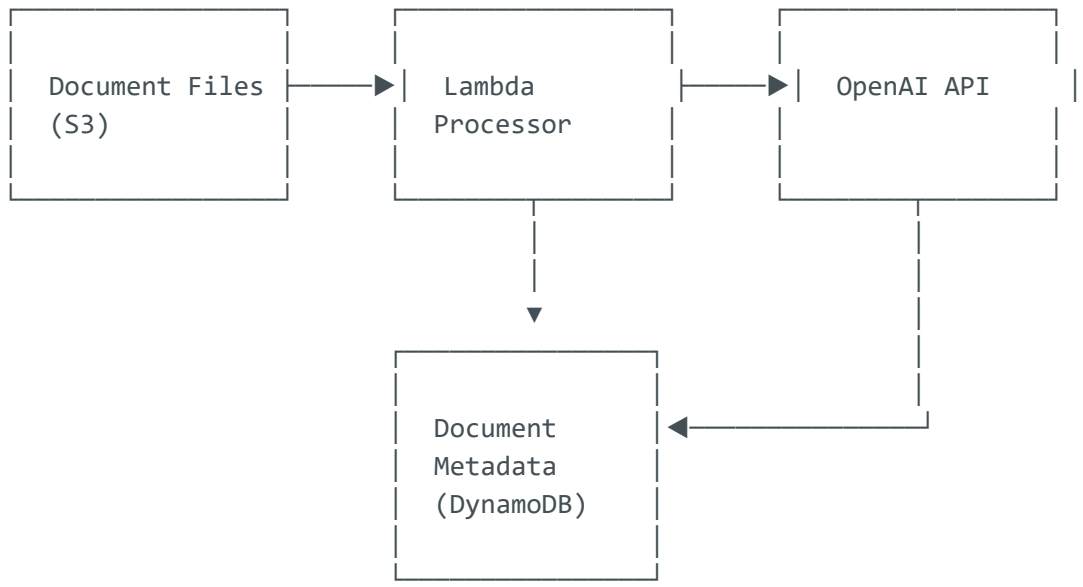
- AI Integration Architecture
  - AI Integration Overview
  - AI Capabilities
    - Document Understanding
    - Information Extraction
    - Document Analysis
  - Integration Patterns
    - Document Processing Workflow
    - Prompt Engineering
    - AI Service Layer
  - Implementation Details
    - OpenAI API Integration
    - Lambda Implementation
  - Model Selection and Usage
    - Model Selection Criteria
    - Token Usage Optimization
  - Error Handling and Quality Control
    - Error Handling
    - Quality Control
  - Cost Management
  - Performance Considerations
    - Latency Management
    - Scalability
  - Monitoring and Analytics
    - Operational Monitoring
    - Business Analytics
  - Future Enhancements
  - Configuration Management

AI Integration Architecture

This document outlines the AI integration architecture of the Document Processing Accelerator, detailing how OpenAI's capabilities are leveraged for intelligent document processing, the implementation patterns, and considerations for production deployments.

AI Integration Overview

The Document Processing Accelerator uses OpenAI's powerful language models to automate document understanding, extraction, and analysis tasks:



AI Capabilities

Document Understanding

- **Document Classification**: Automatically categorize documents (invoices, receipts, contracts, etc.)
- **Layout Analysis**: Understand document structure and formatting
- **Text Extraction**: Extract clean text from various formats including scanned PDFs

Information Extraction

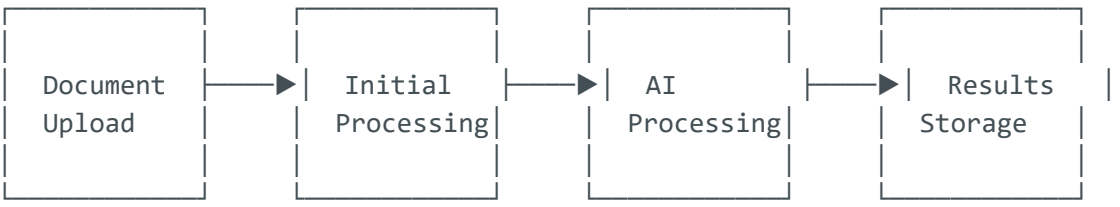
- **Named Entity Recognition**: Identify and extract entities like dates, names, addresses
- **Field Extraction**: Extract structured data from semi-structured documents
- **Table Extraction**: Convert tabular data into structured format

Document Analysis

- **Relationship Detection**: Identify relationships between entities
- **Anomaly Detection**: Flag unusual or missing information
- **Summarization**: Generate concise summaries of document content

Integration Patterns

Document Processing Workflow



1. **Document Upload**: User uploads document to S3 via pre-signed URL
2. **Initial Processing**: Lambda function triggers text extraction
3. **AI Processing**: Extracted text is sent to OpenAI API with appropriate prompts
4. **Results Storage**: Structured results stored in DynamoDB

Prompt Engineering

The system uses carefully engineered prompts to get high-quality results from the OpenAI models:

```
// Example prompt template for invoice processing
const invoicePrompt = `
You are an expert document analyzer specialized in invoices.
Extract the following information from the provided invoice text:
- Invoice Number
- Date
- Due Date
- Vendor Name
- Vendor Address
- Line Items (with quantities, unit prices, and total amounts)
- Subtotal
- Tax Amount
- Total Amount
```

```
Format your response as JSON with these exact field names. If a field isn't found,
set its value to null.

Invoice text:
${documentText}
`;
```

AI Service Layer

The AI integration is encapsulated in a dedicated service layer:

```
// AI service abstraction
export class DocumentAIService {
  private openaiClient: OpenAIClient;

  constructor(apiKey: string) {
    this.openaiClient = new OpenAIClient(apiKey);
  }

  // Process document based on its type
  async processDocument(documentText: string, documentType: string): Promise<any> {
    switch (documentType) {
      case 'INVOICE':
        return this.processInvoice(documentText);
      case 'RECEIPT':
        return this.processReceipt(documentText);
      case 'CONTRACT':
        return this.processContract(documentText);
      default:
        return this.processGenericDocument(documentText);
    }
  }

  // Document type-specific processing methods
  private async processInvoice(documentText: string): Promise<any> {
    const prompt = this.buildInvoicePrompt(documentText);
    const response = await this.openaiClient.complete({
      model: 'gpt-4',
      prompt,
      temperature: 0.1,
      max_tokens: 1000
    });

    return this.parseAIResponse(response);
  }

  // Additional helper methods
  private buildInvoicePrompt(text: string): string {
    // Template-based prompt engineering
  }
```

```
  private parseAIResponse(response: string): any {
    // Response parsing and validation
  }
}
```

Implementation Details

OpenAI API Integration

The system uses OpenAI's API with careful configuration:

```
// OpenAI client with retry logic
class OpenAIClient {
  private apiKey: string;
  private baseUrl = 'https://api.openai.com/v1';
  private maxRetries = 3;

  constructor(apiKey: string) {
    this.apiKey = apiKey;
  }

  async complete(params: OpenAICompletionParams): Promise<string> {
    let retries = 0;

    while (retries < this.maxRetries) {
      try {
        const response = await fetch(`${this.baseUrl}/completions`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${this.apiKey}`
          },
          body: JSON.stringify(params)
        });

        if (!response.ok) {
          const error = await response.json();
          throw new Error(`OpenAI API error: ${error.message}`);
        }

        const data = await response.json();
        return data.choices[0].text;
      } catch (error) {
        retries++;
        if (retries >= this.maxRetries) throw error;

        // Exponential backoff
        await new Promise(r => setTimeout(r, 1000 * Math.pow(2, retries)));
      }
```

```
    }
  }
}
```

Lambda Implementation

Document processing is handled by a dedicated Lambda function:

```
// Document processing Lambda handler
export const handler = async (event: S3Event): Promise<any> => {
  try {
    // Get document ID from S3 key
    const s3Key = event.Records[0].s3.object.key;
    const documentId = extractDocumentId(s3Key);

    // Retrieve document metadata from DynamoDB
    const documentMetadata = await documentService.getDocumentById(documentId);

    // Update status to PROCESSING
    await documentService.updateDocumentStatus(documentId, 'PROCESSING');

    // Get document content from S3
    const documentContent = await s3Service.getDocumentContent(s3Key);

    // Extract text from document
    const documentText = await textExtractionService.extractText(documentContent);

    // Process with AI
    const aiService = new DocumentAIService(process.env.OPENAI_API_KEY);
    const processedData = await aiService.processDocument(
      documentText,
      documentMetadata.documentType
    );

    // Store processing results
    await documentService.updateDocumentWithProcessedData(documentId,
processedData);

    // Update status to COMPLETED
    await documentService.updateDocumentStatus(documentId, 'COMPLETED');

    return {
      statusCode: 200,
      body: JSON.stringify({ message: 'Document processed successfully', id:
documentId })
    };
  } catch (error) {
    // Handle errors and update document status
    await documentService.updateDocumentStatus(documentId, 'ERROR');
    throw error;
```

```
  }
};
```

Model Selection and Usage

The Document Processing Accelerator uses different OpenAI models based on the complexity and needs of each document type:

Model Selection Criteria

Document Type	Model	Rationale
Invoices	GPT-4	Complex structure requiring advanced reasoning
Receipts	GPT-3.5	Simpler structure, cost-effective
Contracts	GPT-4	Complex legal language understanding required
Generic	GPT-3.5	General-purpose text analysis

Token Usage Optimization

To minimize costs and improve performance:

1. **Context Chunking**: Large documents are processed in chunks
2. **Targeted Prompts**: Specific prompts focused on exact information needs
3. **Response Formatting**: Structured JSON responses to reduce token usage

Error Handling and Quality Control

The AI integration includes robust error handling and quality control:

Error Handling

- **Retry Logic**: Automatic retry with exponential backoff for API failures
- **Fallback Models**: Ability to downgrade to simpler models if advanced models fail

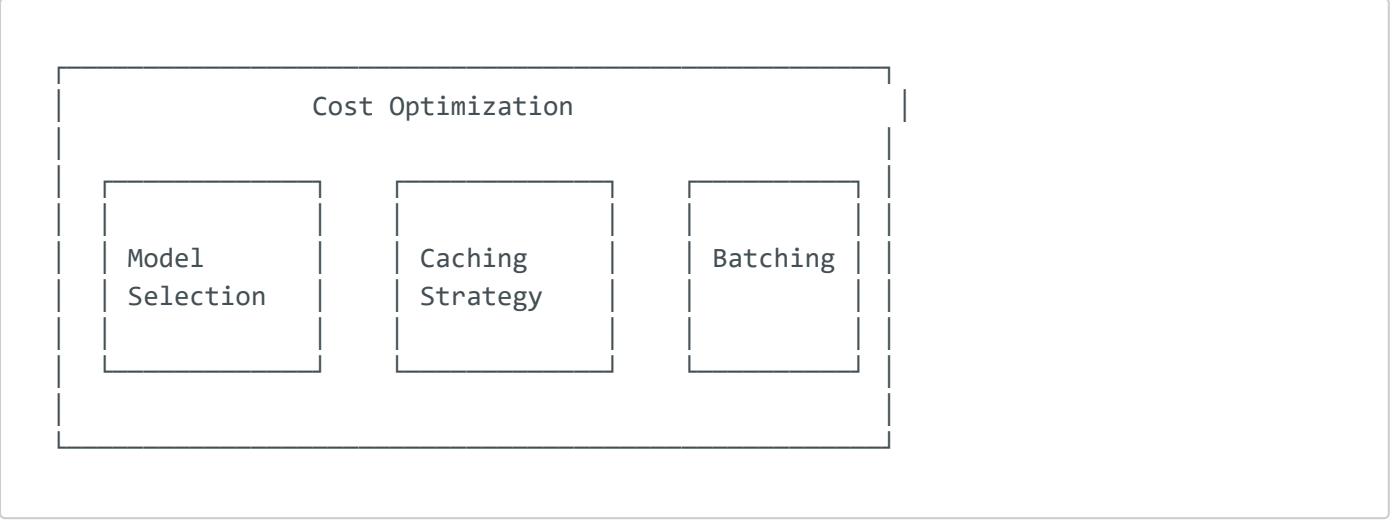
- **Error Classification**: Categorize errors as API failures, model limitations, or content issues

Quality Control

- **Schema Validation**: Validate AI-generated JSON against expected schemas
- **Confidence Scoring**: AI provides confidence level for extracted fields
- **Human Review**: Optional review process for low-confidence extractions

Cost Management

The system implements several cost optimization strategies:



1. **Tiered Model Usage**: Using more expensive models only when needed
2. **Result Caching**: Caching similar document processing results
3. **Batch Processing**: Combining multiple operations in single API calls
4. **Prompt Optimization**: Reducing prompt size without sacrificing quality

Performance Considerations

Latency Management

- **Asynchronous Processing**: Document processing happens asynchronously
- **Progress Tracking**: Real-time status updates for long-running processes
- **Priority Queue**: Optional priority processing for urgent documents

Scalability

- **Parallel Processing**: Multiple documents processed in parallel
- **Rate Limiting**: Careful API rate limit management
- **Queue Management**: SQS queue for processing backlog

Monitoring and Analytics

The AI integration includes comprehensive monitoring:

Operational Monitoring

- **Process Performance**: Track processing times, success rates, error rates
- **API Status**: Monitor OpenAI API availability and response times
- **Token Usage**: Track token consumption for budgeting and optimization

Business Analytics

- **Document Insights**: Aggregate document data for business intelligence
- **Processing Efficiency**: Track time and cost savings vs. manual processing
- **Accuracy Metrics**: Measure extraction accuracy when ground truth is available

Future Enhancements

Planned enhancements to the AI integration include:

1. **Fine-tuned Models**: Custom fine-tuning for specific document types
2. **Multi-modal Models**: Integration with models that process both text and images
3. **Self-improving System**: Using feedback to improve prompt engineering
4. **Alternative AI Providers**: Integration with additional AI services for resilience

Configuration Management

The AI integration is designed for flexible configuration:

```typescript
// Configuration structure
interface AIServiceConfig {
  defaultModel: string;
  modelConfigurations: {
    [documentType: string]: {
      model: string;
      temperature: number;
      maxTokens: number;
      promptTemplate: string;
    }
  };
  retryConfig: {
    maxRetries: number;
    initialDelay: number;
    maxDelay: number;
  };
}

// Example configuration
const aiServiceConfig: AIServiceConfig = {
  defaultModel: 'gpt-3.5-turbo',
  modelConfigurations: {
    INVOICE: {
      model: 'gpt-4',
      temperature: 0.1,
      maxTokens: 1000,
      promptTemplate: '...'
    },
    RECEIPT: {
      model: 'gpt-3.5-turbo',
      temperature: 0.2,
      maxTokens: 500,
      promptTemplate: '...'
    }
  },
  retryConfig: {
    maxRetries: 3,
    initialDelay: 1000,
    maxDelay: 8000
  }
};
```

This configuration can be updated without code changes, allowing for rapid experimentation and optimization.