Problem Statement

Coursera's Enterprise customers face significant friction in finding relevant learning content across a catalog of 16,000+ courses. Key pain points include incomplete and inconsistent metadata, lack of item-level visibility (e.g., specific videos or readings), and limited semantic understanding of course and item content.

The AI-Led Curations system addresses these gaps through an end-to-end discovery pipeline that enables:

- Conversational requirement gathering via an intelligent, finite-state chat workflow
- Semantic + metadata-driven course filtering
- Intent-based dynamic course ranking
- Deep item-level retrieval using pre-embedded transcripts and vector search
- Structured recommendations and curated learning pathways
- A continuous refinement loop based on user feedback
- Full-scale infrastructure for embeddings, vector indexing (e.g., FAISS or equivalent), data quality, and orchestration

Solution Design

The solution is organized into seven core objectives (A–G) that together form a scalable, auditable pipeline from user intent to delivered learning experiences.

Objective	Primary Purpose	Key Inputs	Key Outputs	Core Techniques / Components
Structured Extraction System (Requirement Builder)	Convert natural-language queries into a structured, confidence-aware requirement object	User query; chat history; Coursera ontology; regex rules	Workflow type; topics/skills; proficiency; duration; language; soft constraints; confidence scores	FSM-driven dialog; LLM extraction; ontology-based validation; regex parsing; integrated confidence scoring

Course Universe Filtering Engine	Build a high-recall "candidate course universe" from the catalog based on skills and constraints	Requirement object from A; course metadata; embeddings; ontology	Candidate course IDs; filter summary counts	Semantic skill expansion; fuzzy matching; deterministic metadata filters (difficulty, duration, language, rating, partner, freshness); relaxation logic
Dynamic Course Ranking Engine	Rank candidate courses by relevance and user intent with dynamic weights	Candidate courses from B; requirement object from A; course metadata; skill-match scores	Ranked courses with final_score; feature_scores; weights	Feature engineering (rating, popularity, freshness, duration fit, skill match, flags); dynamic weight templates; weighted scoring; edge-case rules
Item-Level Retrieval (Vector Search + Confidence)	Retrieve and score item-level content (videos/readings/transcripts) across ranked courses	Ranked courses from C; requirement object from A; item embeddings/vector index	Retrieved items with S_item; batch confidence S_conf	Pre-embedding pipeline; vector search; chunk-level retrieval; S_item & S_conf computation; cascading batches (top 5 → 20 → long tail)
Output Assembly & Delivery (Results Layer)	Assemble ranked recommendations or curated pathways into UI-ready responses	Ranked courses from C; retrieved items from D; requirement & constraints from A	JSON structures for recommendations and curations; metadata-enriched lists	Recommendation builder; curation pathway builder; metadata enrichment (logos, thumbnails, timestamps); mapping scores to display order
Feedback Loop & Search Refinement	Interpret user feedback and refine constraints / intent, then re-run the pipeline	Original requirement from A; user feedback; latest results from B–D	Updated requirement object; refreshed results via B → C → D	LLM feedback classifier; constraint vs intent detection; state update; strict feedback constraints; pipeline re-entry rules
Data, Embeddings & Infrastructure (Foundational)	Provide embeddings, vector storage, metadata refresh, and monitoring for the entire system	Raw content; metadata from Databricks; course lifecycle events	Embeddings; vector indices; refreshed metadata tables; monitoring signals	Batch + event-driven embedding pipeline; vector DB with metadata filters; metadata management; latency/cost/health monitoring

Objective A - Structured Extraction System (Requirement Builder)

Purpose

Convert natural-language queries into a structured, confidence-aware requirement object that captures workflow type, skills, proficiency, duration, language, and soft preferences.

How We Solve It

Component	Solution Approach
Requirement extraction	Finite State Machine controls intent classification, follow-ups (max two), and extraction completeness.
Core extraction engine	LLM extracts workflow type, skills, domain, proficiency, duration, and soft constraints with confidence scores.
Ontology alignment	Validate and adjust outputs using embedding similarity with Coursera's canonical taxonomy.
Deterministic parsing	Regex/rules refine explicit signals such as duration, level, and language.
Confidence framework	Weighted combination of LLM confidence, extraction consistency, ontology similarity, deterministic validation, and FSM signal.
Output	Final structured requirement object powering filtering (B), ranking (C), and retrieval (D).

Objective B - Course Universe Filtering Engine

Purpose
Construct a wide-coverage "candidate course universe" based on semantic skill signals and metadata constraints, ensuring high recall.

How We Solve It

Component	Solution Approach
Semantic expansion	Embeddings of LOs, descriptions, and allskills; fuzzy match for subskills; domain boosting for context.
Deterministic filters	Apply constraints for difficulty, duration, language, ratings, partner, freshness.
Relaxation logic	If the candidate set too small: drop noncritical constraints (duration, format), then fall back to skill-only matching.
Explainability	Log inclusion/exclusion reasons, filter responsible, semantic similarity scores.
Output	Candidate course IDs + filter summaries.

Objective C - Dynamic Course Ranking Engine

Purpose
Rank candidate courses according to user intent through dynamic weights, normalized features, and PRD-defined edge rules.

How We Solve It

Component	Solution Approach
Feature extraction	Compute normalized features: rating, popularity, freshness, duration fit, skill-match, learner flags.
Weighting	Rule-based PRD templates + optional LLM adjustments; future ML ranker.
Scoring	Final Score = $\Sigma (W_i \times F_i)$, sorted descending; tiebreak on rating/popularity.
Edge cases	Niche topics → W_match = 1. New courses → freshness boost. Low duration-confidence → reduce duration weight.
Explainability	Log feature vectors, weights, and ranking decisions.

Objective D - Item-Level Retrieval (Vector Search + Cascading Confidence)

Purpose
Retrieve semantically relevant item-level content (videos/readings/transcripts) from ranked courses using vector search and cascading retrieval.

How We Solve It

Component	Solution Approach
Pre-embedding pipeline	Clean transcripts/readings, chunk (300–500 tokens), embed, store in vector DB.
Retrieval strategy	Cascading batches: Batch 1 (top 5), Batch 2 (6–20), Batch 3 (long tail) triggered by confidence threshold.
Item scoring	S_item = F_sem × M_match × D_penalty.
Batch confidence	S_conf = avg(top S_item) or min(bucket-wise) for curations.
Stopping rule	Stop once S_conf ≥ threshold; otherwise move to next batch.

Output	Retrieved items sorted by S_item, enriched with timestamps, module/section context.

Objective E - Output Assembly & Delivery

Purpose
Transform ranked courses or retrieved items into user-facing recommendations or curated pathways with consistent structure and metadata.

How We Solve It

Component	Solution Approach
Recommendation builder	Sort by final_score; include metadata (thumbnails, partners, duration, badges).
Curation builder	Order items by pedagogical buckets (intro→core→advanced) with narrative transitions.
Metadata enrichment	Attach timestamps, module hierarchy, duration, logos.
Score-to-UI mapping	Convert S_item and final_score into display ordering.
Feedback hook	Connect UI interactions to Objective F for refinement.

Objective F - Feedback Loop & Search Refinement

Purpose
Incorporate user feedback into the search pipeline by updating constraints or intent and re-running the appropriate stages.

How We Solve It

Component	Solution Approach
Feedback classification	LLM identifies constraint update, override, or intent shift.
Requirement update	Adjust duration, difficulty, language, format, topic/domain.
Pipeline restart	Constraint updates → restart at B; Intent changes → restart at A.
Strict constraints rule	Feedback constraints cannot be relaxed once stated.

Objective G - Data, Embeddings & Infrastructure (Foundational Layer)

Purpose
Provide embeddings, vector storage, metadata refresh, and system monitoring required for all upstream objectives.

How We Solve It

Component	Solution Approach
Embedding pipeline	Batch and event-driven pipelines to maintain fresh embeddings.
Vector store	Scalable index with metadata filters and fast kNN retrieval.
Metadata management	Daily refreshes, schema normalization, validation.
Monitoring	Latency, index freshness, cost dashboards, accuracy tests.
Orchestration	Databricks jobs + cloud functions for automation and scaling.

Assumptions

Area	Assumption
Ontology & metadata	Skill taxonomy and metadata refreshed daily and sufficiently complete.
Embeddings	All text content accessible for embedding; vector store supports metadata filtering.
Thresholds	Semantic and confidence thresholds tuned empirically by domain.
Logging	SOC supports anonymized logs for tuning and monitoring.
Latency	B → D stages complete within ~1.5–2 seconds.
UI	UI accepts JSON formats produced by Objectives D & E.

Challenges & Limitations

Challenge / Limitation	Impact
Metadata gaps	Affects extraction (A), filtering (B), ranking (C), and output accuracy (E).
Ontology mismatches	User phrasing may not align with Coursera taxonomy.
Semantic retrieval noise	Item-level retrieval may surface contextually irrelevant chunks.
Duration inaccuracies	Weakens filtering (B), ranking (C), and item penalties (D).
Transcript unavailability	Reduces retrieval quality and S_conf reliability.
Niche topics	Sparse metadata and embeddings reduce relevance in B–D.
Drift	Requires ongoing re-embedding and metadata refresh (G).
Constraint propagation	Errors in A flow through B→C→D; corrected only via F.
No personalization (MVP)	Org-level rather than learner-level relevance.
Infra/cost constraints	Embedding and vector index maintenance is resource-intensive.