Adaptive Microlearning System Architecture and Roadmap

Principal AI Architect Team

October 23, 2025

Contents

1 Executive Summary

- **Proposed Solution:** A retrieval-augmented microlearning platform that delivers ultra-targeted content snippets (text/video) and adaptive questioning, orchestrated by an AI agent. This approach maximizes factual accuracy and minimizes content length by grounding responses in our proprietary materials. It outperforms end-to-end trained alternatives by leveraging existing knowledge bases and focusing model capacity on pedagogy rather than relearning content.

- **Microlearning Efficacy:** The system emphasizes short, focused learning resources. Breaking learning into small chunks is empirically shown to improve efficiency and test performance (e.g. 20% higher scores with chunked content vs. one long session). By providing only the "smallest sufficient" video clip or document excerpt to answer a question, we cater to student attention spans and accelerate understanding.

- **Retrieval-Augmented Accuracy:** Using Retrieval-Augmented Generation (RAG), the AI draws on up-to-date, domain-specific content for each query. RAG mitigates hallucinations and builds user trust by citing sources. It also offers more control and lower cost than training a domain-specific model from scratch. This means our agent can answer complex academic questions with authoritative references, a critical advantage over black-box generative models.

- **Adaptive Learning Loop:** The agent operates in a loop of content suggestion $\rightarrow$ practice question $\rightarrow$ evaluation. After delivering a minimal resource, it poses formative questions and evaluates answers to check if the learner mastered the concept. Based on performance, it dynamically adjusts difficulty and selects the next microlearning nugget. This closed-loop design targets actual learning gains (knowledge θ increase) rather than just engagement time.

- **Data-Efficient and Scalable:** The roadmap begins with zero training data for recommendations, using heuristic and semantic search for cold-start. Over time, interaction data (which resources led to learning gains) feeds a feedback loop to improve content ranking via bandit algorithms. Crucially, we avoid early heavy fine-tuning; instead, we use lightweight adapters (LoRA) and reinforcement learning only when the data supports it. This modular approach keeps us agile as better base models or new data arrive.

- **Pedagogical Layer:** Beyond retrieval, the solution integrates pedagogical AI modules: question generation aligned with Bloom's taxonomy, high-quality distractors for multiple-choice, and AI grading with rubrics. These ensure each learning step is instructive and appropriately challenging. By instrumenting the system with Item Response Theory (IRT) models, we quantify question difficulty and student ability on a common scale, enabling personalization and measurement of actual learning outcomes (not proxy metrics like clicks).

- **Why It Wins:** Compared to alternatives, this architecture best meets our constraints. It avoids overwhelming learners with irrelevant material by using segment-level retrieval and summarization. It sidesteps the need for long videos or entire chapters for a single question. It can start delivering value on day one by leveraging existing content (no cold-start paralysis), and then improve via data-driven refinement (bandit optimization of content selection, targeted fine-tuning for style). This stepwise improvement plan focuses on measurable learning gains at each milestone, ensuring the system demonstrably boosts mastery rather than just providing a flashy UX.

2 Architecture Options

To ensure we select the optimal approach, we compared several architecture strategies against our needs (scalability, cold-start viability, learning impact, team fit). Table 1 summarizes four candidate solutions:

After evaluation, **Option A** (RAG with re-ranking and agent orchestration) emerged as the best foundation for our needs, with elements of **Option B** and **Option C** phased in to enhance pedagogy and adaptivity. **Option D** was deemed too costly and inflexible. Next, we detail the final architecture recommendation that combines the strengths of A, B, and C in a modular way.

3 Final Recommended Architecture

Figure 1 outlines the end-to-end system. The design is modular, comprising distinct layers for retrieval, content minimization, pedagogy, assessment, and agentic control:

Agent Planner (decides next step based on student profile and query) → **Retriever** (hybrid search in content repository) → **Segment Selector** (chooses minimal snippet) → **Learner Interface:** show resource or ask question → **Evaluator** (grades response, updates student model) → **Planner (next action)** (adjust difficulty/content choice)

Figure 1: Adaptive Learning System Architecture (Conceptual Flow)

- **Retrieval:** At the core is a **hybrid search** retrieval system over all company materials (textbooks, PDFs, video transcripts, etc.). We use **dense embeddings** (from a domain-adapted model) to capture semantic similarity, combined with **BM25 lexical search** for keyword matching (to handle acronyms or exact terms). User queries are first embedded and used to retrieve a broad set of candidate chunks (e.g. top 50) from a vector database. These results are then **re-ranked with a cross-encoder** that scores each (query, chunk) pair with deeper attention. This two-stage retrieval ensures high recall and precision: the hybrid dense+BM25 fetches any potentially relevant snippets, and the cross-encoder ranks them so the most on-point, authoritative snippet is on top. We also apply **Maximal Marginal Relevance (MMR)** to promote diversity in the top suggestions – avoiding redundant results and covering different facets if the query is broad. The output of retrieval is a short list of highly relevant content snippets (text paragraphs or video segments).

- **Content Minimization:** Rather than returning entire documents or long videos, the system **segments content into "micro-nuggets."** For PDFs and text, we chunk by sections or paragraphs (using logical subheadings as guides). For videos, we employ automatic speech recognition (ASR) to get transcripts, then use scene change detection or slide cues to identify coherent segments. We define a *sufficiency score* for each segment with respect to the query: a combination of semantic similarity to the question and coverage of its key points, normalized by segment length. The agent selects the segment (or top k segments) that maximizes answer coverage while minimizing duration/pages. **Length is capped** (e.g. prefer a 2-minute clip or half-page text if available). If a segment is still too long, the system can generate a **summary** or highlight to deliver just the necessary facts. This ensures no hour-long videos are suggested for a simple question. Over time, the content repository can be pre-indexed at the granularity of these micro-units, making retrieval directly fetch short snippets. The agent's response to

the learner will cite the source and possibly include a brief preamble (e.g. "I found a 90-second video clip that explains this concept."). By keeping content bite-sized, we adhere to microlearning best practices and reduce cognitive load on students.

- **Pedagogical Layer:** On top of content delivery, the system integrates AI-driven tutoring capabilities:
  - *Dynamic Question Generation:* After the learner reviews a resource, the agent generates a formative question to test understanding. Using few-shot prompts or a fine-tuned question generator, it can create various types of questions (e.g. conceptual MCQs for recall, open-ended questions for deeper application). We incorporate **Bloom's taxonomy** – for foundational queries, the agent might ask simple recall questions ("Define X"), whereas for advanced topics it asks analytical questions ("Compare X and Y in context Z"). The question generator is constrained to use the same information the student just saw, ensuring alignment.
  - *High-Quality Distractors:* For multiple-choice questions, the system generates plausible but incorrect options that target common misconceptions. A small model or prompt library ensures distractors are pedagogically useful (not too obvious, not too tricksy, but fair and instructive).
  - *Hints and Explanations:* If the student struggles or requests help, the agent can provide hints derived from the content ("Recall what the video said about...") rather than giving away the answer. After the student answers, the agent always gives an explanation or solution walkthrough – effectively a mini *worked example* if the student was wrong, to cement learning.
  - *Rubric-based Grading:* For open-ended responses, an LLM (possibly the same as the main model) will grade the answer against expected key points. We define rubrics for each question type (via prompt or fine-tuned classifier) so the AI can give partial credit and specific feedback ("You mentioned A and B, which is good, but you missed C"). The grader is tuned to be consistent with human grading standards on our content.

  These pedagogical tools ensure that simply watching a video or reading text is followed by active recall practice – crucial for durable learning. The AI tutor adapts its question difficulty based on past student performance, aiming to keep the challenge optimal (not too hard to discourage, not too easy to bore).

- **Assessment & Learning Analytics:** All student interactions feed into a learner model. We employ an **Item Response Theory (IRT)** framework to maintain an estimate of each student's ability θ and each question's parameters (difficulty b, discrimination a, guessing c). After each question, the student's θ is updated (e.g. via Bayesian or SGD) and item parameters are refined as data accumulates. This allows the system to quantify learning progress: if θ increases over successive sessions, that indicates real learning. We calibrate question difficulty to the student's level by using the b parameter – e.g. choose questions with $b \approx \theta$ for an appropriate challenge. We also track **item information** (how much an item reduces uncertainty in ability estimate) to decide which questions are most diagnostic. Over time, as students answer more questions, the system's estimates of their mastery become more confident (smaller standard error on θ). We can report metrics like *mastery of topic X: 80%* or *ability score improved from 0.5 to 1.2*. Importantly, **learning gains ($\Delta\theta$)** are measured over time and used as the primary success metric, distinguishing our focus on learning outcomes from shallow engagement metrics.

- **Agentic Orchestration:** An AI *planner/agent* oversees the whole loop, deciding at each turn what the student needs next. This could be implemented via a high-level policy (a prompt-based "meta-controller" LLM) or a scripted logic. The orchestration works as follows: When a learner asks a question, the agent (1) invokes the Retriever to fetch relevant content; (2) uses the Segment Selector to pick the best snippet and presents it (with a citation); (3) after content delivery, generates a question via the Pedagogical layer to test retention; (4) evaluates the response (using the grading component) and logs the result; (5) updates the student's profile (ability level, problem areas); (6) decides the next action. The next action could be another question (if the student hasn't demonstrated mastery), a hint (if the student is stuck), or moving to a new topic (if mastery is achieved or the query was fully answered). This agent is essentially a *policy* that may be refined over time (possibly with reinforcement learning). Initially it will follow an expert-designed script (e.g. always do one question after showing content; if score ¡ 80%, offer a remedial explanation; otherwise proceed). As we gather data, we can train this policy to be more optimal. The agent also handles **safety and accuracy checks:** if retrieval results are poor or a question is out of scope, it will avoid guessing and either ask for clarification or politely respond that it cannot find a confident answer (ensuring no hazardous misinformation is given).

4 Data Plan (Cold-Start to Flywheel)

Launching a learning system with no existing content recommendation data is challenging. Our data strategy bootstraps the system with heuristics and gradually transitions to data-driven optimization as we gather feedback:

- **Cold-Start Content Recommendation:** Since we lack historical "which resource answers which question" labels, we start with content-based heuristics. Every piece of content (videos, PDFs, chapters) will have metadata (titles, tags, transcript keywords). When a student asks a question, we rely on **semantic search** (embedding-based retrieval) to surface potentially relevant segments. To enforce microlearning, we filter out segments above a length threshold. We also utilize **coverage heuristics**: e.g. if a PDF chapter covers multiple topics, locate the subsection whose heading or content matches the query keywords. For videos, if the transcript contains the question's key terms, pick that timestamp range. In absence of usage data, these rule-based approaches (keyword matching, content outline following) provide initial recommendations that are at least topically relevant.

- **Weak Labels and Expert Seeding:** Early on, we generate some synthetic training signals. For instance, we can take a set of representative questions from our domain and have subject matter experts label which part of the textbook or which video best answers each. This yields a **reference dataset** of (question $\rightarrow$ best minimal resource) pairs. If experts are scarce, we can use the AI itself to suggest an answer and then have a human verify that the snippet indeed addresses the question. These labeled pairs serve to train a preliminary content ranker or at least evaluate our retrieval. Additionally, we can use Q&A pairs from historical assessments to map onto content: for each known question, see if its answer is contained in any document. If yes, label that document section as relevant. These are "weak" labels since they assume answer text implies the resource is a good teaching snippet.

- **Metadata and Taxonomy Filters:** Our company materials likely have an existing taxonomy (chapters, topics, tags). We leverage the question's keywords to narrow search space by topic. For example, if the question mentions "mitosis", restrict retrieval to biology content

rather than searching all subjects. This improves precision and cold-start relevance. Over time, user interactions will confirm or refine these mappings (if users asking about mitosis consistently click a particular video segment, we'll strengthen that association in the index).

- **Active Learning Loop:** As soon as students begin using the system, every interaction provides data to refine it. We will log (privately and with consent) each question, the content snippet shown, whether the student got the follow-up question correct, and whether they requested additional help. Using this, we can train a **learning-to-rank model** for content recommendations: features might include (query and snippet text embeddings, snippet length, popularity score) and the target can be a measure of success (e.g. 1 if the student answered correctly after viewing snippet, 0 if not). This is a noisy signal but across many users and questions it will indicate which content tends to lead to learning. Periodically retraining the ranker on this data will turn the cold-start heuristic approach into a personalized, data-driven approach. The more the system is used, the better it should get – a virtuous cycle where better recommendations yield better outcomes which yield more data.

- **Dataset Creation and QA:** To accelerate the flywheel, we will conduct focused data sprints. For example, define a clear labeling protocol for "minimal sufficient resource" (it must fully answer the question and be as concise as possible). Have multiple educators label a common set of Qs to ensure consistency (measure inter-rater agreement, refine guidelines if needed). Use active learning: pick some questions where the retrieval is uncertain (scores borderline) to label, as those labels will be most informative for training. All curated datasets will come with data cards documenting their source and any limitations, which will guide future model fine-tuning and avoid training on faulty data.

- **Leveraging Existing Q&A Data:** We do have historical Q&A/assessment logs which can be repurposed to bootstrap the pedagogical AI:

  - Past questions can help fine-tune or prompt the question generator: we can condition it to produce questions similar in style and difficulty to those in our archives (assuming those are high quality). Also, if we have solutions or explanations for those, we can fine-tune the explanation style.
  - Past student answer data can train the grading model or at least provide exemplars for few-shot grading. For example, for a given question, we have seen common incorrect answers – the AI grader can be taught to recognize those and assign partial credit if appropriate (or provide targeted feedback).
  - Difficulty estimates from past data: if we know 80% of students got question Q1 right and only 30% got Q2 right, we have an initial sense that Q2 is harder. We can seed our IRT model with those estimates for the corresponding topics, so that initial personalization has some grounding (e.g., the agent won't start a new user with known-difficult questions).

- **Human-in-the-Loop Refinement:** At key stages, we will involve human experts to review and correct the AI's outputs, which not only prevents errors from reaching students but also produces new training data. For instance, in early deployment, we might have an educator oversee the questions generated for a sample of sessions and flag any bad ones – these can be fed back into the training loop (with corrections) to improve the question generator. Similarly, any time the AI says "I don't know" or struggles, we might have a content developer provide the answer which we then add to the knowledge base for next time.

- **Privacy Considerations:** All student interaction data used for model improvement will be anonymized. We will aggregate learning outcomes for analysis without exposing individual identities. If any personal data is to be used (e.g. a user profile feature in contextual bandit), we will obtain necessary consent and follow data protection regulations. Additionally, any sensitive content (like proprietary PDFs) remains within our secure system (if using external APIs for AI, we either use ones with data privacy guarantees or avoid sending raw content).

5 Metrics & Evaluation

We define concrete metrics for each component of the system. These metrics will be computed at every milestone to quantitatively demonstrate progress. Crucially, we differentiate **learning metrics** from mere engagement:

- **Retrieval Effectiveness:** Using a test set of queries with known relevant content, measure:
  - *Recall@k:* the fraction of queries for which the correct answer-containing snippet is present in the top-k retrieved. This ensures our hybrid search isn't missing key info (goal: high recall so relevant info is almost always found in the candidates).
  - *nDCG@k (Normalized Discounted Cumulative Gain):* measures ranking quality by rewarding correct results appearing higher up. We want a high nDCG, especially at small k, meaning the agent usually finds the answer in the first few suggestions.
  - *Precision@1:* how often the very first snippet returned fully answers the question. This directly ties to user experience (did they get what they need immediately?).
  - *Coverage:* ensure the retrieval covers the breadth of our content repository. For instance, the percentage of syllabus topics for which the system has at least one relevant resource indexed (to identify any blind spots in our knowledge base).
  - *Time-to-first-result:* an operational metric – from question asked to snippet shown, the elapsed time. We aim to keep this low (a few seconds). This will be measured through logs; if it starts creeping up (due to long re-ranker or slow DB), we optimize.
- **Minimality Metrics:** These verify the system is delivering micro content:
  - *Median Resource Length:* track the median duration of video clip or length of text snippet provided. Goal: under a set threshold (e.g. ¡ 3 minutes or ¡ 1 page). A downward trend over milestones indicates improved trimming or segmentation.
  - *Overkill Rate:* the percentage of cases where the system suggests a resource that clearly exceeds the necessary length (e.g. gave a whole 30-page chapter when a 1-page summary existed). We want this as low as possible (target $< 5\%$). We will sample recommendations and label whether they were overkill to compute this.
  - *Compression Ratio:* for cases where we summarize or excerpt content, define ratio = original length / delivered length. Track the average or median. A high ratio (within reason) implies we are effectively compressing content. However, we also ensure via quality checks that compressed answers still contain correct info.
- **Question Quality:** We will evaluate the AI-generated questions on multiple criteria:

- *Expert Rubric Score:* We have educators rate a sample of questions on clarity, alignment to the learning material, appropriateness of difficulty, and cognitive level. Score on 5-point scale. We expect these scores to improve as we fine-tune the question generation (e.g. from an average of 3/5 at M3 to 4+/5 by M5).
- *Factual Alignment:* We verify that every generated question is answerable based on the content provided. If the AI asks something outside the scope, that's a flaw. We'll measure the proportion of questions where the answer can be found in the preceding resource (aiming for 100%). This can be partially automated by checking if keywords of the question appear in the snippet.
- *Distractor Quality:* For MCQs, measure if distractors function as intended. We can use statistics from student responses: a good distractor will attract some fraction of incorrect answers but very few correct students will pick it (indicating it's not a "gotcha" that confuses the knowledgeable). We can compute, for each wrong option, the
- *Bloom's Taxonomy Coverage:* Over many questions, ensure we're not stuck at one cognitive level. We might log each question's inferred Bloom category (using either manual labeling or an AI classifier) and see a distribution. For higher-ed, we expect a mix, with maybe 50% at understanding/applying, some at analyzing, etc., depending on subject. This is more of a check to ensure variety.

- **Assessment Reliability:** We treat AI grading and question selection as an assessment, subject to psychometric analysis:
  - *Item Discrimination (a):* Using IRT or classical item analysis, measure how well each question differentiates students by ability. We want reasonably high discrimination for most questions (in IRT terms, $a > 0.7$ for important questions). If we find many questions with very low discrimination (e.g. everyone gets it right or wrong regardless of ability), we adjust difficulty or discard those items.
  - *Item Difficulty Calibration (b):* Compare the observed difficulty (percentage of students who answer correctly) to our intended difficulty or predicted difficulty. We aim to tighten this calibration over time. A metric could be the mean absolute error between predicted $p(\text{correct})$ from the model and actual outcomes. Reducing this indicates our ability estimation and difficulty tagging are improving.
  - *Test Reliability:* If a student answers a batch of questions, the consistency of the results (e.g. Cronbach's alpha or IRT marginal reliability) should be high. In practice, since questions adapt to the student, we might instead use a test-retest approach: if the same student repeats similar questions, does the score correlate strongly? We target reliability coefficients > 0.8, meaning the assessment is stable.
  - *Ability Estimate Convergence:* Track the standard error of θ for each student as they answer more questions. It should decrease with more data. For example, after 10 questions, maybe the average SE is 0.3; after 20 questions, 0.2, etc. If not, it may indicate inconsistent item behavior that we need to fix.

- **Learning Outcome Metrics:** Ultimately, we measure actual learning:
  - *Learning Gain ($\Delta\theta$):* Using the IRT model, track how much students' ability scores improve. We can compute the average gain per student over a course or the fraction of students who achieve a certain gain. We can also use normalized gain (e.g. $\frac{\theta_{\text{post}} - \theta_{\text{pre}}}{1 - \theta_{\text{pre}}}$ if

treating θ as probability of mastery) to gauge progress relative to remaining opportunity. Our goal is a positive gain for the majority of users, and higher gains than a control group not using the system (if such comparison is possible).

- *Mastery Progression:* Monitor how many topics or objectives each student masters over time. Mastery can be defined as θ above a threshold for that topic. We can report things like "X
- *Efficiency (Questions to Mastery):* How many questions or how much time does it take on average for a student to master a new concept? If our adaptivity works, this should decrease compared to baseline (e.g. if traditionally it took 10 questions to learn a concept, perhaps our tailored approach gets it done in 7 on average).
- *Engagement vs Learning:* We will collect engagement stats (session length, return rate, etc.) but interpret them in conjunction with learning metrics. We specifically avoid optimizing for "time spent" without learning. Instead, a useful composite metric might be *learning per hour* or per session. Ideally, as we refine the system, learning per unit time increases (i.e. students learn more in the same or less time, thanks to efficient teaching).
- *Downstream Performance:* If possible, correlate usage of our system with performance on external exams or course grades. For instance, if a class of students uses the tutor and their exam scores improve 10

- **Safety & Accuracy:** Ensuring no harm is done:
  - *Hallucination Rate:* We will manually review a sample of answers and check how many contain any incorrect information not backed by sources. Our aim is zero tolerance for hallucinations. Using retrieval as a backbone, we expect this to be very low; any instance found will be investigated. This can be measured as (# of incorrect or ungrounded statements) / (# of answers reviewed). We want $< 5\%$ at first, trending down to 0%.
  - *Refusal/Deferral Appropriateness:* When the system says it cannot answer, is it the right call? We'll create some unanswerable or outside-scope queries and see if the system appropriately refuses. And conversely ensure it doesn't refuse things it actually could have answered. We measure accuracy of these decisions perhaps by a confusion matrix on a set of test queries labeled "answerable" or "not answerable". We want high true positive and true negative rates for answerability detection.
  - *Bias and Fairness Checks:* We will instrument logs to see if there are any disparities – e.g. does the system perform worse on questions about certain cultures or on students of a certain background? This is hard to quantify, but we'll do periodic bias testing using curated test cases (e.g. ensuring examples or names used in questions span diverse groups). We might count any problematic outputs (like stereotyping in a generated example) per 1000 questions and aim to eliminate those entirely through prompt tweaks or fine-tuning.

Each milestone in development will include an evaluation report covering relevant subsets of these metrics, to track improvement and catch regressions early.

6 Stepwise Roadmap (8–12 Weeks, Shippable Increments)

We propose a rapid 12-week development cycle, broken into 6 bi-weekly milestones (M1–M6). Each milestone delivers a functional increment and associated evaluations:

M1 (Weeks 1–2): RAG Baseline with Minimality – *Goal:* Build the core question-answering functionality with strict adherence to concise answers.

- **Features:** Set up the vector database (or use an in-memory index for prototype) and ingest a sample of content (e.g. a textbook chapter and one video transcript). Implement the embedding model for semantic search and basic BM25. The system should accept a user question, retrieve top-k text snippets or video segments, and return the top answer snippet with a source citation. We will also implement a simple length filter (ignore snippets beyond a certain size) to enforce micro answers.

- **Deliverables:** A demo CLI or simple web UI where one can input a question and get an answer snippet with citation. Initial evaluation on a few known Q&A pairs to see if the correct info is retrieved. We will produce an evaluation report including metrics: e.g. Recall@5 on a small test set, and examples of outputs. Also, a list of failure cases (if any questions didn't get answered or had hallucination) with analysis.

- **Success Criteria:** Even at this stage, the system should answer at least 70% of straightforward factual questions correctly using the provided content. The answers should be reasonably concise (e.g. median length ¡ 100 words). Any instance of the system not finding an answer should result in an "I don't know" rather than a made-up answer. We'll do a quick internal user test: ask 10 sample questions; expect at least 7 answered well, 0 answered with hallucination (others either correctly refused or partially answered).

M2 (Weeks 3–4): Enhanced Retrieval & Segmentation – *Goal:* Improve relevance of results and handle multimedia sources properly.

- **Features:** Integrate a cross-encoder re-ranker to improve retrieval precision (particularly for the top result). Add the pipeline to process a video: transcribe via ASR and split into chunks (with timestamps). Similarly, parse PDF documents into sections. Implement the Segment Selector logic that chooses the best chunk among retrieved pieces based on the sufficiency score (coverage vs length). Output answers in a structured JSON including metadata like source name, and for videos, a timestamp or URL for the clip.

- **Deliverables:** Updated prototype where the answers are noticeably more relevant (thanks to re-ranking). For example, if in M1 the correct answer was sometimes buried at rank 3, now it should surface to rank 1. A set of test queries with their top-5 retrieval before and after re-ranking to show improvement. Also, demonstrate the system returning a video clip (could be a YouTube link with start/end parameters or an internal player reference) for a video question. Provide a second report focusing on retrieval metrics (nDCG, Precision@1 improvements) and verifying segment extraction is working (e.g. "Question X -¿ Video Y 00:45-02:15").

- **Success Criteria:** Offline, nDCG@3 should improve over M1 (say from 0.6 to 0.8 on our mini test set). The system should be able to answer at least one question via a video segment, demonstrating multimodal capability. All answers should still be concise – if a video is 10 minutes but we only need 1 minute, the system must return the 1-minute clip. Latency might increase with re-ranking, but we aim to keep total response under 5 seconds for now. If we find any major drops in speed, we consider caching or narrower initial retrieval.

M3 (Weeks 5–6): Pedagogy Layer (Q&A Generation) – *Goal:* Close the loop by asking the student a question and evaluating the answer.

- **Features:** Implement question generation after content delivery. Likely use an existing LLM (GPT-4 via API or a smaller fine-tuned model) with a prompt that includes the snippet and asks for a relevant question. For grading, implement a simple rule-based evaluator: for multiple-choice, check answer directly; for open text, perhaps use keyword matching or an AI judging if the essential content is present. Integrate this so that after showing the snippet, the system prompts the user with a question and processes their response. Update the student model (even a simple score tally).

- **Deliverables:** An interactive demo: one can ask a question, get a snippet, then the agent asks a follow-up quiz question. Provide correct and incorrect answers to see how it responds (should acknowledge correct or give correction if wrong). A set of sample QA pairs generated to inspect quality (and perhaps have an educator score them against our rubric). Metrics include initial question quality scores (maybe our SME gives an average of 3/5 and notes common issues like too simplistic phrasing), and grading accuracy on a small set of known answers (if we have a few sample student answers, see if our auto-grader marks them right/wrong appropriately).

- **Success Criteria:** The system engages the user with at least one follow-up question per query session. In a test with 5 users, if we intentionally answer 10 generated questions (some correctly, some incorrectly), the agent should correctly identify at least 90% of those (since simple questions should be straightforward to grade). Any blatantly bad questions (e.g. nonsensical or unrelated) should be under 10%. We also expect to see evidence that the student model is updated (even if just logging "Student proficiency up" or similar). This milestone essentially proves the concept of learning assessment, albeit primitive.

M4 (Weeks 7–8): Adaptive Selection (Bandit Prototype) – *Goal:* Personalize content selection using multi-armed bandit or simple reinforcement learning.

- **Features:** Incorporate a mechanism to choose between multiple content options or strategies dynamically. For instance, if there are two good snippets from M2's retrieval, the system can randomly pick one initially, observe if the student answered correctly, and then update weights favoring the snippet that led to a correct answer. This can be done with a contextual bandit: context = question + student ability, actions = content segment A vs B, reward = 1 if student answered follow-up correctly. Implement a basic Thompson Sampling or ε-greedy approach to update action preferences.

- **Deliverables:** A simulation or offline experiment demonstrating the bandit learning. For example, create a fake scenario where snippet A has a 70% chance of leading to correct answer and snippet B has 50%. Show that over repeated trials the bandit learns to favor A (e.g. selection probability of A goes from 0.5 towards ¿0.8). If possible, a very small-scale live test: e.g. with two questions that have two possible tutorial snippets, use the bandit to adapt within even a single user session if multiple attempts are allowed. Provide a report with learning curve graphs (regret decreasing, reward increasing).

- **Success Criteria:** The bandit logic should be functioning and not breaking core QA flow. We should see that it makes different decisions for different contexts if appropriate (e.g. if a student already knows a topic, maybe it chooses a harder question; if not, an easier one). This is hard to fully validate manually, but at least the simulation result should align with theory (learning the better content). Importantly, we ensure this adaptive layer doesn't degrade user experience: if it's uncertain, it can still default to a reasonable choice. Any extreme

exploration (like showing irrelevant content) must be curbed by our safety constraints from design. By end of M4, we effectively have an MVP that personalizes content selection and can articulate why (via the learned value estimates).

M5 (Weeks 9–10): Refinement & Fine-Tuning – *Goal:* Improve the system quality via fine-tuning and finalize any optional features.

- **Features:** At this stage, we have collected some data (from internal testing or a small pilot). We attempt fine-tuning the question generator or the answer explanation style using domain-specific data. For instance, fine-tune an LLM on a set of QA pairs from our content so that it naturally uses the terminology and style we want (this could be done with LoRA on a smaller model, given limited data). Introduce the fine-tuned model into the pipeline and do side-by-side comparisons with the prompt-based outputs. Also, refine the student model update to a proper Bayesian update or use a simple 3PL IRT estimation for ability (even if offline).

- **Deliverables:** Fine-tuned model files and documentation (e.g. "LoRA adapter on Llama-2 13B for our domain questions"). A report comparing outputs: e.g. "Original question: ... vs Fine-tuned question: ..." and experts' feedback that the latter is better structured. Updated evaluation metrics: we expect question quality rubric score maybe improved to 4.0, and grading consistency improved if we fine-tuned the grader. Additionally, an A/B test plan for a larger pilot (if we're gearing up to deploy, how will we measure success with real students? define control and treatment).

- **Success Criteria:** Fine-tuning should show a tangible improvement (or we decide it's not worth deploying). For example, if fine-tuned model reduces token count in prompts (cost saving) and outputs are equally good, that's a win. Or if it produces better questions (maybe fewer edits needed by SMEs in review). We should also ensure that we haven't introduced regression: no increased hallucination or slower performance. The system at M5 should feel more polished in responses. Additionally, by now all core components are in place; we should be ready to test with actual users. So a criterion is that we have a go from internal QA and stakeholders to move to a pilot, meaning they've seen the system handle a variety of scenarios end-to-end.

M6 (Weeks 11–12): Production Hardening – *Goal:* Prepare for scaling to real users with reliability, safety, and monitoring.

- **Features:** Implement final safety filters (e.g. if user asks something inappropriate or outside our scope, ensure a safe refusal). Set up logging and analytics pipeline to monitor metrics in production (nDCG, accuracy, etc., perhaps sampled). Stress test the system for concurrency (can it handle 100 simultaneous users?). Finalize integration aspects like API endpoints or LMS integration if needed. Create user-facing documentation or help (since it's an educational product, maybe a tutorial for students on how to use the AI effectively).

- **Deliverables:** Deployment scripts/infrastructure configuration (so we can reliably deploy the system on cloud or on-prem). Monitoring dashboard screenshots showing real-time metrics (even if just on test data). A security and privacy report confirming compliance with requirements (e.g. we did a security audit for data storage, ensured using secure channels, etc.). Also, a plan for scaling: e.g. which components need horizontal scaling (vector DB, LLM API throughput) and how we will handle that as usage grows.

- **Success Criteria:** Before broad launch, run a pilot with e.g. 50 users for a week. Success would be: no major crashes or bugs, feedback from users is generally positive (they found answers helpful, liked the questions). And ideally some evidence of learning improvement even in that short pilot (maybe compare pre vs post quiz for pilot participants). All critical metrics from section 5 should be at acceptable levels: e.g. hallucination rate 0%, refusal accuracy high, learning gain positive. We also expect that the team is comfortable with maintainability: everything is documented, and the system can be easily retrained or updated with new content. With these boxes checked, we would be ready to scale to more users and continuously improve based on real-world data.

This roadmap ensures that at each step we have a working product we could demonstrate, and we mitigate risks early (by testing fundamental pieces like retrieval quality and user interactions in isolation before making it more complex).

7 Reinforcement Learning Design (Practical)

We will employ reinforcement learning (RL) in a controlled, data-driven manner to optimize the agent's behavior, primarily for content sequencing and adaptation:

- **Reward Definition:** We design a reward that captures our objectives:

  $$R = w_1 \cdot (\text{learning outcome}) + w_2 \cdot (\text{brevity bonus}) - w_3 \cdot (\text{error penalty}) - w_4 \cdot (\text{latency cost}).$$

  Specifically, the learning outcome could be proxied by quiz success (1 for correct on first try, 0 for incorrect, maybe 0.5 for correct after a hint). This directly encourages the agent to help the student get answers right through learning. The brevity bonus gives a small positive if the content shown was short (to reinforce minimality). The error penalty is large negative if the agent provides a wrong or hallucinated answer (to enforce using retrieval and not guessing). Latency cost is a tiny negative per second or per additional step, to discourage overly long interactions when not needed. These weights w_i will be tuned experimentally or set by policy (e.g. we value learning gain far more than speed, but speed still matters).

- **Contextual Bandit to Start:** We initially treat each decision in isolation (myopic optimization) – a contextual bandit setup. The context (state) can include the student's current ability level, the difficulty of the content just seen, the performance on last question, etc. The action could be something like "show a harder question" vs "review another example" vs "move to next topic". Once the student responds, we get an immediate reward (e.g. got it right with high confidence = high reward). This bandit approach simplifies the RL problem since we don't have to consider long-horizon credit assignment – each step is optimized locally. It's suitable because each interaction's goal (did the student learn this piece?) is relatively contained. In fact, prior work in educational technology has shown that contextual bandit algorithms can effectively personalize learning content and increase overall learning gains.

- **Transition to Full RL (if needed):** As we accumulate data, if we see the need to optimize sequences (not just one step), we can expand to a full RL formulation. Here, the state would carry over (e.g. the student's updated ability θ and progress on various topics), and rewards might be given at the end of a session or sequence (e.g. overall improvement in θ). We could then use policy gradient or Q-learning methods to learn an optimal policy for long-term learning gains. However, this requires a lot more data (many trajectories) or a simulated

environment. We will attempt this only if simpler approaches plateau. The nice part is we can simulate to some extent: using our student model, we can generate synthetic students and run episodes to train an RL policy. This simulation-based RL could be an R&D project in parallel without risking user experience.

- **Off-Policy Evaluation:** Before deploying any learned policy to real students, we will test it offline with historical data. Using techniques like Inverse Propensity Scoring (IPS) or more advanced Doubly Robust estimators, we can estimate how a new policy might have performed if it had been in place during past interactions. For example, we can use the logs of what content was shown and whether it led to success, then evaluate if our new policy's choices (for those same contexts) would have resulted in success (re-weighted by how probable those choices were under the logging policy). This helps catch a bad policy (one that might, say, consistently choose too-hard content) before any student sees it.

- **Safety Constraints:** We hard-code certain constraints into the policy to ensure safety regardless of learning: e.g., the agent must always present content that has a minimum relevance score and never present content flagged as inappropriate. These act as guardrails so the RL agent can only explore within reasonable bounds (no wild deviations like showing random Wikipedia pages). We may also constrain the action space – for instance, not allow the agent to skip necessary steps (like not jumping to a new topic if current one isn't mastered, unless we have evidence the student is saturated).

- **Exploration vs. Exploitation:** In education, we must be careful: too much exploration (trying ineffective strategies often) wastes student time; too little and we might miss better methods. We will likely use an algorithm like Thompson Sampling which naturally balances this by maintaining uncertainty estimates for each action's value. As confidence in one action being better grows, exploration tapers off. Additionally, we can use the student model as a guide: exploration is safer if we think the student can handle some diversion, but if a student is struggling, the agent should exploit known helpful content rather than "experiment" at their expense.

- **Reward Shaping and Multi-Objective RL:** Our reward is multi-component, which can be tricky if the agent tries to game one part at cost of another. We might use a weighted sum as above, or we could adopt a more constrained approach (e.g. make it a constrained optimization: maximize learning gain subject to minimality constraint). Alternatively, we can train a separate "reward model" via supervised learning to combine these factors into a single scalar – akin to how RLHF uses a learned model of human preference. For example, we could have human judges rate various outcomes (taking into account learning and satisfaction) and train a model to predict this rating. That model could then be the reward signal for RL. This is an option if manual weighting proves inadequate.

- **Human Override and Iterative Tuning:** We will keep a close human-in-the-loop oversight during any live RL deployment. If the agent does something odd (e.g. consistently choosing an unconventional strategy), we can pause and adjust. The nice aspect is our system can always fall back to a deterministic script if needed (essentially Option A style behavior). So if the RL policy isn't outperforming, we won't force it into production. Only when data shows a clear learning benefit (say, students in the adaptive condition learned 20% faster than those in a non-adaptive control) will we fully deploy it.

8 Fine-Tuning Policy (When and How)

We adopt a conservative, data-driven policy for fine-tuning any models:

- **Don't Fine-Tune Initially:** Our base LLM (the core of answer generation and reasoning) will not be fine-tuned at project start. We rely on prompts and retrieval. The reason is twofold: (1) we lack sufficient high-quality data early on, and (2) we want to preserve the model's broad knowledge and abilities. Fine-tuning too soon on a narrow dataset could cause it to lose general problem-solving capacity or introduce bias.

- **Use Parameter-Efficient Tuning:** When we do fine-tune, we prefer techniques like Low-Rank Adaptation (LoRA) or other adapter-based approaches. These allow us to inject new capabilities or styles without overwriting the base model weights. For instance, we might train a LoRA on the base LLM to adopt a "tutor persona" that always explains step-by-step and uses encouraging language. This way, we keep the original model intact (which means we can still get updates/upgrades to it and simply reapply or retrain our adapters) and reduce the risk of catastrophic forgetting. Research shows LoRA tends to regularize and maintain base performance on other tasks.

- **Targeted Fine-Tuning Modules:** Rather than one big fine-tune, we plan small fine-tunes for specific tasks:

  - *Question Generator:* Once we have a decent corpus of QA pairs or we've curated many example questions for our content, we can fine-tune a model (or a smaller model like T5) to generate questions in our style. This could improve on the prompt-based generation by learning domain phrasing or specific curriculum standards (e.g. always include units in physics questions, etc.).
  - *Solution Explainer:* We could fine-tune another model (or the same with a different prompt) on a set of high-quality explanations or textbook content, so that when the agent explains an answer, it mimics the clarity of an instructor. This might involve training on say all the "worked example" sections of our textbooks.
  - *Distractor Model:* If generating good distractors proves hard with just prompting, we could fine-tune a model to generate them by training on our item bank (we likely have many multiple-choice items from past assessments that we can use as examples of good distractors).
  - *Grading Rubric Model:* Fine-tune a model to rate answers by training on a set of student answers graded by teachers. This will help the AI grader be more consistent and perhaps even provide richer feedback ("this answer is missing a mention of X" if that pattern was in training).

  Each of these is a smaller scope than tuning one giant model to do everything. They can be trained and evaluated independently. We will also consider using smaller models for these if feasible (for faster inference).

- **Criteria to Initiate Fine-Tuning:** We will only fine-tune when: (a) we have gathered sufficient training data of good quality for that task (roughly, a few hundred to a few thousand examples, depending on task complexity), (b) the prompt-based approach is not meeting performance needs or is too costly, and (c) we have a clear way to measure the fine-tuned model's improvement. For example, if our question generation quality is stagnating around

3/5 and we have 500 example questions from experts, that's a cue to try fine-tuning and see if we can push to 4/5.

- **A/B Testing Fine-Tuned vs. Prompt:** Whenever we introduce a fine-tuned component, we will test it against the status quo. It could be offline evaluation (e.g. have SMEs compare outputs blindly) or a live A/B where some users get the new model, others get the original. We only fully switch to the fine-tuned version if it demonstrably performs better or significantly cheaper without quality loss.

- **Maintaining Upgradability:** Suppose our base LLM is an API like OpenAI's or a large model like Llama2. If a new version comes out (with more knowledge or better reasoning), we want to leverage it. Our use of adapters allows for quick porting: we might need to re-fine-tune the adapters on the new base (because weights differ), but since it's cheap to do so (the data is already there and it's a small-scale operation), we can iterate quickly. We avoid any approach that couples us irreversibly to an old model. Moreover, we will keep our prompts and data so that even if we had to start from scratch on a new model, we could reproduce the fine-tunings.

- **Monitoring Fine-Tuned Models:** Fine-tuning can sometimes lead to unexpected behaviors (like a model might become too rigid or style-specific). We will monitor key metrics after deploying any fine-tune: e.g. hallucination rate (did it go up?), user feedback (do they like the new style?), and any error reports. If issues arise, we can roll back to the prior model easily, thanks to modularity.

- **Continuous Learning:** In production, we might gather more training data (especially for grading or question gen via feedback). We'll periodically update fine-tuned models using an incremental learning approach or by retraining adapters with combined old+new data (to avoid forgetting). Each update will be versioned and validated. By doing this, the system's AI components stay current with evolving content and usage patterns.

9 Risks & Mitigations

We enumerate key risks and how we'll mitigate them:

Cold-Start Accuracy Risk: Without prior data, the initial recommendations might miss the mark (irrelevant snippet, wrong difficulty). This could disengage early users. *Mitigation:* Start with a narrow scope/content set and questions we know we can answer (perhaps a specific chapter) to tune the system before expanding. Use expert review for initial outputs: we can run a bunch of likely student questions through the system and have educators verify the answers. Adjust retrieval or add manual mappings for any glaring misses. Additionally, include a user feedback mechanism ("Was this helpful?") from day one, so we can catch and correct bad recommendations quickly.

Hallucination/Misinformation: Despite grounding, the AI might sometimes present info incorrectly, especially in explaining answers. *Mitigation:* Employ a strict retrieve-and-read approach: the answer and explanation will be directly citing the source whenever possible. Use an answer verification step – after generating an answer, cross-check key facts against the content (this can be done by a secondary search or by ensuring the generation includes direct quotes from source). If confidence is low, rather than guessing, the agent will defer ("The material doesn't explicitly cover that; let's look it up together…"). We will log any errors reported and analyze the cause (was it a bad retrieval? LLM misinterpretation?) and fix the pipeline accordingly (e.g. refine prompts or add that content to the knowledge base).

User Frustration (Too Many Questions or Too Hard): The system could overwhelm or demotivate students by asking too many questions or ones that are too difficult. *Mitigation:* Implement bounds on how the pedagogical policy works: for example, at most 1-2 follow-up questions per content snippet unless the student explicitly wants more. If a student keeps failing, the agent should simplify the task – maybe switch to an easier question or provide a direct explanation instead of another question. We'll also allow user agency: the student can say "I want to move on" or "I need more practice", and the agent will respect that to some degree. Frequent user feedback and iterative design with real students will help calibrate this balance.

Privacy Breach: We have sensitive internal content and possibly student data. A bug could accidentally expose content (e.g. showing the wrong snippet from another client's data) or someone could prompt the AI to reveal answers to tests. *Mitigation:* Partition data properly by client and enforce that at the retrieval layer. For example, use separate indexes or namespace filtering so one client's data never gets retrieved for another's query. Implement role-based controls: maybe instructors can get answer keys but students cannot. We also plan a red-team security test focusing on data leakage: e.g. try to get the AI to output full document text by crafty prompts. Ensure the agent only outputs small snippets as intended and never whole documents or any personal data from logs. All data will be encrypted in transit and at rest; compliance standards (like FERPA) will be followed by design (limited retention of student-identifiable performance data, etc., unless needed and approved).

Bias in Content or Model: The AI might reflect biases present in training data (e.g. always using certain names in examples or giving analogies that not all cultures get). *Mitigation:* We will review the output for biased or insensitive content during testing. Also, because our system is grounded in our company's materials (which likely have been reviewed for bias as educational content), the risk is reduced on the answer side. For generated questions/examples, we might fine-tune on a deliberately diverse set of examples. If any bias incidents occur (user reports or we catch it), we'll add that to the fine-tuning data as a negative example (telling the model such content is not acceptable). Periodic bias audits will be done, possibly using external tools or libraries that scan text for gender/racial bias cues.

System Overreach (Acting as Tutor vs. Tool): There's a risk the AI tries to do too much, e.g. outright giving answers when it should be guiding, or conversely being too Socratic when the student just wants a quick answer. *Mitigation:* Clearly define the agent's scope in its prompt/governance: if a direct factual question, just answer with a snippet; if a conceptual understanding question, maybe go into tutoring mode. Also allow the user to control the depth ("give me the answer" vs. "teach me"). Through UX design, set expectations (maybe a toggle for "learning mode" vs "exam mode"). This ensures the AI's behavior aligns with user needs and they don't get annoyed by an overbearing tutor or a not-helpful-enough assistant.

Technical Scalability Issues: As we scale to millions of learners, the infrastructure might strain (vector DB size, LLM costs, latency). *Mitigation:* We plan scalability from the start: using scalable cloud services or clusterable components (e.g. Pinecone or Elastic for vectors, load balancing LLM inference with multiple replicas or using on-demand APIs). We also continuously measure latency and throughput during the pilot; if certain operations are bottlenecks (e.g. ASR for videos), we'll invest in optimization or asynchronous processing. Cost-wise, we monitor cost per query – the RAG approach significantly cuts token usage vs. naive LLM use, and further optimizations (like caching popular questions' answers) can be employed. We will also explore distilling parts of the pipeline to smaller models (e.g. use a smaller model for re-ranking or for simple question generation) to save compute, guided by usage patterns.

Project/Team Risk (Complexity): The architecture is ambitious, with many moving parts (IR, NLP, RL, IRT). Integration bugs or difficulty coordinating team efforts could arise. *Mitigation:*

We keep components decoupled and have clear ownership of each (one group on retrieval, one on pedagogy, etc., with regular syncs). Use API contracts between modules (e.g. the retriever returns JSON with certain fields; the question generator takes input X and output Y) to allow parallel development. If one part lags (say RL), it shouldn't block deployment since we can fallback to a simpler policy. The roadmap is phased to catch integration issues early (e.g. by M3 we already have most pieces connected, so M4–M6 are enhancements). We also allocate some buffer time in later milestones for refactoring or fixing tech debt from earlier quick iterations.

User Adoption Risk: Even if we build it, will students use it effectively? They might treat it as a shortcut (just get answers) rather than a learning tool, or distrust it. *Mitigation:* Product design and messaging are key. We will include guidance within the UI, like "Try answering the question yourself!" or show progress bars of mastery to encourage usage for learning, not cheating. Also, by providing high-quality explanations and making the value clear (e.g. "you improved 10

10 Deliverables per Milestone

Each milestone produces specific outputs and documentation to track progress and facilitate review:

- **M1 Deliverables:** Prototype Q&A system (code and a runnable application, e.g. Jupyter notebook or simple web demo). A brief report with initial retrieval metrics and example Q&A pairs. A log of "red team" attempts (questions we tried to break the system with) and how the system responded, to document its initial robustness. Also, a list of content sources used (so stakeholders know what knowledge it has at this point).

- **M2 Deliverables:** Updated system code with re-ranking and segmentation capabilities. The vector database (or equivalent) loaded with processed content (transcripts, etc.) – essentially our knowledge base at this stage. An evaluation report focusing on improvements: e.g. a before-and-after comparison of search results ranking for test queries. Sample outputs showing JSON structure of answers (to verify the schema for integration). Possibly a short video capture demonstrating a query answered with a video clip.

- **M3 Deliverables:** Pedagogical module code: prompts or fine-tuned models for question generation, and the grading logic. A set of example questions generated from various snippets, with expert annotations or ratings. Internal documentation on how the student model is updated (even if trivial, record the approach). Demo of a full loop interaction (captured as a script or video): user question -¿ snippet -¿ AI asks question -¿ user answer -¿ AI feedback. This will be shown to stakeholders for feedback. Additionally, a revised risk assessment now that we have user interaction (e.g. did we see any problematic content in AI questions?).

- **M4 Deliverables:** The adaptive strategy code (bandit or policy). Simulation results (graphs, explanation of methodology) showing that the learning algorithm behaves as expected. If a shadow mode test was done, the data from that (like policy decisions logged alongside actual outcomes). A document detailing the reward function and parameters chosen, and any constraints coded in. This will serve as reference for future tuning and for reviewers (e.g. educational experts might want to see how we mathematically defined "good outcome"). Also, an updated evaluation report focusing on personalization – e.g. does the system start to differentiate between an "advanced" vs "beginner" user in content selection?

- **M5 Deliverables:** Fine-tuned model files (with versioning info, hyperparams, training dataset description – effectively a Model Card for each fine-tune). Before-and-after examples for outputs that changed due to fine-tuning (to clearly illustrate the differences). Results of any A/B tests or human evals demonstrating the impact. Also, a deployment plan for these models (e.g. if using an open-source model with LoRA, note how we load it in our pipeline). On the analytics

side, by M5 we likely have more data, so a refined learning analytics dashboard might be delivered here, showing per-student or aggregate performance over time (used internally to verify learning gains).

- **M6 Deliverables:** Production-ready system (could be containerized services, with orchestration scripts). All final documentation: System Design Doc, Risk Mitigation Log (updated with any new issues found and how resolved), Data Privacy and Compliance report, and User Guides. Monitoring/Telemetry Dashboard set up (with instructions to access). A final evaluation report summarizing all key metrics achieved vs. targets (this serves as a "proof of efficacy" for stakeholders or potential clients). Additionally, a plan for rollout: which pilot institutions or how we'll increase user base, including support plans (how users can report issues, etc.).

- **First 14 Days Task List (Post-Delivery):** Although not a formal deliverable, we will prepare an operational checklist for the two weeks after initial launch. This includes monitoring responsibilities (who checks the dashboards daily?), issue triage plan (if a critical bug appears, how to patch quickly?), and collection of feedback (schedule meetings with pilot users/instructors after first week). This ensures the project's transition to real-world use is smooth and that we are prepared to respond rapidly.

Throughout, we also ensure each deliverable is reviewed and signed off by relevant experts: e.g. pedagogy experts review M3 outputs, security team signs off at M6, etc. By making deliverables tangible and reviewable, we increase accountability and clarity for all stakeholders. Ultimately, this structured plan and its deliverables aim to prove incrementally that our system is effective at improving learning outcomes in a measurable, scalable way.

Table 1: Comparison of Candidate Architecture Options

Criteria	Option A: RAG + Re-Rank + Agentic Orchestration	Option B: LoRA Fine-Tuning + RAG	Option C: Bandit/RL on RAG	Option D: Fully Fine-Tuned Specialized Models
Components	Pre-trained LLM with retrieval-augmentation; Vector DB + BM25 for hybrid search; cross-encoder re-ranker; Agent orchestrator (tools for search, reasoning).	Same as A, plus custom fine-tuned adapters for pedagogical style (on top of base LLM); RAG for factual grounding remains.	RAG pipeline (as in A) plus a learning algorithm (contextual bandit or RL policy) to select content based on reward signals.	Multiple task-specific models: e.g. one fine-tuned model for question generation, one for grading, possibly smaller LMs each trained on specific tasks; a coordinator to pipe between models.
Infra Complexity	Moderate. Requires setting up a vector database, search index, and orchestration (could use LangChain or custom). Few moving parts; no training cluster needed initially.	Moderate-High. Adds model training pipeline for LoRA fine-tuning. Need to manage versions of fine-tuned models. Serving infra slightly more complex (may host custom model or use API with adapters).	High. Involves running online learning or simulation infrastructure. Need system to log rewards, update policy frequently. More complex testing (stochastic policies).	Very High. Must maintain N different ML models and ensure they interoperate. Pipeline orchestration is complex (passing data between models). Each model training and updating is separate.
Cold-Start Viability	Strong. Uses existing content via retrieval; no historical usage data needed to start. Initial relevance from semantic similarity and manual heuristics.	Moderate. Still can start with RAG for content, but fine-tuned modules (style) need some domain data (e.g. conversations or content summaries) – cold-start for those = using general model (falls back to A).	Moderate. Can start with baseline RAG (Option A) and a simple uniform or heuristic policy. But the adaptive part (bandit/RL) needs interaction data to improve – initial phase will rely on the non-learned strategy.	Poor. Requires training data for each specialized model (questions, grading, etc.) from scratch. Cold-start is lengthy: system can't function well until those models are trained on sizable labeled datasets.
Latency	Moderate. One retrieval + one LLM generation per query. Re-ranking (cross-encoder on top-k) adds some overhead (hundreds of ms) but improves result quality. Overall response in a couple seconds, acceptable for tutoring.	Similar to A per query. Fine-tuned model doesn't necessarily increase per-inference latency (if using same base LLM size). If using on-prem adapter, no external call overhead. The extra training process is offline, not affecting runtime.	Potentially higher. Bandit policy adds slight runtime selection logic (negligible), but exploring multiple content options or multi-step planning could increase average interactions. If implemented in real-time, learning updates could slow response unless done async/offline.	Possibly faster per task since each model can be smaller and optimized for that task. But sequentially chaining multiple models (retrieval → QG → grading) adds up latency. Overall likely slower and less predictable due to pipeline overhead.
Cost	Low-Medium. No large training cost initially. Inference cost dominated by LLM calls; using retrieval keeps context small. Vector DB hosting is moderate cost. Re-ranker cross-encoder adds GPU cost if self-hosted. Scales linearly with queries.	Medium. Additional cost for fine-tuning (compute for training runs, which is modest with LoRA on small data). Inference cost similar to A; hosting custom model weights might incur some overhead but minor.	Medium-High. If using bandits, training is online but lightweight per interaction. However, evaluating policies may require experimentation. Possibly need simulation or offline analysis (compute cost). Long-term, improved efficiency might reduce content length and thus LLM token costs (savings).	High. Significant upfront cost to train specialized models (data collection, labeling, compute). Maintaining multiple models = multiple serving endpoints (costly if each is large). If smaller models are used, training cost might be less than one giant model, but overall development cost is high.
Data Needs	Low to start. Does not require labeled pairs for content selection	Moderate. Needs some exemplars to fine-tune pedagogical style or do-	High. Needs a stream of interaction data (questions asked, content	Very High. Each model needs its own dataset (question generation re-