Problem Statement
Coursera catalogue has over 16000 courses. A key challenge for Enterprise teams is surfacing the right content when they need it. The reasons for the challenge are multifold - we don't have the right search algorithm that can reliably go through our content data, further we don't have any view into item level data. The problem is only exacerbated by limited course level skill metadata availability.

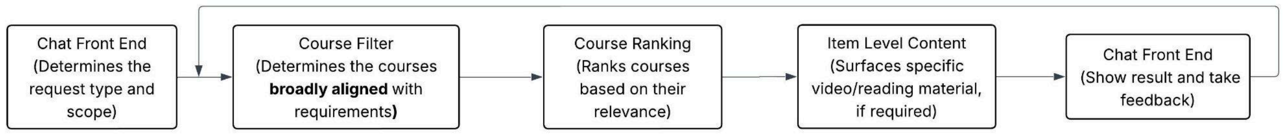
Hence when a user searches for a very specific topic, even when we have content on that topic in some course, we fail to surface it up for the user. Also, it is practically impossible to curate content from the entire catalogue manually.

Through AI led curations, we are essentially solving the discoverability problem.

Solution Design

The solution is conceptualized in 5 phases listed below. We will deep dive into each of the phases in the later sections.

- **Chat Frond End** - A natural language chat interface which goes back and forth in a human-like conversation to gather requirements from users.
- **Course Filter -** The first backend step whose function is to get the right courses from where we can do deep dive.
- **Course Ranking** - We rank the courses as per their relevance and adherence to user requirements.
- **Item/Course Level Content -** Surfacing item/course level recommendations which is highly specific to the user requirements
- **Chat Front End -** Displays the results back to the user and takes feedback. If the user wants changes to the results, it should rerun the search incorporating human feedback.


Chat Front End (Determines the request type and scope)
Course Filter (Determines the courses **broadly aligned** with requirements)
Course Ranking (Ranks courses based on their relevance)
Item Level Content (Surfaces specific video/reading material, if required)
Chat Front End (Show result and take feedback)

<u>Part 1: User Request Types / Workflows</u>

The product as a whole has the following workflows or types of user requests that it would need to cater:

1. Item Level Curation
2. Item Level Recommendation
3. Course Level Curation
4. Course Level Recommendation

A recommendation request essentially means "best content options" for XYZ topic. A curation request would have a pedagogical order or “pathway" for a sequence of content to build skills in XYZ topic.

The request could also be either item level or course level. A course level request would only give recommendations and curations of course. It would leverage our existing specializations or curations (available in Enterprise Highspot) made by Solutions Consulting team, whenever relevant or applicable. Thus a course curation request would look like the below

- Course 1 - Course 2 - Course 3

For item level requests, we will need to go one level deeper into individual items i.e., specific videos or reading materials within a course. The important thing to keep in mind here is that item level requests would analyze items spread **across** different relevant courses and stitch together curation/recommendation basis **specific** user requirements. It may be the case that an entire course is highly relevant to user requirements, along with ancillary items from other different courses.
Thus, an item level curation request could look like any of the below.

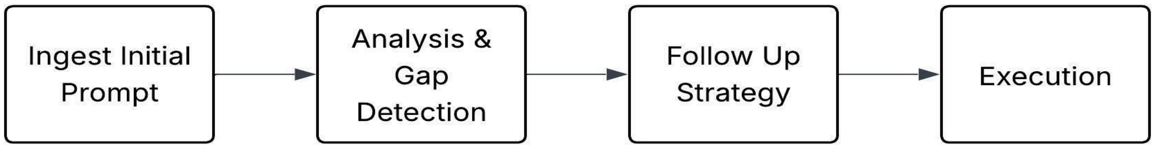
- Item 1 - Item 2 - Item 3 - Item 4
- Item 1 - Course 1 - Item 2 - Item 4
- Course 1 - Course 2 - Item 1 - Item 2

Part 2: Chat Front End

The goal of chat workflow is to extract the "Must-Have" requirements without overwhelming the user.

To solve this, we will use a **"Reactive approach”** that is, to protect user experience, instead of asking a standard list of questions, we will only ask what is missing from the initial query. Also we will limit follow up questions to a maximum of 2 turns. This would require prioritization and limiting to only essential questions.

The chat workflow is as follows:



A simple example visualization of the chat workflow is as follows

```HTML
```

```
Interaction Step (User Request)
User: "I need to train my finance team on Python."

Backend Step
Skill Identified: Python
Audience: "Finance team" (Implies need for business context,
maybe not software engineering).
Missing: Proficiency Level, Language, Time commitment.

Interaction Step (AI Response and Follow Up)
Bot: "Understood. You want curated courses on Python suitable for
Finance Team. To give you the best options, could you please give
some more context on the below:

   Are they beginners, or do they have some coding experience?

   Do you have a preference for short courses (under 10 hours) or
      a comprehensive certification?"

Interaction Step (User Request)
User: "I prefer short courses and the audience is at beignner
level."

Interaction Step (AI Response)
User: "Understood. Do you have any other requirements for the
content?"
```

At the backend, “Analysis and Gap Detection” would work in two steps.

Step 1: Identifying Workflow

The goal here is to analyze user requests and bucket it into one of the four workflows by identifying user intent. Examples are below

Example User Query	User Intent Identification	Workflow Bucket
"Show me the top-rated courses on Python."	Discovery. *"I have a general intent and want to see the best options"*	Course Recommendation

"I need to learn Leadership skills for managers."		Course Recommendation
"What is new in the catalog regarding GenAI?"		Course Recommendation
"Create a learning path to become a Data Scientist."	Transformation. "*I want to go from Point A to Point B over a long period."*	Course Curation
"I'm a beginner. Give me a roadmap to learn Java from scratch to advanced."		Course Curation
"How do I fix a #DIV/0 error in Excel?"	Just-in-Time Solution. *"I am stuck and need an answer NOW."*	Item Recommendation
"Show me a video explaining the 4 Ps of Marketing."		Item Recommendation
"I need a quick refresher on Python List Comprehensions."		Item Recommendation
"I need to prepare for an interview. Give me videos on 'Weaknesses', 'Strengths', and 'Salary Negotiation'."	Hyper-Personalized Study. *"I know exactly which specific topics I want, and I want them in a specific order."*	Item Curation
"Compile a reading list on the history of Blockchain and then its modern use cases."		Item Curation

Step 2: Gap Analysis
The crucial purpose of the chat front end is to extract all relevant user requirements before launching the search. To do this, we need to ask the right follow up questions, if the initial user prompt is incomplete.
We are categorizing the follow up questions into Hard and Soft Gates. If any of the parameters from Hard Gates are missing in the initial user prompt, we must mandatorily ask it as the first follow up questions. The Soft Gates are the second follow up questions, which can be optionally asked if the user requirements are still unclear.

Category A: Hard Gates
Minimum requirement to launch a Search

1. **Topic/Skill:** (e.g., "Python", "Negotiation").
2. **Proficiency Level:** *“What is the proficiency level that you’re looking for?”.*
3. **Duration Constraints:** *"Do you have a time limit in mind?"*

Edge Case: Vague Requirements [Unable to Determine User Intent in Stage 1]
If the user requirement is highly vague eg “I need project management” we will need to ask follow up questions to determine which of the 4 workflow buckets the request will fall under.

The following are the examples of questions can be asked as follow up:

*Are you looking for **upskilling** from scratch, or do you need **specific resources** to solve an immediate problem?*

- If "Upskill from scratch": Lean towards **Course Curation**.
- If "Solve immediate problem": Lean towards **Item Recommendation**.

Category B: Soft Gates
We ask these as a second follow up question ONLY if the user is vague or the requirements are still not clear.

1. **Match:** *"Do you need an exact match to your topic or are you okay with related content?"*
2. **Pedagogical Format:** "Do you prefer hands-on or theory?".
3. **Domain:** *" Are you looking for training content for a specific role or team?"*

Contextual Requirements

Every other requirement that may be mentioned by users are Contextual Requirements. Meaning, they are only used as filters if the user **specifically** mentions them. We will not ask follow up questions on these, by default. The full list of filters are explored in Part 3.

For example, "I want Python courses in *Spanish Language*". Here the Spanish Language becomes a Contextual Filter and applicable just for this search.

Part 3: Course Filters

The goal of this step is to map out user requirement parameters with enterprise datasets columns to filter relevant courses.
It is practically impossible to search our entire catalogue for every user request, especially if it is an item level request. Thus, we need to first identify the right set of courses which could have the content that the user is requesting for.

To do this, first we need to go wide and capture all the courses that could be relevant to the skill, so that we do not potentially miss out on the best/most relevant content. Then we narrow it down to the specific user requirements.

Step 1: Skill Filters
We will identify the "Universe" of relevant course content for the skill. Skill filters are **permissive** and **semantic.** We want to cast a wide net here, so that we do not miss out on any content even if the keyword is not explicitly mentioned. Hence, we are relying primarily on Fuzzy Match/Semantic Search.

We will use the following Databricks columns as filters.

Filter Name	Primary Column	Logic / Operation
Explicit Skill Tags	allskills _	**Fuzzy Match / Contains.**Check if user's requested skill exists in this comma-separated list.
Primary Domain	course_primary_domain	**Exact Match**.High-level filtering (e.g., "Data Science" vs "Business"). useful for disambiguation.
Sub-Domain	course_secondary_subdomain	**Fuzzy Match.**More granular (e.g., "Data Analysis" within Data Science).
Semantic Content	learning_objectives	**Semantic Search.**Used when all_skills misses. E.g., User asks for "Pivot Tables", but the skill tag only says "Excel". This column bridges that gap.
Semantic Content	course_desc	**Semantic Search**. Add on to learning objectives and look into course description section for additional context

Step 2: Ancillary Filters
Once we have shortlisted the courses as per the skills, we need to narrow down the relevant courses based on **contextual requirements.** Ancillary filters are **restrictive** in nature. The following list of filters would be applicable. All the filters are from Cou

Filter Category	Filter Name	Column Name	Logic / Operation
Proficiency	Difficulty Level	course_difficulty_level	Exact Match.(Values: BEGINNER, INTERMEDIATE, ADVANCED).
Language	Source Language	course_language_root_cd	Exact Match.Primary language of instruction.
Language	Translated Languages	machine_translation_language s	Contains.Check if the user's language exists in the available subtitles list.
Duration	Course Duration	total_duration_minutes	Range.Use this for strict time limits (e.g., "under 30 mins").
Quality	Star Rating	course_star_rating	Threshold (>=).e.g., "Rated 4.5+".
Freshness	Launch Date	course_launch_ts	Date Range.For "Newest courses" requests.
Format	Assignment Load	graded_assignment_questions _count	Threshold (>0).Filters for courses with rigorous testing vs. passive viewing.
Credibility	Partner/University	partner_names	Exact/Fuzzy Match.e.g., "Courses from Yale or Google".

All filters are from the below Databricks query
https://coursera-data-dev.cloud.databricks.com/sql/editor/95209f01-d093-447f-bd59-8b8fbe98b6ea?o=4788768134532473

However, we need to have **relaxation logic** as a redundancy mechanism in case there are not relevant courses satisfying the user criteria. The user redundancy mechanism is as per below:

For Ancillary Filters, we might need to widen the net if few/no results are found as per exact user requirements. The Relaxation Logic is as per below

- Level 1 (Strict): Match all Ancillary filters exactly.
- Level 2 (Relaxed): Drop "Duration" and "Format" constraints; keep "Difficulty" and "Language".
- Level 3 (Fallback): Keep only "Skill" matches (ignore all metadata except maybe Language).

Additionally, courses with a high (threshold to be defined later) degree of semantic match with learning_objectives, course_desc can automatically be put under Relaxed criteria.

Part 4: Course Ranking

The goal of this step is to have the most relevant courses ranked first. This is to ensure that the computing resources are being optimally utilised where there is higher probability of getting the right content.

The way courses would be ranked is dependent on *what the user wants to see*. User requirements could be anything from “best courses” to “short intros” etc. Hence, the ranking algorithm needs to be **dynamic**, as it's practically impossible to cover all scenarios and we can’t hardcode the same.

Approach: We will need to implement an ML model which analyzes user intent from their requirements and **dynamically assign weights** to predetermined features.

Features
The following features along with their normalization logic would be used for the course ranking algorithm. Normalization is required so that outliers do not overpower the final rank.

Feature Name	**Source Column**	**Normalization Logic**	**Why?**
F_rating (Quality)	course_star_rating	Min-Max Scaling (0-5 mapped to 0-1).	Captures "User Satisfaction."
F_pop (Popularity)	total_enrollment	Log Transformation (log(x+1)). Why? Prevents a "mega-course" with 1M learners from drowning out a great niche course with 10k learners.	Captures "Social Proof."

F_fresh (Freshness)	course_launch_ts	Time Decay Function (1/(1+age_in_months)). Result: New courses get ≈1.0, old courses decay towards 0.	Captures "Newness."
F_dur (Duration Fit)	avg_total_learning_hours	Inverse Decay Function. Difference to target time matters less with higher denomination	user_time - course_time
F_match (Skill Match)	All_skills, course_desc, learning_objectives, course_branch_module_desc		Captures "Relevance" through vector similarity search"
F_lfa (No of learner flags)	flag_count		Captures "Content Quality"

Formula

The final ranking algorithm would be a simple weighted average formula multiplying the dynamically assessed weights along with their normalized feature scores.

Final_Score = (W_rating*F_rating) + (W_pop*F_pop) + (W_fresh*F_fresh) + (W_dur*F_dur) + (W_match*F_match) + (W_lfa*F_lfa)

Examples:

Scenario A: "Show me the absolute best Python courses."
User Intent: Quality & Popularity matter most.

- W_rating: 0.5 (High)
- W_pop: 0.3 (Medium)
- W_freshness: 0.1 (Low)
- W_match: 1.0 (Always max for relevance)

Scenario B: "What's new in Generative AI?"
User Intent: Freshness is king. Quality/Popularity matters less (new courses have low stats).

- W_rating: 0.1
- W_pop: 0.0 (Ignore low enrollments for new items)
- W_fresh: 0.8 (Very High)
- W_match: 1.0 (Always max for relevance)

Scenario C: "I need a short intro to Marketing (under 5 hours)."
User Intent: Time constraint is the primary friction point.

- W_dur: 0.6 (High Penalty for long courses)
- W_rating: 0.3
- W_fresh: 0.1
- W_match: 1.0 (Always max for relevance)

Edge Cases: Niche/Newly Launched Courses

1. **Niche Course:** Niche courses pertaining to rare search topics must compete primarily on relevance to search criteria.

- If the search term is rare/niche (e.g., "3G Communicaton"), the algorithm should dynamically drop all non quality weights (W_pop, W_fresh, W_dur) to 0, even if the user mentions.
- The quality metrics W_lfa, W_rating would have low weights (0.2), even if the user mentions it.
- *Why?* In a niche market, popularity is irrelevant because the sample size is too small. Relevance F_match is the only thing that matters.

2. **Newly Launched Course:** Newly launched course deserves a boost over existing popular courses, if it covers **updated** content.

    - Boost if the New Course is Semantically Similar (e.g., >80%) to a Best Seller **BUT** covers *newer concepts relevant to the user query* (e.g., "New Features in Python 3.12" vs the Best Seller's "Python 3.9"), then Boost.
    - The boost lasts only for a specific time, e.g. 3 months.

Part 5: Item Level Retrieval
The goal of this step is to deep dive into courses to fish out item level curation or recommendation. We will need to balance speed and accuracy for the same.

Pre-requisite: Pre-Embeddings and Data Ingestion Pipeline
We need to pre-embed the transcript and reading materials we have and store them into a vector database to make the system scalable. We cannot be doing real time embedding/processing on queries as it would be too time consuming and expensive.

To update the embeddings any time a course is updated or a new course published, we will need to set up an **Event Driven data ingestion pipeline.**

Cascading Item Retrieval Logic

While higher ranked courses from Part 4 are more likely to contain the content we need, it is possible that the perfect content could be hidden away within a video of a lower ranked course. However, we also need to efficiently direct our computing resources.

Step 1: Dynamic Batching

Instead of a hard cutoff, we fetch courses in "Batches" from the Part 4 Ranking.

- **Batch 1 (High Probability):** Top 5 Courses. (Most likely to have high-quality, popular content).
- **Batch 2 (Discovery):** Ranks 6–20. (Likely specific/niche courses).
- **Batch 3 (Long Tail):** Ranks 21+. (Only searched if user intent is extremely obscure).

Step 2: Item Retrieval and Confidence Score

We will search and retrieve the item level content through Similarity Search proceeding on a batch wise basis mentioned above. The search would stop whenever we cross the Confidence Score threshold on the items retrieved. If the threshold is not met in batch 1, we go to batch 2 and so forth. If the threshold is still not met at the end of all ranked courses, we display the content with the highest Confidence Score (with disclaimer that this is the closest match).

The confidence score logic is elaborated below. To determine the confidence score, we need to analyze confidence in two levels:

1. Item Conf Score (S_item): Is the single item (video/reading material) good enough for the user query?
2. Batch Conf Score (S_conf): Is the group of videos **together** relevant to the user query.

Item Conf Score (S_item)
S_item would be determined through semantic match of the item embeddings with the user query. However, we need to account for user preferences on type of content (eg videos or reading materials) if applicable. Hence S_item would be:

S_item = F_sem (Semantic Match) * M_match (Format Penalty)

M_match -> 1.0 if no format specified or format match. 0.8 if format mismatch

Batch Conf Score (S_conf)
Since item level requests are of two types (curation and recommendation), we will need to derive two types of S_conf.

Scenario A: Item Recommendation Requests
S_conf logic: Average Relevance

For recommendations, the recommendation group is only as good as its average quality. If we find N distinct videos that all score highly against the user query, we have high confidence.

$$S\_conf = \text{Average } S\_item \text{ of Top N results}$$

Scenario B: Item Curation Requests
S_conf logic: Minimum of S_item across all “buckets”.
For curation, the entire curation is only as good as the weakest link of the curation.

Buckets are defined as per user requirement. It could be difficulty levels (easy, medium, hard). It could be pedagogical progression (eg reading list of blockchain), or it could be specific user topics.

$$S\_conf = \text{Minimum}(S\_item \text{ Bucket } 1, S\_item \text{ Bucket } 2, \ldots)$$

Duration Constraint: At this stage, we will need to account for the duration constraint, provided by the user for item level requests. However, this is not a hard constraint. Content with high Conf level (highly relevant content) will have relaxations to duration constraint.

Note: All Conf thresholds, relaxation levels to be determined later as we begin experimenting with data.

Part 6: Results and Feedback

The last step is to show the curated/recommended course or items back to the user. The recommendations along with their links and thumbnails need to be displayed.

1. For recommendation requests, the courses need to be displayed rank wise (most relevant/highly recommended) at top.
2. For curation requests, the courses need to be displayed in a serial format (e.g. topic 1, topic 2, topic 3 or easy, medium, hard) depending on the user request.

Additionally, we also need to keep the chat window open for further feedback from users. If the user provides any feedback, (eg “I need shorter courses”), we will need to incorporate that by modifying the duration criteria and keeping all other criteria the same and relaunching search. **Any user feedback at this stage, becomes non-negotiable strict criterias.**

Deployment and Roll-out Plan

Project assets

File	Description
course_recom_1	Databricks Output - Course Metadata
XDP2.0_Content.xlsx	Databricks Output - Course XDP
https://coursera.highspot.com/items/68faa712b9491dc8e2deed44	Enterprise Highspot (For Enterprise Curations)
Curation_Data Mapping for …	Inputs from Solution Consulting on Filters and Ranking (For reference)