**Production-Ready AWS Tech Stack for Perspective-Based Learning (v7.0)**

This document outlines the v7.0 launch-ready architecture. This version includes a highly-detailed **RAG Workflow**, a robust **Resilience and Observability Framework**, a **Closed-Loop Feedback System**, a **realistic First-Year Cost Analysis**, and critical pre-launch specifications for **multi-document course synthesis**, **data governance**, and **pipeline versioning**.

**Layer 0: File Validation & Pre-Processing (AWS Lambda & Amazon Textract)**

* **PDF Validation & Hashing:** An **AWS Lambda** function triggers on file upload. It uses PyPDF2 to check for password protection and corruption and generates a SHA256 hash for caching.
* **OCR Fallback for Scanned PDFs:** If the PDF is image-based, this layer uses **Amazon Textract**, a managed AWS service, for superior OCR performance.

**Layer 1: Caching & Asynchronous Task Queuing (Amazon ElastiCache, SQS & Fargate)**

* **Caching & Versioning:** **Amazon ElastiCache for Redis** caches the final processed results. To ensure maintainability, cache keys are versioned (e.g., processed:{file\_hash}:v{PIPELINE\_VERSION}). When a new pipeline version is deployed, the system can either reprocess popular documents proactively or fall back to a compatible older version for non-breaking changes.
* **Task Queue:** **Amazon SQS (Simple Queue Service)** provides a durable queue for new document processing requests, with a configured Dead Letter Queue (DLQ) for failed tasks.
* **Processing Workers:** **AWS Fargate** containers poll the SQS queue and execute the main pipeline.

**Layer 2: PDF Parsing, Chunking & Normalization**

* **Layer 2.0: PDF Parsing (LlamaParse):** The **LlamaParse** API converts the PDF into structured Markdown. The implementation includes a robust fallback chain (see Operational Resilience).
* **Layer 2.1: Hierarchy Normalization:** A dedicated post-processing step normalizes the extracted structure, mapping all sections to a consistent ID scheme (e.g., chapter\_1\_section\_2).
* **Layer 2.5: Past Exam Parsing:** The same pipeline processes optional past exam PDFs, normalizing their structure.

**Layer 3: Hierarchical Document Processing (Amazon SageMaker)**

* **Model Hosting:** The howey/HDT-E model is deployed on **Amazon SageMaker**.
* **Hosting Strategy:** **SageMaker Serverless Inference** is used. To handle large documents and mitigate cold starts, embeddings are generated in **batched requests**. If a document's keyword count exceeds the payload limit, the list is chunked into multiple sequential batch requests.

**Layer 4: Multi-Method Keyword Extraction & Cross-Analysis**

* **Ensemble Method:** A combination of **KeyBERT**, **YAKE**, and **spaCy's TextRank** extracts keywords.
* **Layer 4.5: Exam Relevance Score Calculation:** The exam\_relevance\_score is calculated based on the **semantic similarity (cosine similarity)** between its embedding and the embeddings of keywords extracted from a past exam.
* **Layer 4.6: Applying Exam Relevance:** Decision rules are applied to the score, assigning a priority ("high", "medium", or "low") to keywords, which the frontend uses for visual styling.

**Layer 5: RAG-Powered Concept Mapping & Chapter View Generation**

* **Duplicate Keyword Handling:** The system adopts a **"Primary Node + Reference Nodes"** strategy. The API response includes a specific visual\_style for reference nodes (e.g., { "opacity": 0.7, "border\_style": "dotted" }) for frontend differentiation.
* **RAG Workflow:** The system runs a chapter-scoped semantic search against **Amazon RDS (pgvector)**. If results are sparse, the search expands to neighboring chapters to find meaningful connections. The retrieved context is then passed to **Claude 3.5 Sonnet via Amazon Bedrock**.
* **Layer 5.5: Chapter-Focused Map Generation:** This step pre-computes individual chapter views.

**Layers 6 & 7: Data Storage & Auth (S3, RDS, Cognito)**

* **Layer 6: Vector & Object Storage:** **Amazon S3** for original PDFs. **Amazon RDS for PostgreSQL** with **pgvector** for embeddings.
* **Layer 7: Primary Database & Schema:** **Amazon RDS** schema is expanded:
  + processed\_documents: Stores maps, pipeline\_version, and a subject field (e.g., "Biology") for homograph detection.
  + courses & course\_documents: Tables to manage course-to-document relationships.
  + user\_annotations: Stores user feedback.
* **Authentication:** **Amazon Cognito** handles user authentication.

**Layer 8: API Layer & Orchestration (AWS Fargate & API Gateway)**

* **API Hosting & Rate Limiting:** A FastAPI application on **AWS Fargate**, with rate limits (/upload: 5/hr, /feedback: 50/day, etc.) enforced by **API Gateway Usage Plans**.
* **API Endpoints:**
  + /upload-document, /status/{task\_id}, /feedback
  + **/courses & /courses/{course\_id}/documents:** Full CRUD endpoints for managing courses and their associated documents.
  + **/concept-map/course/{course\_id}:** Endpoint for multi-document synthesis.

**Layer 9: Continuous Improvement & Feedback Loop (AWS Lambda & EventBridge)**

* **Feedback Processing:** A weekly scheduled **AWS Lambda** function processes feedback based on a **consensus model** (3+ unique users flagging the same keyword).
* **Feedback Integrity & Reputation:** The /feedback endpoint implements rate limiting. A user's reputation score is calculated based on their feedback history's alignment with consensus.

**Layer 10: Cross-Document Synthesis**

* **Multi-Textbook Merging:** A core feature available via /concept-map/course/{course\_id}. The endpoint has a default limit of 5 documents to prevent timeouts and provides a graceful fallback to returning individual maps if the merge process fails.
* **Entity Resolution Algorithm:** Merges nodes based on a **cosine similarity > 0.95**. To resolve homographs (e.g., "cell" in biology vs. "cell" in law), it performs an additional check on the document subject field.
* **Conflict & Edge Resolution:** For conflicting relationships between nodes, the API provides the user with the different options, an AI recommendation from Claude, and a UI path to resolution. Synonymous edges (e.g., "uses," "requires") are merged into a single edge with a tooltip on the frontend.

**CI/CD, Governance & Compliance**

* **Layer 11: Feature Flags (AWS AppConfig):** An A/B testing framework using **AWS AppConfig** is implemented to test pipeline improvements (e.g., new models, different prompts) on a subset of users before full rollout.
* **Data Governance & GDPR:** A strict **Data Retention Policy** is defined and implemented via S3 Lifecycle policies and scheduled database jobs. A GDPR-compliant /users/{user\_id}/data endpoint provides a full data deletion cascade, anonymizing feedback annotations while removing all PII.

**Monitoring & Observability (AWS X-Ray & CloudWatch)**

* **Distributed Tracing & Metrics:** The pipeline is instrumented with **AWS X-Ray**. A **CloudWatch Dashboard** monitors pipeline latency, cache hit/miss rates, feedback quality, and fallback costs.
* **Alerting Strategy:** CloudWatch alarms are configured for critical (P1), error (P2), and warning (P3) thresholds on key metrics like SQS queue depth and Fargate error rates.

**Operational Excellence & Resilience**

* **Fallback Chain:** If LlamaParse fails after exponential backoff, the system automatically falls back to **Amazon Textract** + **Bedrock/Claude** to ensure the user receives a result.
* **Disaster Recovery:** Critical services have defined recovery paths (e.g., SQS DLQs, cache-to-RDS fallback). The processed\_documents table stores the pipeline\_version used for each map, enabling traceability and targeted re-processing.

**Complete AWS Architecture Diagram (v6.0)**

User Upload (PDF/Exam) → [API Gateway w/ Usage Plans] → [Fargate API]

↓

[S3 Bucket] → triggers → [Layer 0: Lambda Validator]

↓ (Validation OK)

[ElastiCache for Redis] ← Cache Check (Versioned Key) → Return Result

↓ (Cache Miss)

[SQS Queue (with DLQ)] → triggers → [Fargate Worker Pool]

--- Fargate Worker Task (Instrumented with AWS X-Ray) ---

1. [Layer 2: Parse & Normalize (with Textract/Claude fallback)]

2. [Layer 3: SageMaker Serverless (Batch Embeddings)]

3. [Layer 6: RDS w/ pgvector] → STORE chunks & embeddings

4. [Layer 4: Keyword Extraction & Exam Scoring]

5. [Layer 5: RAG Workflow (Homograph-aware Search)] → Full Map

6. [Layer 5.5: Generate Chapter Views]

7. [Layer 7: RDS] → STORE Maps, Metadata, Version

8. [Layer 1: ElastiCache] → CACHE final result

--- End of Task ---

[EventBridge (Weekly)] → triggers → [Layer 9: Lambda Feedback Processor]

[CloudWatch Dashboard] ← Monitors all services via X-Ray & Logs