perplexity

Comprehensive Execution Plan: Intelligent Document Processing System

Executive Summary

This execution plan delivers a **five-tier serverless document processing architecture** optimized for case file automation that achieves 97% cost reduction on standardized documents, 85-95% straight-through processing rates, and sub-second to 60-second processing times depending on complexity. The system leverages AWS Textract, Bedrock with dynamic model selection, and template caching to process PDF case files with intelligent routing based on document characteristics.[1] [2] [3] [4] [5]

Architecture Overview

Core Components

Storage Layer: Amazon S3 with Intelligent-Tiering for document storage, S3 Event Notifications for upload triggers, and lifecycle policies for cost optimization.[6] [7] [1]

Processing Orchestration: AWS Step Functions coordinating multi-step workflows with automatic retry logic, error handling, and state management.[8] [1]

Compute Layer: AWS Lambda functions with provisioned concurrency for critical paths (Lambda 0 and Tier 0) and auto-scaling for variable workload tiers.[9] [10] [11]

AI/ML Services: Amazon Textract for OCR and structured extraction, Amazon Bedrock (Claude 3.5 Haiku, Sonnet, and Opus) for intelligent field mapping.[12] [13] [14] [1]

Data Stores: DynamoDB for document metadata and template cache with TTL, DynamoDB Streams for event-driven processing triggers.[15] [16] [1]

Queue Management: Amazon SQS for decoupled processing with separate queues per tier, enabling independent scaling and buffering during traffic spikes.[17] [9]

Monitoring: Amazon CloudWatch for logs, metrics, and dashboards; Amazon SNS for notifications and alerts.[18] [1]

Human Review: Amazon Augmented AI (A2I) for confidence-based human validation workflows. [2] [4] [1]

Processing Tier Architecture

Lambda 0: Initial Document Classification (200-300ms)

Trigger: S3 PutObject event via EventBridge notification.[7] [19]

Function Configuration: 256MB memory, 10-second timeout, provisioned concurrency of 2-3 instances to eliminate cold starts.[10] [9]

Processing Logic:

1. Read S3 object metadata without downloading full document (content-type, content-length, custom tags)[20] [7]
2. Extract first 1-5KB of PDF to determine page count using PyPDF2's header-only read[21] [22]
3. Apply filename pattern matching using regex: `*_form.pdf`, `*application*.pdf`, `new_client_*.pdf` → Tier 2; `case_notes_*.pdf`, `*_transcript.pdf` → Tier 1[5] [23]
4. Generate structural hash of first page using PyMuPDF coordinate extraction: `page.get_text("dict")` returns text block positions creating layout fingerprint[24] [25]
5. Query DynamoDB Template Cache table using hash as partition key[15]
6. Write classification decision to DynamoDB metadata table with routing destination[19] [26]
7. Send message to appropriate SQS queue (Tier 0, 1, 2, or page-filtering workflow)[17] [9]

Routing Decision Matrix:

- **Template cache hit + confidence >95%** → Tier 0 (template extraction)
- **Single page, <100KB, simple filename** → Tier 1 (DetectDocumentText)
- **2-10 pages, form pattern in filename** → Tier 2 (AnalyzeDocument)
- **50+ pages** → Page-filtering workflow then Tier 1/2
- **Metadata tag** `document-type: financial_statement` → Tier 2 with TABLES feature
- **Metadata tag** `priority: urgent` → Allocate provisioned concurrency

Cost per Classification: $0.0000002 (200ms execution, 256MB memory).[10]

Tier 0: Template-Based Fast Path (500-800ms)

Trigger: SQS message from Lambda 0 indicating template cache hit.[9]

Function Configuration: 512MB memory, 3-second timeout, provisioned concurrency of 1-2 instances.[10]

Processing Logic:

1. Retrieve full document from S3
2. Fetch template definition from DynamoDB cache using document hash[15]
3. Load PDF using PyMuPDF for coordinate-based extraction[25] [24]

4. For each field in template mapping, execute `page.get_textbox(rect)` to extract text within specified coordinates[27] [24]
5. Apply validation rules: field format checks (date patterns, name alphabetic validation)[28] [2]
6. Calculate extraction confidence score based on field completion and pattern matching[2]
7. If confidence >90%, write to DynamoDB and mark complete[4] [2]
8. If confidence <90%, escalate to Tier 1 with extracted data as hints[29] [28]
9. Update template usage counter and last_validated timestamp[15]

Template Record Structure:

```
{
  "template_hash": "sha256_hash_value",
  "version": 1,
  "field_mappings": [
    {
      "field_name": "client_name",
      "bbox": [72, 150, 300, 170],
      "page": 1,
      "validation": "^[A-Za-z\\s]+$"
    },
    {
      "field_name": "case_number",
      "bbox": [72, 200, 200, 220],
      "page": 1,
      "validation": "^[A-Z]{2}\\d{6}$"
    }
  ],
  "confidence_score": 97.5,
  "usage_count": 1247,
  "last_validated": 1727856000,
  "ttl": 1735718400
}
```

Cost per Document: $0.00008 (Lambda + DynamoDB reads/writes), **97% savings vs Tier 1**.[12] [10] [15]

Template Learning: Documents successfully processed through Tier 2/3 with >95% confidence automatically generate candidate templates after 3-5 identical structural hashes confirm reusability.[28] [2]

Tier 1: Simple Text Extraction (3-5 seconds)

Trigger: SQS message from Lambda 0 for straightforward documents or Tier 0 escalations.[9]

Function Configuration: 1024MB memory, 30-second timeout, auto-scaling based on SQS queue depth.[9] [10]

Processing Logic:

1. Retrieve document from S3

2. For multi-page documents from page-filtering workflow, process only flagged pages[6]
3. Invoke Textract DetectDocumentText API asynchronously[11] [12]
4. Store job ID and poll SNS topic for completion notification[1] [11]
5. Retrieve extracted text with confidence scores[30]
6. Calculate aggregate confidence: if >90% proceed to database storage, if 70-89% trigger validation review, if <70% escalate to Tier 2[4] [28]
7. Write results to DynamoDB with confidence metadata[1]

Textract API: `StartDocumentTextDetection` for async processing.[11]

Cost per Page: $0.0015.[31] [12]

Optimization: Split multi-page PDFs and process pages in parallel using Step Functions Map state, reducing processing time by 60-75% for 10+ page documents.[8] [6]

Tier 2: Structured Document Analysis (10-25 seconds)

Trigger: SQS message from Lambda 0 for documents with forms/tables, or Tier 1 escalations.[9]

Function Configuration: 2048MB memory, 60-second timeout.[10]

Processing Logic:

1. Retrieve document from S3
2. Determine required FeatureTypes based on Lambda 0 classification: FORMS for application documents, TABLES for financial statements, QUERIES for specific data extraction[30] [12]
3. Invoke Textract AnalyzeDocument API asynchronously with specific features[12] [30]
4. Await SNS completion notification[11] [1]
5. Retrieve structured extraction results including form key-value pairs, table cells, and query responses[30]
6. Calculate field-level confidence scores[30]
7. Fields with >90% confidence proceed directly; 70-89% queue for validation; <70% escalate to Tier 3 Bedrock processing[4] [28]

Textract API: `StartDocumentAnalysis` with FeatureTypes parameter.[12] [30]

Cost per Page:

- FORMS only: $0.050
- TABLES only: $0.015
- QUERIES: $0.001 per query
- Combined features: Additive pricing[31] [12]

Template Generation: Successful Tier 2 extractions with >95% confidence capture field coordinates and create candidate templates for future Tier 0 routing.[32] [6]

Tier 3: AI-Powered Intelligent Mapping (25-60 seconds)

Trigger: SQS message for complex documents or Tier 2 escalations with low confidence.[9]

Function Configuration: 3008MB memory, 90-second timeout.[10]

Dynamic Model Selection Logic:

Claude 3.5 Haiku (70-85% Textract confidence, 5-15 fields, 1-3 pages):

- Use case: Basic forms, invoices, simple case intake documents
- Cost: $0.02-$0.05 per document
- Processing time: 1-3 seconds
- Best for: Straightforward field mapping with clear contextual relationships[13] [14]

Claude 3.5 Sonnet (60-70% Textract confidence, 15-30 fields, 4-10 pages):

- Use case: Complex multi-section documents, contracts, medical records
- Cost: $0.08-$0.15 per document
- Processing time: 3-8 seconds
- Best for: Documents requiring contextual understanding across pages[14] [13]

Claude 3 Opus (<60% Textract confidence, 30+ fields, 10+ pages, or previous model failures):

- Use case: Heavily degraded scans, handwritten sections, unstructured narratives
- Cost: $0.15-$0.30 per document
- Processing time: 8-15 seconds
- Best for: Maximum accuracy on challenging documents with nuanced relationships[13] [14]

Processing Logic:

1. Retrieve Textract extraction results from previous tier
2. Analyze document complexity metrics: page count, field count, confidence distribution[14]
3. Select appropriate Bedrock model using decision matrix[14]
4. Construct prompt with Textract raw text and database field schema[1] [4]
5. Invoke Bedrock model via InvokeModel API[33] [4]
6. Parse JSON response mapping extracted data to database fields[1]
7. Calculate mapping confidence score[28] [4]
8. If Haiku/Sonnet confidence <75%, automatically retry with next tier model[28]
9. Final confidence >90% → database; 70-89% → validation queue; <70% → human review via A2I[29] [4] [28]

Automatic Fallback Chain: Haiku (fails) → Sonnet (fails) → Opus → Human Review.[29] [28]

Cost Optimization: Intelligent routing reduces Tier 3 costs by 40-55% versus Opus-only approach while maintaining accuracy within 2-3%.[14]

Implementation Phases

Phase 1: Foundation Infrastructure (Weeks 1-2)

Week 1 Objectives:

S3 Bucket Configuration:

- Create primary document ingestion bucket with versioning enabled[1]
- Configure S3 Event Notifications to EventBridge for PutObject events[7] [19]
- Enable S3 Intelligent-Tiering for automatic storage optimization[6]
- Create lifecycle policies: delete after 90 days or archive to Glacier[6]
- Configure server-side encryption with KMS[1]

DynamoDB Tables:

- **DocumentMetadata**: Partition key `document_id`, attributes include upload_timestamp, classification_result, processing_tier, status, confidence_score[19] [1]
- **TemplateCache**: Partition key `template_hash`, sort key `version`, GSI on `document_type`, TTL attribute set to 90 days[16] [15]
- Enable DynamoDB Streams on DocumentMetadata for event-driven processing[1]
- Configure point-in-time recovery for both tables[1]

IAM Roles and Policies:

- Lambda execution roles with least-privilege access to S3, DynamoDB, Textract[1]
- Service roles for Step Functions to invoke Lambda and Textract[1]
- KMS key policies for encryption/decryption[1]

Week 2 Objectives:

Lambda 0 Development:

- Implement classification function with PyPDF2 and PyMuPDF dependencies packaged as Lambda Layer[21] [24]
- Configure EventBridge rule triggering Lambda 0 on S3 uploads[7] [19]
- Implement filename pattern matching regex library[23] [5]
- Build structural hash generation using PyMuPDF coordinate extraction[24] [25]
- Integrate DynamoDB cache lookup logic[15]
- Create SQS queues for each processing tier[17] [9]
- Deploy with provisioned concurrency (2 instances)[10] [9]

Tier 1 Lambda Development:

- Implement DetectDocumentText invocation logic[11]
- Configure SNS topic subscription for Textract completion notifications[11] [1]

- Build confidence scoring and routing logic[30]
- Implement DynamoDB write operations for results storage[1]
- Create CloudWatch log groups and metric filters[1]

Basic Monitoring:

- CloudWatch dashboard with Lambda invocation counts, durations, errors[1]
- Alarms for Lambda throttling and error rates[1]
- SQS queue depth monitoring[9]

Testing:

- Unit tests for Lambda 0 classification logic
- Integration tests with sample PDFs of varying types
- Load testing with 100 concurrent uploads
- Validate end-to-end flow from S3 upload to Tier 1 completion

Deliverables: Functional Lambda 0 classification and Tier 1 simple extraction with 95% uptime.

Phase 2: Orchestration and Structured Processing (Weeks 3-4)

Week 3 Objectives:

Step Functions Workflow:

- Design state machine orchestrating Lambda 0 → Tier routing → Processing → Validation[8] [1]
- Implement error handling with exponential backoff retry (3 attempts with 2x multiplier)[9] [1]
- Configure catch blocks for Lambda failures routing to dead letter queue[9]
- Add Map state for parallel page processing on multi-page documents[8] [6]
- Deploy workflow with CloudWatch integration for execution history[1]

SQS Configuration:

- Create separate queues: Tier0Queue, Tier1Queue, Tier2Queue, Tier3Queue, ValidationQueue, DLQ[17] [9]
- Set visibility timeout to 900 seconds matching Lambda max execution time[9]
- Configure dead letter queue redrive policy after 3 receive attempts[9]
- Enable SQS message retention for 14 days[9]

Tier 2 Lambda Development:

- Implement AnalyzeDocument API with dynamic FeatureTypes selection[12] [30]
- Build form key-value pair extraction logic[30]
- Implement table cell parsing for financial documents[30]
- Add query-based extraction for specific data points[30]

- Configure SNS subscription for async job completion[11] [1]

Week 4 Objectives:

Page Filtering Workflow:

- Implement Lambda function analyzing page count from Lambda 0 metadata[6]
- Build logic identifying relevant pages based on content patterns[6]
- Create Step Functions sub-workflow splitting PDFs and processing pages in parallel[8] [6]
- Aggregate results from parallel executions[8]

Enhanced Monitoring:

- Add custom CloudWatch metrics: documents_per_tier, average_confidence_score, processing_duration_by_tier[1]
- Create dashboard visualizing tier distribution and throughput[1]
- Configure SNS topics for operational alerts[1]

Cost Tracking:

- Enable AWS Cost Allocation Tags on all resources[6]
- Tag resources by tier for granular cost analysis[6]
- Set up Cost Explorer reports for Textract usage[6]

Testing:

- Test Step Functions workflow with various failure scenarios
- Validate page filtering reduces Textract calls by 40%+[6]
- Load test with 1,000 concurrent documents across all tiers
- Verify SQS buffering handles traffic spikes without data loss

Deliverables: Complete orchestration with Tier 1 and Tier 2 processing achieving 10-25 second structured extraction times.

Phase 3: AI Integration and Human Review (Weeks 5-6)

Week 5 Objectives:

Bedrock Integration:

- Configure Amazon Bedrock access and model permissions (Claude 3.5 Haiku, Sonnet, Claude 3 Opus)[13] [14]
- Request service quota increases if needed (default: 100 requests/minute)[34]
- Implement Tier 3 Lambda with dynamic model selection logic[14]
- Build prompt templates with Textract output and database schema[4] [1]
- Develop JSON parsing for Bedrock responses with error handling[1]
- Implement automatic fallback: Haiku → Sonnet → Opus[28]

Model Selection Algorithm:

```
def select_bedrock_model(textract_confidence, page_count, field_count):
    if textract_confidence >= 70 and page_count <= 3 and field_count <= 15:
        return "claude-3-5-haiku"
    elif textract_confidence >= 60 and page_count <= 10:
        return "claude-3-5-sonnet"
    else:
        return "claude-3-opus"
```

Cost Optimization Layer:

- Implement batch processing for multiple documents when possible[34]
- Cache Bedrock responses for identical Textract outputs (hash-based deduplication)[18]
- Monitor token usage and optimize prompt length[14]

Week 6 Objectives:

Amazon A2I Human Review Workflows:

- Create human task UI templates displaying document and extracted fields side-by-side[2] [29] [4]
- Configure work teams with reviewers[4]
- Set confidence thresholds: 90%+ auto-approve, 70-89% validation, <70% full review[4] [28]
- Implement callback Lambda processing A2I review results[29] [4]
- Build review dashboard showing pending items and completion rates[2]

Template Cache Population:

- Identify successful Tier 2/3 extractions with >95% confidence[2] [28]
- Generate structural hashes and field coordinate mappings[25] [24]
- Store candidate templates after 3-5 identical hash confirmations[2] [28]
- Implement template validation tracking accuracy over time[2]

Feedback Loops:

- Capture human review corrections and update confidence scoring models[28] [2]
- Track documents requiring multiple processing attempts[28]
- Refine Lambda 0 classification rules based on routing accuracy[2]

Testing:

- Validate Bedrock model selection routes 60% to Haiku, 30% to Sonnet, 10% to Opus[14]
- Test automatic fallback chain with intentionally degraded documents
- Verify A2I workflows trigger at correct confidence thresholds[4]
- Confirm template candidate generation from high-confidence extractions

Deliverables: Full AI-powered processing with human-in-the-loop validation achieving 85% straight-through processing rate.

Phase 4: Template Fast Path and Optimization (Weeks 7-8)

Week 7 Objectives:

Tier 0 Implementation:

- Develop ultra-lightweight Lambda (512MB) for template-based extraction[24]
- Implement PyMuPDF coordinate-based text extraction using `page.get_textbox(rect)`[27] [24]
- Build field validation rules (regex patterns, format checks)[2]
- Configure provisioned concurrency (1-2 instances)[10]
- Create confidence scoring for template match quality[2]
- Implement automatic escalation to Tier 1 on validation failures[28]

Enhanced Lambda 0:

- Integrate structural hash generation into classification workflow[25] [24]
- Add DynamoDB cache lookup before tier routing[15]
- Route cache hits with >95% confidence directly to Tier 0[2]
- Track cache hit rates in CloudWatch metrics[1]

Template Management:

- Build admin interface for reviewing and approving candidate templates
- Implement template versioning for incremental improvements[15]
- Configure TTL to auto-expire unused templates after 90 days[16]
- Create GSI for querying templates by document_type[15]

Week 8 Objectives:

Performance Optimization:

- Analyze CloudWatch metrics identifying bottlenecks[1]
- Optimize Lambda memory allocation based on actual usage (2-4 week observation)[35] [36]
- Implement Lambda Layer for shared dependencies reducing deployment package size[10]
- Configure S3 Transfer Acceleration for faster large file uploads[1]
- Enable DynamoDB Auto Scaling based on read/write capacity patterns[15]

Cost Optimization:

- Review Textract usage and eliminate unnecessary feature combinations[6]
- Implement smart caching for repeated document processing[18]
- Optimize Step Functions state transitions to reduce state change costs[1]

- Configure Savings Plans for consistent Lambda usage[36] [35]

Comprehensive Monitoring:

- Deploy full CloudWatch dashboard with all tier metrics[1]
- Add business metrics: documents_processed_per_hour, cost_per_document, straight_through_rate[6]
- Configure detailed alarms: template_cache_hit_rate <40%, confidence_score_degradation, processing_duration_p95 >60s[1]
- Set up weekly cost reports via SNS[1]

Load Testing:

- Stress test with 10,000 documents over 1 hour (2.7 docs/sec sustained)
- Validate auto-scaling handles 100 docs/sec burst traffic
- Confirm SQS queues buffer spikes without message loss[9]
- Verify Tier 0 processes 70% of standardized forms

Testing:

- Template cache hit rate >40% on standardized documents
- Tier 0 processing completes in <1 second 95th percentile
- End-to-end cost reduction of 35-45% vs baseline (no template caching)[6]
- System handles 5x normal traffic without degradation

Deliverables: Production-ready system with template fast path achieving 92-95% straight-through processing and <$0.01 average cost per document.

Phase 5: Production Hardening and Continuous Improvement (Weeks 9-10)

Week 9 Objectives:

Disaster Recovery:

- Configure S3 cross-region replication for document buckets[1]
- Deploy DynamoDB Global Tables for multi-region template cache[15] [1]
- Create failover Step Functions workflows in secondary region[1]
- Implement health checks and automatic region failover[1]
- Document RTO (Recovery Time Objective: 15 minutes) and RPO (Recovery Point Objective: 0 data loss)[1]

Security Hardening:

- Enable AWS CloudTrail for audit logging of all API calls[1]
- Configure VPC endpoints for Textract and Bedrock to avoid internet routing[1]
- Implement S3 bucket policies blocking public access[1]

- Enable GuardDuty for threat detection[1]
- Conduct security review of IAM policies following least-privilege principle[1]

Compliance and Data Privacy:

- Implement data retention policies aligned with regulatory requirements[6]
- Configure PII detection using Amazon Comprehend for sensitive documents[1]
- Enable field-level encryption for highly sensitive data[1]
- Document data flow for compliance audits[1]

Week 10 Objectives:

Machine Learning Improvements:

- Analyze 30 days of processing data to refine Lambda 0 classification rules[2]
- Retrain confidence thresholds based on human review patterns[28]
- Identify document types requiring custom Bedrock prompts[14]
- Optimize template matching tolerance (bounding box variance)[24]

Operational Runbooks:

- Document troubleshooting procedures for common failure scenarios
- Create escalation procedures for critical alerts
- Build knowledge base of document processing edge cases
- Train operations team on monitoring dashboards and manual interventions

Performance Benchmarking:

- Establish baseline metrics: average processing time per tier, cost per document type, accuracy rates[37]
- Compare against initial Phase 1 metrics showing improvements
- Document ROI: processing cost savings, time reduction, error rate improvements[6]

Continuous Improvement Framework:

- Monthly review of template cache effectiveness and addition of new templates
- Quarterly analysis of Bedrock model performance and cost-optimization opportunities[14]
- Bi-weekly review of human review feedback to improve automation[28] [2]
- Regular assessment of AWS service updates and feature enhancements[1]

Production Cutover:

- Conduct final end-to-end testing with production-like data volume
- Perform security and compliance sign-off
- Execute gradual rollout: 10% traffic week 1, 50% week 2, 100% week 3
- Monitor closely for unexpected behaviors

- Maintain rollback capability to previous system for 30 days

Deliverables: Production-hardened system with documented processes, disaster recovery, and continuous improvement framework supporting long-term operations.

Technical Specifications

Lambda Function Specifications

Function	Memory	Timeout	Concurrency	Avg Duration	Cold Start
Lambda 0	256MB	10s	Provisioned: 2-3	200-300ms	0ms (provisioned)
Tier 0	512MB	3s	Provisioned: 1-2	500-800ms	0ms (provisioned)
Tier 1	1024MB	30s	Auto-scaling	3-5s	500-800ms
Tier 2	2048MB	60s	Auto-scaling	10-25s	800-1200ms
Tier 3	3008MB	90s	Auto-scaling	25-60s	1000-1500ms

SQS Queue Configuration

Queue	Visibility Timeout	Message Retention	Max Receives	DLQ
Tier0Queue	10s	14 days	3	Yes
Tier1Queue	60s	14 days	3	Yes
Tier2Queue	120s	14 days	3	Yes
Tier3Queue	180s	14 days	3	Yes
ValidationQueue	7 days	14 days	N/A	No

DynamoDB Table Design

DocumentMetadata Table:

- Partition Key: `document_id` (String)
- Attributes: `upload_timestamp`, `s3_key`, `classification_result`, `processing_tier`, `status`, `confidence_score`, `extracted_data`, `human_review_required`
- DynamoDB Streams: Enabled (New and Old Images)
- Capacity: On-demand billing mode for variable workloads

TemplateCache Table:

- Partition Key: `template_hash` (String)
- Sort Key: `version` (Number)
- Attributes: `field_mappings` (Map), `confidence_score`, `usage_count`, `last_validated`, `document_type`, `ttl`
- GSI: `document_type-index` (document_type as partition key)

- TTL: Enabled on `ttl` attribute (90 days from last use)
- Capacity: On-demand billing mode

AWS Service Limits and Quotas

Textract:

- Concurrent Jobs (DetectDocumentText): 100 (default, can increase to 1000)
- Concurrent Jobs (AnalyzeDocument): 100 (default, can increase to 1000)
- Max Document Size: 500MB
- Max Pages: 3000 per document

Bedrock:

- Requests per minute (Claude 3.5 Haiku): 100 (default)
- Requests per minute (Claude 3.5 Sonnet): 100 (default)
- Requests per minute (Claude 3 Opus): 100 (default)
- Max tokens per request: 200,000 (Claude 3.5)

Lambda:

- Concurrent Executions: 1000 (account-wide default, can increase)
- Function Timeout: 900s maximum
- Deployment Package Size: 250MB unzipped

S3:

- No limit on storage
- 5,500 GET/HEAD requests per second per prefix
- 3,500 PUT/COPY/POST/DELETE requests per second per prefix

Cost Projections

Per-Document Cost Breakdown (100,000 documents/month)

Tier 0 (45,000 docs - template cache hits):

- Lambda execution: $0.00008 × 45,000 = $3.60
- DynamoDB reads/writes: ~$5.00
- **Subtotal**: $8.60 ($0.0002/doc)

Tier 1 (30,000 docs - simple extraction, avg 3 pages):

- Textract DetectDocumentText: $0.0015 × 3 × 30,000 = $135
- Lambda execution: $0.002 × 30,000 = $60
- **Subtotal**: $195 ($0.0065/doc)

Tier 2 (20,000 docs - structured analysis, avg 2 pages, FORMS only):

- Textract AnalyzeDocument: $0.050 × 2 × 20,000 = $2,000
- Lambda execution: $0.005 × 20,000 = $100
- **Subtotal**: $2,100 ($0.105/doc)

Tier 3 (5,000 docs - AI mapping):

- 3,000 docs via Haiku: $0.04 × 3,000 = $120
- 1,500 docs via Sonnet: $0.10 × 1,500 = $150
- 500 docs via Opus: $0.25 × 500 = $125
- Lambda execution: $0.008 × 5,000 = $40
- **Subtotal**: $435 ($0.087/doc)

Infrastructure Costs:

- S3 storage (100GB active + 200GB Intelligent-Tiering): $15
- DynamoDB (on-demand): $50
- Step Functions (200,000 state transitions): $50
- CloudWatch logs and metrics: $30
- SNS notifications: $5
- SQS messages: $2
- **Subtotal**: $152

Monthly Total: $2,890.60 for 100,000 documents = **$0.029 per document average**

Yearly Projection (1.2M documents): $34,687

Cost Comparison vs No Template Caching

Without Tier 0, all 45,000 template-hit documents would process through Tier 1:

- Additional Textract cost: 45,000 × 3 pages × $0.0015 = $202.50/month
- Additional Lambda cost: 45,000 × $0.002 = $90/month
- **Monthly savings from template caching**: $292.50 (10% total cost reduction)
- **Annual savings**: $3,510

ROI Analysis

Traditional Manual Processing (baseline):

- Staff time: 5 minutes per document × 100,000 docs = 8,333 hours
- Cost at $25/hour: $208,325/month
- Error rate: 2-5% requiring rework

Automated IDP System:

- Infrastructure cost: $2,890.60/month
- Human review (8% of documents): 8,000 docs × 2 min × $25/hr = $6,667/month
- Total monthly cost: $9,557.60
- **Monthly savings**: $198,767.40
- **ROI**: 2080% (payback period: 0.5 months)
- Error rate: <1% on automated processing

Performance Expectations

Processing Time Targets

Document Type	Volume %	Tier	Target Time	P95 Time
Standardized forms (cache hit)	45%	0	600ms	800ms
Simple text documents	30%	1	4s	5s
Forms with tables	15%	2	15s	25s
Complex multi-page	8%	3 (Haiku/Sonnet)	35s	50s
Highly complex	2%	3 (Opus) + Review	120s	180s

Overall Average Processing Time: 8.5 seconds (weighted by volume distribution)

Throughput Capacity

Sustained Load: 2,000 documents/hour (0.55 docs/sec)
Burst Capacity: 20,000 documents/hour (5.5 docs/sec) for up to 2 hours via SQS buffering
Maximum Capacity (with quota increases): 100,000 documents/hour (27 docs/sec)

Accuracy Targets

Straight-Through Processing Rate: 92-95% (documents completing without human review)

Tier-Specific Accuracy:

- Tier 0: 97-99% (template-based, highest confidence)
- Tier 1: 85-90% (simple text extraction)
- Tier 2: 88-93% (structured forms)
- Tier 3: 95-98% (AI-enhanced mapping)

Error Rates:

- False positives (incorrect data auto-approved): <0.5%
- False negatives (correct data flagged for review): <3%

System Availability

Uptime Target: 99.9% (8.76 hours downtime/year)
RTO (Recovery Time Objective): 15 minutes
RPO (Recovery Point Objective): 0 (no data loss)

Monitoring and Operations

CloudWatch Dashboards

Executive Dashboard:

- Documents processed today/this week/this month
- Average cost per document (trending)
- Straight-through processing rate
- Human review queue depth
- System health score (composite metric)

Operational Dashboard:

- Lambda invocations per tier with success/error rates
- Processing duration percentiles (P50, P95, P99) per tier
- SQS queue depths with age of oldest message
- Textract API throttling occurrences
- Bedrock model usage distribution
- DynamoDB consumed capacity units

Cost Dashboard:

- Daily spend by service (Textract, Bedrock, Lambda, storage)
- Cost per tier breakdown
- Template cache hit rate impact on costs
- Projected monthly spend vs budget

Quality Dashboard:

- Confidence score distribution across tiers
- Documents requiring human review (trending)
- Template cache accuracy over time
- Human review approval vs rejection rates
- Error categories and frequencies

CloudWatch Alarms (Critical)

Alarm	Condition	Action
High Error Rate	Lambda errors >5% over 5 min	SNS to on-call, auto-retry
Queue Buildup	SQS depth >1000 messages for 15 min	Scale up Lambda concurrency, SNS alert
Processing Delay	P95 duration >120s for 10 min	Investigate bottlenecks, SNS alert
Textract Throttling	>10 throttles per minute	Request quota increase, route to slower queue
Template Cache Degradation	Hit rate <35% over 24 hours	Review template expirations, regenerate
Human Review Backlog	>500 pending reviews	Notify review team, consider confidence threshold adjustment
Cost Spike	Daily spend >150% of average	SNS to finance team, review usage patterns
System Availability	Health check fails for 5 min	Trigger failover to secondary region

Operational Runbooks

Scenario: Textract Throttling Detected

1. Check CloudWatch for throttling rate and affected API (DetectDocumentText vs AnalyzeDocument)
2. Verify current concurrent job count against quota
3. Short-term: Reduce Lambda concurrency to slow request rate
4. Medium-term: Request quota increase via AWS Support ticket
5. Long-term: Implement more aggressive SQS rate limiting

Scenario: Template Cache Hit Rate Dropping

1. Query DynamoDB for templates with approaching TTL expiration
2. Review recent documents not matching cached templates
3. Identify new document format variations requiring new templates
4. Manually process sample documents through Tier 2/3 to generate new templates
5. Validate new templates before cache insertion

Scenario: Human Review Queue Backlog

1. Check current queue depth and average review time
2. Assess if backlog is due to volume spike or confidence threshold issue
3. If volume spike: notify additional reviewers, consider temporary confidence threshold increase
4. If threshold issue: analyze rejected reviews for patterns, adjust tier routing

5. Implement queue prioritization for urgent document types

Scenario: Region Failover Required

1. Confirm primary region outage via AWS Health Dashboard
2. Update Route 53 to point to secondary region endpoint
3. Verify DynamoDB Global Table replication lag <1 minute
4. Start Step Functions in secondary region
5. Monitor processing resumption, communicate ETA to users
6. Once primary region recovers, plan controlled failback during low-traffic window

Success Metrics (90-Day Targets)

Efficiency Metrics:

- ✓ Straight-through processing rate: >92%
- ✓ Template cache hit rate: >60%
- ✓ Average processing time: <10 seconds
- ✓ P95 processing time: <45 seconds

Quality Metrics:

- ✓ Extraction accuracy: >95% on automated processing
- ✓ Human review approval rate: >90% (low false positive rate)
- ✓ Error rate: <1%
- ✓ Customer satisfaction score: >4.5/5

Cost Metrics:

- ✓ Average cost per document: <$0.035
- ✓ Cost reduction vs manual processing: >95%
- ✓ ROI achievement: Break even within 30 days

Operational Metrics:

- ✓ System availability: >99.9%
- ✓ Mean time to detection (MTTD): <5 minutes
- ✓ Mean time to resolution (MTTR): <15 minutes
- ✓ Incident-free days: >85 out of 90

Risk Management

Technical Risks

Risk: Textract accuracy degradation on poor-quality scans

- **Mitigation**: Implement image preprocessing Lambda with contrast enhancement and noise reduction
- **Fallback**: Route low-quality documents directly to Tier 3 with Opus for maximum capability

Risk: Template cache false positives (wrong template applied)

- **Mitigation**: Strict validation rules with automatic escalation on format mismatches
- **Monitoring**: Track Tier 0 validation failure rate, alert if >5%

Risk: Bedrock API rate limits during traffic spikes

- **Mitigation**: SQS buffering with backpressure, progressive backoff, request quota increases
- **Fallback**: Queue documents for delayed processing, notify users of extended wait times

Risk: DynamoDB hot partition on popular templates

- **Mitigation**: Use compound hash keys distributing requests, enable on-demand capacity mode
- **Monitoring**: Track consumed capacity and throttling events

Operational Risks

Risk: Human review team unavailable causing queue buildup

- **Mitigation**: Implement on-call rotation, cross-train team members, document review procedures
- **Escalation**: Temporarily raise confidence thresholds to reduce review volume

Risk: AWS service outage impacting processing

- **Mitigation**: Multi-region deployment, S3 cross-region replication, DynamoDB Global Tables
- **Testing**: Quarterly disaster recovery drills

Risk: Cost overruns from unexpected usage patterns

- **Mitigation**: AWS Budgets with alerts at 80%/100%/120% thresholds, automatic scaling limits
- **Review**: Weekly cost analysis calls during first 90 days

Compliance Risks

Risk: PII/PHI exposure in logs or error messages

- **Mitigation**: Implement log scrubbing removing sensitive data, encrypt CloudWatch logs
- **Audit**: Quarterly security reviews of logging practices

Risk: Data retention violating regulatory requirements

- **Mitigation**: Automated lifecycle policies enforcing deletion schedules, encryption at rest
- **Documentation**: Maintain data flow diagrams for compliance audits

Conclusion

This comprehensive execution plan delivers an **enterprise-grade intelligent document processing system** optimized for case file automation with five-tier processing architecture, template caching, and dynamic AI model selection. The phased 10-week implementation provides incremental value delivery while managing risk through thorough testing at each stage. [3] [5] [17] [1]

The system achieves **97% cost reduction** on standardized forms through Tier 0 template caching, **92-95% straight-through processing** rates eliminating manual data entry, and **sub-second to 60-second processing times** depending on document complexity. Serverless architecture ensures automatic scaling from zero to thousands of concurrent documents without infrastructure management. [3] [17] [4] [2] [9] [1]

Expected **ROI exceeds 2000%** with infrastructure costs under $3,000/month processing 100,000 documents compared to $208,000+ for manual processing, with payback achieved in the first month of operation. The architecture supports long-term success through continuous improvement frameworks, comprehensive monitoring, and disaster recovery capabilities. [28] [2] [6] [1]

⁂

1. https://aws.amazon.com/solutions/guidance/intelligent-document-processing-on-aws/
2. https://parseur.com/blog/hitl-best-practices
3. https://www.vellum.ai/blog/document-data-extraction-in-2025-llms-vs-ocrs
4. https://aws.amazon.com/blogs/machine-learning/process-multi-page-documents-with-human-review-using-amazon-bedrock-data-automation-and-amazon-sagemaker-ai/
5. https://aihub.hkuspace.hku.hk/2024/04/12/cost-effective-document-classification-using-the-amazon-titan-multimodal-embeddings-model/
6. https://aws.amazon.com/blogs/machine-learning/build-well-architected-idp-solutions-with-a-custom-lens-part-5-cost-optimization/
7. https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingMetadata.html
8. https://aws.amazon.com/blogs/machine-learning/implement-smart-document-search-index-with-amazon-textract-and-amazon-opensearch/
9. https://aws.amazon.com/blogs/compute/building-scalable-serverless-applications-with-amazon-s3-and-aws-lambda/

10. https://www.serverless.com/aws-lambda
11. https://docs.aws.amazon.com/textract/latest/dg/lambda.html
12. https://aws.amazon.com/textract/pricing/
13. https://caylent.com/blog/open-ai-vs-bedrock-optimizing-generative-ai-on-aws
14. https://aws.amazon.com/blogs/machine-learning/effective-cost-optimization-strategies-for-amazon-bedrock/
15. https://xebia.com/blog/using-dynamodb-as-a-cache/
16. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/TTL.html
17. https://github.com/aws-samples/amazon-textract-serverless-large-scale-document-processing
18. https://aws.amazon.com/blogs/compute/caching-data-and-configuration-settings-with-aws-lambda-extensions/
19. https://aws.amazon.com/blogs/storage/integrating-custom-metadata-with-amazon-s3-metadata/
20. https://www.cloudthat.com/resources/blog/organizing-and-searching-data-at-scale-with-amazon-s3-metadata
21. https://unstract.com/blog/evaluating-python-pdf-to-text-libraries/
22. https://substack.com/home/post/p-162342870
23. https://aws.amazon.com/blogs/machine-learning/moderate-classify-and-process-documents-using-amazon-rekognition-and-amazon-textract/
24. https://pymupdf.readthedocs.io/en/latest/app1.html
25. https://pymupdf.readthedocs.io/en/latest/recipes-text.html
26. https://github.com/aws-samples/amazon-s3-contentmetadata
27. https://stackoverflow.com/questions/22898145/how-to-extract-text-and-text-coordinates-from-a-pdf-file
28. https://instabase.com/blog/overcoming-the-limitations-of-llms-data-validation-and-human-review/
29. https://electroneek.com/use-case/human-validation-of-document-processing/
30. https://docs.aws.amazon.com/textract/latest/dg/textract-best-practices.html
31. https://cloudchipr.com/blog/aws-textract
32. https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/automatically-extract-content-from-pdf-files-using-amazon-textract.html
33. https://aws.amazon.com/blogs/machine-learning/scalable-intelligent-document-processing-using-amazon-bedrock-data-automation/
34. https://milvus.io/ai-quick-reference/does-amazon-bedrock-support-scaling-up-for-highthroughput-scenarios-and-what-steps-should-i-take-to-ensure-my-application-scales-effectively-with-bedrock
35. https://www.prosperops.com/blog/aws-cost-optimization/
36. https://spot.io/resources/aws-cost-optimization/8-tools-and-tips-to-reduce-your-cloud-costs/
37. https://www.easydataworld.com/ocr-speed/
38. https://github.com/aws-samples/amazon-serverless-document-processing
39. https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/sample-architecture-patterns.html
40. https://aws.amazon.com/blogs/machine-learning/automate-document-processing-with-amazon-bedrock-prompt-flows-preview/

41. https://serverlessland.com/repos/serverless-intelligent-document-processing