE TABLE extraction_results (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID NOT NULL,
    extraction_type VARCHAR(50) NOT NULL, -- 'entities', 'tables', 'forms'

    -- Extracted data
    extracted_data JSONB NOT NULL,
    confidence_scores JSONB NOT NULL,
    validation_results JSONB DEFAULT '{}',

    -- Processing metadata
    processing_time DECIMAL(8,3) NOT NULL,
    model_versions JSONB NOT NULL,

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_extraction_results_document FOREIGN KEY (document_id) REFERENCES documents(id) ON DELETE CASCADE
);
```

This HLD provides comprehensive component specifications, API designs, and data models that enable implementation of all functional requirements while meeting performance and reliability targets specified in previous documents. # Low Level Design (LLD) ## Document Intelligence and Processing Platform - AI-Powered Document Processing System

Building upon README, PRD, FRD, NFRD, AD, and HLD foundations for implementation-ready specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… PRD completed with business objectives, user personas, success metrics, and core features
- âœ… FRD completed with 42 detailed functional requirements across 5 system modules
- âœ… NFRD completed with performance (<30s processing), scalability (1M+ documents/month), security (AES-256), reliability (99.9% uptime)
- âœ… AD completed with microservices architecture, AI/ML pipeline, data layer, integration patterns, and deployment strategy
- âœ… HLD completed with detailed component specifications, API designs, data models, and processing workflows

TASK

Develop implementation-ready low-level design specifications including detailed class structures, database implementations, API implementations, algorithm specifications, configuration files, and deployment scripts.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All HLD components implemented with detailed class structures and method signatures - [ ] Database schemas implemented with indexes, constraints, and optimization strategies - [ ] API implementations include request/response models, validation, error handling, and security - [ ] Algorithm implementations provide step-by-step logic for OCR, classification, and extraction

Validation Criteria: - [ ] Code structures validated with senior developers and technical leads - [ ] Database implementations validated with database administrators and performance engineers - [ ] API implementations validated through contract testing and integration validation - [ ] Algorithm implementations validated with data scientists and ML engineers

EXIT CRITERIA

- âœ… Complete implementation-ready class structures for all microservices
- âœ… Production-ready database schemas with performance optimizations
- âœ… Fully specified API implementations with comprehensive error handling
- âœ… Detailed algorithm implementations for all OCR, classification, and extraction components

---

1. Database Implementation

1.1 PostgreSQL Schema with Performance Optimizations

Documents Table with Partitioning

```
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL,
    user_id UUID NOT NULL,
    filename VARCHAR(255) NOT NULL,
    original_filename VARCHAR(255) NOT NULL,
    content_type VARCHAR(100) NOT NULL,
    file_size BIGINT NOT NULL CHECK (file_size > 0),
    file_hash VARCHAR(64) NOT NULL,

    -- Document metadata
    title VARCHAR(500),
    description TEXT,
    tags JSONB DEFAULT '[]' NOT NULL,
    category VARCHAR(100),
    confidentiality_level VARCHAR(20) NOT NULL DEFAULT 'internal',

    -- Processing status
    processing_status VARCHAR(50) NOT NULL DEFAULT 'uploaded',
    processing_progress INTEGER DEFAULT 0,
    error_message TEXT,

    -- Storage information
    storage_provider VARCHAR(50) NOT NULL,
    storage_path VARCHAR(1000) NOT NULL,
    storage_region VARCHAR(50),

    -- Quality metrics
    quality_score DECIMAL(5,4),

    -- Audit fields
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_documents_organization FOREIGN KEY (organization_id) REFERENCES organizations(id),
    CONSTRAINT fk_documents_user FOREIGN KEY (user_id) REFERENCES users(id)
) PARTITION BY RANGE (created_at);

-- Performance indexes
CREATE INDEX CONCURRENTLY idx_documents_org_created ON documents (organization_id, created_at DESC);
CREATE INDEX CONCURRENTLY idx_documents_status ON documents (processing_status, created_at DESC);
CREATE INDEX CONCURRENTLY idx_documents_tags ON documents USING GIN(tags);
```

OCR Results Table

```
CREATE TABLE ocr_results (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID NOT NULL,
    processing_engine VARCHAR(50) NOT NULL,
    processing_time DECIMAL(8,3) NOT NULL,

    -- OCR metrics
    total_pages INTEGER NOT NULL,
    overall_confidence DECIMAL(5,4) NOT NULL,
    language_detected VARCHAR(10),

    -- Extracted content
    full_text TEXT NOT NULL,
    text_blocks JSONB NOT NULL DEFAULT '[]',
    layout_analysis JSONB NOT NULL DEFAULT '{}',

    -- Quality metrics
    quality_score DECIMAL(5,4),
    potential_errors JSONB DEFAULT '[]',

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_ocr_results_document FOREIGN KEY (document_id) REFERENCES documents(id) ON DELETE CASCADE
) PARTITION BY RANGE (created_at);

-- Performance indexes
CREATE INDEX CONCURRENTLY idx_ocr_results_document ON ocr_results (document_id, created_at DESC);
CREATE INDEX CONCURRENTLY idx_ocr_results_confidence ON ocr_results (overall_confidence DESC);
CREATE INDEX CONCURRENTLY idx_ocr_results_text_blocks ON ocr_results USING GIN(text_blocks);
```

2. Backend Service Implementation

2.1 Document Ingestion Service (Node.js/TypeScript)

```
import express, { Request, Response } from 'express';
import multer from 'multer';
import { v4 as uuidv4 } from 'uuid';
import { Pool } from 'pg';
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

interface DocumentUploadRequest {
  file: Express.Multer.File;
  metadata?: DocumentMetadata;
  organizationId: string;
  userId: string;
}

interface DocumentMetadata {
  title?: string;
  description?: string;
  tags?: string[];
  category?: string;
  confidentialityLevel: 'public' | 'internal' | 'confidential' | 'restricted';
}

export class DocumentIngestionService {
  private readonly dbPool: Pool;
  private readonly s3Client: S3Client;
  constructor(dbPool: Pool, s3Client: S3Client) {
    this.dbPool = dbPool;
    this.s3Client = s3Client;
  }

  async uploadDocument(request: DocumentUploadRequest): Promise<DocumentUploadResponse> {
    const documentId = uuidv4();

    try {
      // Step 1: Validate file
      const validation = await this.validateFile(request.file);
      if (!validation.isValid) {
        throw new Error(`Validation failed: ${validation.errors.join(', ')}`);
      }

      // Step 2: Upload to storage
      const storagePath = this.generateStoragePath(documentId, request.file.originalname);
      await this.uploadToStorage(request.file.buffer, storagePath);

      // Step 3: Create database record
      const documentRecord = await this.createDocumentRecord({
        id: documentId,
        organizationId: request.organizationId,
        userId: request.userId,
        filename: request.file.originalname,
        contentType: request.file.mimetype,
        fileSize: request.file.size,
        storagePath,
        metadata: request.metadata
      });

      // Step 4: Queue processing
      await this.queueProcessingJob(documentId);

      return {
        documentId,
        status: 'uploaded',
        estimatedProcessingTime: this.estimateProcessingTime(request.file.size)
      };

    } catch (error) {
      console.error(`Upload failed for ${documentId}:`, error);
      throw error;
    }
  }

  private async validateFile(file: Express.Multer.File): Promise<ValidationResult> {
    const errors: string[] = [];

    // Size validation
    if (file.size > 100 * 1024 * 1024) {
      errors.push('File too large');
    }

    // Type validation
    const allowedTypes = ['application/pdf', 'image/jpeg', 'image/png'];
    if (!allowedTypes.includes(file.mimetype)) {
      errors.push('Unsupported file type');
    }

    return {
      isValid: errors.length === 0,
      errors
    };
  }

  private generateStoragePath(documentId: string, filename: string): string {
    const date = new Date();
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');

    return `documents/${year}/${month}/${day}/${documentId}/${filename}`;
  }

  private async uploadToStorage(buffer: Buffer, path: string): Promise<void> {
    const command = new PutObjectCommand({
      Bucket: process.env.S3_BUCKET_NAME,
      Key: path,
      Body: buffer,
      ServerSideEncryption: 'AES256'
    });

    await this.s3Client.send(command);
  }

  private async createDocumentRecord(data: DocumentRecordData): Promise<DocumentRecord> {
    const query = `
      INSERT INTO documents (
        id, organization_id, user_id, filename, content_type,
        file_size, storage_path, storage_provider, title, description,
        tags, category, confidentiality_level
      ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12, $13)
      RETURNING *
    `;

    const values = [
      data.id,
      data.organizationId,
      data.userId,
      data.filename,
      data.contentType,
      data.fileSize,
      data.storagePath,
      'aws-s3',
      data.metadata?.title,
      data.metadata?.description,
      JSON.stringify(data.metadata?.tags || []),
      data.metadata?.category,
      data.metadata?.confidentialityLevel || 'internal'
    ];

    const result = await this.dbPool.query(query, values);
    return result.rows[0];
  }
}
```

2.2 OCR Processing Service (Python/FastAPI)

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import asyncio
import cv2
import numpy as np
import pytesseract
from paddleocr import PaddleOCR
import torch
from transformers import TrOCRProcessor, VisionEncoderDecoderModel

app = FastAPI(title="OCR Processing Service")

class OCRRequest(BaseModel):
    document_id: str
    image_data: bytes
    language: str = "auto"
    engine: str = "ensemble"

class OCRResponse(BaseModel):
    document_id: str
    text: str
    confidence: float
    processing_time: float
    blocks: list

class OCRProcessor:
    def __init__(self):
        self.tesseract_config = '--oem 3 --psm 6'
        self.paddleocr = PaddleOCR(use_angle_cls=True, lang='en')
        self.trocr_processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-printed')
        self.trocr_model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-printed')

    async def process_image(self, image_data: bytes, engine: str = "ensemble") -> OCRResponse:
        # Convert bytes to OpenCV image
        nparr = np.frombuffer(image_data, np.uint8)
        image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        if engine == "tesseract":
            return await self.process_with_tesseract(image)
        elif engine == "paddleocr":
            return await self.process_with_paddleocr(image)
        elif engine == "trocr":
            return await self.process_with_trocr(image)
        else:
            return await self.process_with_ensemble(image)

    async def process_with_tesseract(self, image) -> dict:
        # Preprocess image
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Apply denoising
        denoised = cv2.fastNlMeansDenoising(gray)

        # Extract text with confidence scores
        data = pytesseract.image_to_data(denoised, config=self.tesseract_config, output_type=pytesseract.Output.DICT)

        # Filter out low confidence text
        confidences = [int(conf) for conf in data['conf'] if int(conf) > 0]
        texts = [data['text'][i] for i, conf in enumerate(data['conf']) if int(conf) > 30]

        full_text = ' '.join(texts)
        avg_confidence = sum(confidences) / len(confidences) if confidences else 0

        return {
            'text': full_text,
            'confidence': avg_confidence / 100,
            'blocks': self.extract_text_blocks(data)
        }

    async def process_with_paddleocr(self, image) -> dict:
        result = self.paddleocr.ocr(image, cls=True)

        texts = []
        confidences = []
        blocks = []

        for line in result:
            for word_info in line:
                bbox, (text, confidence) = word_info
                texts.append(text)
                confidences.append(confidence)
                blocks.append({
                    'text': text,
                    'confidence': confidence,
                    'bbox': bbox
                })

        return {
            'text': ' '.join(texts),
            'confidence': sum(confidences) / len(confidences) if confidences else 0,
            'blocks': blocks
        }

    async def process_with_ensemble(self, image) -> dict:
        # Run multiple OCR engines in parallel
        tesseract_task = asyncio.create_task(self.process_with_tesseract(image))
        paddleocr_task = asyncio.create_task(self.process_with_paddleocr(image))

        tesseract_result, paddleocr_result = await asyncio.gather(tesseract_task, paddleocr_task)

        # Combine results using confidence weighting
        if tesseract_result['confidence'] > paddleocr_result['confidence']:
            return tesseract_result
        else:
            return paddleocr_result

ocr_processor = OCRProcessor()

@app.post("/process", response_model=OCRResponse)
async def process_document(request: OCRRequest):
    try:
        result = await ocr_processor.process_image(request.image_data, request.engine)

        return OCRResponse(
            document_id=request.document_id,
            text=result['text'],
            confidence=result['confidence'],
            processing_time=0.0,  # Calculate actual time
            blocks=result['blocks']
        )
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

3. Configuration Files

3.1 Kubernetes Deployment Configuration

```
# document-ingestion-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: document-ingestion-service
  namespace: document-intelligence
spec:
  replicas: 3
  selector:
    matchLabels:
      app: document-ingestion-service
  template:
    metadata:
      labels:
        app: document-ingestion-service
    spec:
      containers:
      - name: document-ingestion
        image: document-intelligence/ingestion-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: database-secrets
              key: postgresql-url
        - name: S3_BUCKET_NAME
          value: "document-intelligence-storage"
        - name: AWS_REGION
          value: "us-east-1"
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: document-ingestion-service
  namespace: document-intelligence
spec:
  selector:
    app: document-ingestion-service
  ports:
  - protocol: TCP
    port: 80
    targetPort: 3000
  type: ClusterIP
```

3.2 Environment Configuration

```
# config/production.yaml
server:
  port: 3000
  host: "0.0.0.0"
  cors:
    origin: ["https://document-intelligence.company.com"]
    credentials: true

database:
  postgresql:
    host: "${DATABASE_HOST}"
    port: 5432
    database: "${DATABASE_NAME}"
    username: "${DATABASE_USER}"
    password: "${DATABASE_PASSWORD}"
    ssl: true
    pool:
      min: 10
      max: 50

storage:
  aws:
    region: "${AWS_REGION}"
    bucket: "${S3_BUCKET_NAME}"
    encryption: "AES256"

processing:
  ocr:
    engines: ["tesseract", "paddleocr", "trocr"]
    default_engine: "ensemble"
    confidence_threshold: 0.7

  classification:
    model_path: "/app/models/classification"
    confidence_threshold: 0.8

  extraction:
    model_path: "/app/models/extraction"
    entity_types: ["PERSON", "ORG", "DATE", "MONEY"]

security:
  jwt:
    secret: "${JWT_SECRET}"
    expiresIn: "24h"

  encryption:
    algorithm: "aes-256-gcm"
    key: "${ENCRYPTION_KEY}"

monitoring:
  prometheus:
    enabled: true
    path: "/metrics"

  logging:
    level: "info"
    format: "json"
```

This LLD provides implementation-ready specifications with detailed class structures, database schemas, API implementations, and configuration files that enable direct development of the document intelligence platform. # Pseudocode ## Document Intelligence and Processing Platform - AI-Powered Document Processing System

Building upon README, PRD, FRD, NFRD, AD, HLD, and LLD foundations for executable algorithm specifications

ETVX Framework

ENTRY CRITERIA

- âœ… README completed with problem overview, technical approach, and expected outcomes
- âœ… PRD completed with business objectives, user personas, success metrics, and core features
- âœ… FRD completed with 42 detailed functional requirements across 5 system modules
- âœ… NFRD completed with performance (<30s processing), scalability (1M+ documents/month), security (AES-256), reliability (99.9% uptime)
- âœ… AD completed with microservices architecture, AI/ML pipeline, data layer, integration patterns, and deployment strategy
- âœ… HLD completed with detailed component specifications, API designs, data models, and processing workflows
- âœ… LLD completed with implementation-ready class structures, database schemas, API implementations, and configuration files

TASK

Develop executable pseudocode algorithms for all core system components including document ingestion, OCR processing, document classification, information extraction, integration workflows, and analytics systems.

VERIFICATION & VALIDATION

Verification Checklist: - [ ] All core algorithms implemented with step-by-step pseudocode - [ ] OCR processing algorithms specified with multi-engine ensemble logic - [ ] Classification algorithms include ML model inference and confidence scoring - [ ] Extraction algorithms cover entities, tables, and forms processing

Validation Criteria: - [ ] Pseudocode algorithms validated with software architects and senior developers - [ ] OCR algorithms validated with computer vision experts and ML engineers - [ ] Classification algorithms validated with data scientists and NLP specialists - [ ] Extraction algorithms validated with information retrieval experts

EXIT CRITERIA

- âœ… Complete executable pseudocode for all system components
- âœ… Algorithm specifications ready for direct implementation
- âœ… Performance optimization strategies documented
- âœ… Error handling and edge cases covered

---

1. Document Processing Pipeline

1.1 Main Document Upload Workflow

```
ALGORITHM: ProcessDocumentUpload
INPUT: DocumentUploadRequest (file, metadata, organizationId, userId)
OUTPUT: DocumentProcessingResult

BEGIN
    document_id = GenerateUniqueID()
    start_time = GetCurrentTimestamp()

    TRY
        // Step 1: Validate upload
        validation_result = ValidateDocumentUpload(file, metadata, organizationId)
        IF NOT validation_result.is_valid THEN
            RETURN CreateErrorResponse("VALIDATION_FAILED", validation_result.errors)
        END IF

        // Step 2: Check for duplicates
        file_hash = CalculateFileHash(file.buffer)
        existing_document = CheckDuplicateDocument(file_hash, organizationId)
        IF existing_document != NULL THEN
            RETURN CreateDuplicateResponse(existing_document)
        END IF

        // Step 3: Preprocess if image
        processed_file = file
        IF IsImageFile(file.content_type) THEN
            preprocessing_result = PreprocessImage(file.buffer)
            processed_file.buffer = preprocessing_result.buffer
        END IF

        // Step 4: Upload to storage
        storage_path = GenerateStoragePath(document_id, file.filename)
        upload_result = UploadToCloudStorage(processed_file.buffer, storage_path)

        // Step 5: Create database record
        document_record = CreateDocumentRecord(document_id, organizationId, userId, file, storage_path)

        // Step 6: Queue processing
        processing_job = QueueDocumentProcessingJob(document_id)

        // Step 7: Publish events
        PublishEvent("document.uploaded", document_record)

        RETURN DocumentProcessingResult(document_id, "uploaded", processing_job.id)

    CATCH Exception e
        LogError("Upload failed for " + document_id + ": " + e.message)
        THROW DocumentProcessingException("Upload failed", e)
    END TRY
END
```

1.2 Image Preprocessing Algorithm

```
ALGORITHM: PreprocessImage
INPUT: image_buffer
OUTPUT: PreprocessingResult

BEGIN
    TRY
        // Step 1: Load and analyze image
        image = LoadImageFromBuffer(image_buffer)
        metadata = ExtractImageMetadata(image)

        // Step 2: Orientation correction
        IF metadata.orientation != 1 THEN
            image = CorrectOrientation(image, metadata.orientation)
        END IF

        // Step 3: Resolution optimization
        target_dpi = 300
        IF metadata.dpi < target_dpi THEN
            scale_factor = target_dpi / metadata.dpi
            image = ResizeImage(image, scale_factor)
        END IF

        // Step 4: Quality enhancement
        IF DetectImageNoise(image) > 0.3 THEN
            image = ApplyNoiseReduction(image)
        END IF

        image = AdjustContrast(image, 1.2)
        image = AdjustBrightness(image, 1.1)

        // Step 5: Skew correction
        skew_angle = DetectSkewAngle(image)
        IF ABS(skew_angle) > 0.5 THEN
            image = CorrectSkew(image, skew_angle)
        END IF

        processed_buffer = ConvertImageToBuffer(image, "PNG")

        RETURN PreprocessingResult(processed_buffer, metadata)

    CATCH Exception e
        RETURN PreprocessingResult(image_buffer, {error: e.message})
    END TRY
END
```

2. OCR Processing Engine

2.1 Multi-Engine OCR Algorithm

```
ALGORITHM: ProcessOCRWithEnsemble
INPUT: document_id, image_data, language
OUTPUT: OCRResult

BEGIN
    start_time = GetCurrentTimestamp()

    TRY
        // Step 1: Prepare image
        preprocessed_image = PrepareImageForOCR(image_data)

        // Step 2: Language detection
        IF language == "auto" THEN
            language = DetectLanguage(preprocessed_image)
        END IF

        // Step 3: Select engines
        engines = ["tesseract", "paddleocr", "trocr"]

        // Step 4: Run engines in parallel
        ocr_tasks = []
        FOR EACH engine IN engines DO
            task = CreateAsyncTask(RunOCREngine, engine, preprocessed_image, language)
            ocr_tasks.ADD(task)
        END FOR

        engine_results = AwaitAll(ocr_tasks)

        // Step 5: Ensemble fusion
        ensemble_result = FuseOCRResults(engine_results)

        // Step 6: Layout analysis
        layout_analysis = AnalyzeDocumentLayout(preprocessed_image, ensemble_result.text_blocks)

        // Step 7: Quality assessment
        quality_metrics = AssessOCRQuality(ensemble_result, layout_analysis)

        // Step 8: Create result
        final_result = OCRResult(
            document_id,
            ensemble_result.full_text,
            ensemble_result.text_blocks,
            layout_analysis,
            quality_metrics,
            GetCurrentTimestamp() - start_time
        )

        // Step 9: Store results
        StoreOCRResults(final_result)
        PublishEvent("ocr.completed", final_result)

        RETURN final_result

    CATCH Exception e
        LogError("OCR failed for " + document_id + ": " + e.message)
        THROW OCRProcessingException("OCR processing failed", e)
    END TRY
END
```

3. Document Classification

3.1 Classification Algorithm

```
ALGORITHM: ClassifyDocument
INPUT: document_id, text_content, metadata
OUTPUT: ClassificationResult
```

```
BEGIN
    start_time = GetCurrentTimestamp()

    TRY
        // Step 1: Feature extraction
        features = ExtractDocumentFeatures(text_content, metadata)

        // Step 2: Load models
        models = LoadClassificationModels("standard")

        // Step 3: Run classification
        predictions = []
        FOR EACH model IN models DO
            prediction = RunClassificationModel(model, features)
            predictions.ADD(prediction)
        END FOR

        // Step 4: Ensemble voting
        ensemble_result = CombineClassificationPredictions(predictions)

        // Step 5: Generate explanations
        explanations = GenerateClassificationExplanations(ensemble_result, features)

        // Step 6: Create result
        classification_result = ClassificationResult(
            document_id,
            ensemble_result.primary_category,
            ensemble_result.confidence,
            ensemble_result.alternatives,
            explanations,
            GetCurrentTimestamp() - start_time
        )

        // Step 7: Store and publish
        StoreClassificationResults(classification_result)
        PublishEvent("classification.completed", classification_result)

        RETURN classification_result

    CATCH Exception e
        LogError("Classification failed for " + document_id + ": " + e.message)
        THROW ClassificationException("Classification failed", e)
    END TRY
END
```

4. Information Extraction

4.1 Information Extraction Algorithm

```
ALGORITHM: ExtractInformation
INPUT: document_id, text_content, document_type, layout_analysis
OUTPUT: ExtractionResult

BEGIN
    start_time = GetCurrentTimestamp()

    TRY
        // Step 1: Initialize extractors
        extractors = InitializeExtractors(document_type)

        // Step 2: Named entity recognition
        entities = ExtractNamedEntities(text_content, extractors.ner_models)

        // Step 3: Table extraction
        tables = []
        IF layout_analysis.has_tables THEN
            tables = ExtractTables(text_content, layout_analysis.table_regions)
        END IF

        // Step 4: Form extraction
        forms = []
        IF layout_analysis.has_forms THEN
            forms = ExtractFormFields(text_content, layout_analysis.form_regions)
        END IF

        // Step 5: Data validation
        validation_result = ValidateExtractedData(entities, tables, forms)

        // Step 6: Quality assessment
        quality_metrics = CalculateExtractionQuality(entities, tables, forms, validation_result)

        // Step 7: Create result
        extraction_result = ExtractionResult(
            document_id,
            entities,
            tables,
            forms,
            quality_metrics,
            validation_result,
            GetCurrentTimestamp() - start_time
        )

        // Step 8: Store and publish
        StoreExtractionResults(extraction_result)
        PublishEvent("extraction.completed", extraction_result)

        RETURN extraction_result

    CATCH Exception e
        LogError("Extraction failed for " + document_id + ": " + e.message)
        THROW ExtractionException("Information extraction failed", e)
    END TRY
END
```

5. Integration Workflows

5.1 Enterprise System Integration

```
ALGORITHM: IntegrateWithEnterpriseSystem
INPUT: integration_config, document_data, processing_results
OUTPUT: IntegrationResult

BEGIN
    TRY
        // Step 1: Validate integration config
        ValidateIntegrationConfig(integration_config)
        // Step 2: Transform data format
        transformed_data = TransformDataForSystem(document_data, processing_results, integration_config.mapping)

        // Step 3: Authenticate with target system
        auth_token = AuthenticateWithSystem(integration_config.credentials)

        // Step 4: Send data to target system
        response = SendDataToSystem(transformed_data, integration_config.endpoint, auth_token)

        // Step 5: Handle response
        IF response.success THEN
            LogInfo("Integration successful: " + response.message)
            RETURN IntegrationResult("SUCCESS", response.data)
        ELSE
            LogError("Integration failed: " + response.error)
            RETURN IntegrationResult("FAILED", response.error)
        END IF

    CATCH Exception e
        LogError("Integration error: " + e.message)
        RETURN IntegrationResult("ERROR", e.message)
    END TRY
END
```

6. Performance Optimization

6.1 Processing Optimization Algorithm

```
ALGORITHM: OptimizeProcessingPipeline
INPUT: document_characteristics, system_load
OUTPUT: OptimizedProcessingPlan

BEGIN
    // Step 1: Analyze document characteristics
    complexity_score = CalculateDocumentComplexity(document_characteristics)

    // Step 2: Check system resources
    available_resources = GetAvailableResources()
    current_load = GetCurrentSystemLoad()

    // Step 3: Select optimal processing path
    IF complexity_score < 0.3 AND current_load < 0.7 THEN
        processing_plan = CreateFastProcessingPlan()
    ELSE IF complexity_score > 0.7 OR current_load > 0.9 THEN
        processing_plan = CreateRobustProcessingPlan()
    ELSE
        processing_plan = CreateBalancedProcessingPlan()
    END IF

    // Step 4: Resource allocation
    processing_plan.resources = AllocateOptimalResources(
        complexity_score,
        available_resources,
        processing_plan.requirements
    )

    RETURN processing_plan
END
```

This comprehensive pseudocode provides executable algorithm specifications for all core components of the document intelligence platform, enabling direct implementation by development teams while ensuring all functional and non-functional requirements are met.