Adaptive Learning System Design Roadmap

Table of Contebts

1. Executive Summary

- **Hybrid RAG + Adaptive Strategy:** We recommend a Retrieval-Augmented Generation (RAG) architecture enhanced with adaptive learning policies. This approach grounds answers in trusted content to minimize hallucinations[1] while dynamically personalizing resource selection for each learner via multi-armed bandits. It balances factual accuracy and personalization better than end-to-end trained models.
- **Stepwise Integration of Techniques:** We will start with a RAG baseline (retrieving concise video/PDF segments) and incrementally layer on improvements: cross-encoder reranking for relevance[2], micro-content segmentation for brevity[3], pedagogical tools (question generation, automated grading), and a lightweight reinforcement learning loop for content selection. This phased rollout ensures measurable gains at each milestone.
- **Cost-Effective Customization:** By using parameter-efficient fine-tuning (LoRA) on specific pedagogical tasks, we achieve a tailored teaching style without expensive full-model training. LoRA freezes the base model weights and trains only small adapter matrices, drastically reducing training cost (e.g. fine-tuning a 11B model with LoRA can train <0.2% of parameters[4]). This yields comparable performance to full fine-tuning at a fraction of the compute cost[5].
- **Data-Driven Personalization from Day 1:** Even with no historical content recommendation data, the system can leverage metadata and semantic overlap to suggest relevant micro-content (short video clips 2–5 minutes[6], PDF sections

1–3 pages). As users interact, we rapidly accumulate feedback to train better ranking models and a bandit policy focused on learning gains, not just clicks.

- **Focus on Learning Outcomes:** Unlike typical recommenders that maximize engagement, our reward function explicitly prioritizes learning gains (e.g. improved post-quiz scores or ability estimates) while penalizing unnecessary content length and inaccuracies. This multi-objective reward steers the system toward efficient, mastery-driven suggestions[7].
- **Measurable Progress at Each Milestone:** We define clear metrics for success at each phase – from IR retrieval quality (nDCG, Recall) and "overkill rate" of content length, to pedagogy quality (question clarity via expert rubrics), to learner growth (IRT ability θ improvements and normalized learning gains[8]). Each 2-week sprint delivers a shippable increment with evaluation reports to demonstrate improvements in *learning efficacy*, not just UX polish.
- **Reduced Risk via Modularity:** The chosen stack is modular and model-agnostic. Retrieval can plug in new embedding models or data sources without rearchitecting. Pedagogical modules (e.g. question generator) are separate from the core LLM, allowing updates or fine-tuning on new data without affecting the whole system. This de-risks the project against rapid shifts in foundation model technology by keeping us agile (adapters over full model retrains).
- **Agentic Orchestration for Adaptivity:** An AI "planner" agent coordinates the workflow: it retrieves content, poses questions, evaluates answers, and decides the next step (more practice, hint, or move on). This closed-loop design ensures the system responds to learner performance in real time, akin to a personal tutor. Over time, the orchestration policy will be refined (via bandit algorithms) to present the right content at the right time for each student, maximizing long-term retention[9].
- **Why It Beats Alternatives:** Our approach leverages existing resources and large pre-trained models for a strong cold-start, then steadily learns from data to surpass static strategies. Pure end-to-end solutions would require prohibitively large training data and risk student safety, while a manual or rules-based system wouldn't scale or adapt. By combining RAG for fidelity[1], LoRA fine-tuning for pedagogical nuance, and bandits for adaptive sequencing, we get the best of all worlds – a scalable system that is **grounded, concise, adaptive, and laser-focused on learning outcomes**.

2. Architecture Options Comparison

We considered four main architecture paradigms (A–D), evaluated across factors like implementation cost, runtime latency, data requirements, cold-start viability, expected learning impact, and team fit. The table below summarizes the trade-offs:

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
A) RAG-first + ReRank + Agent Orchestration *Baseline Retrieval-Augmented Generation pipeline with an agent coordinating steps*	- Bi-encoder vector search over content (embeddings index) - BM25 + dense *hybrid* retrieval for coverage[10] - Cross-encoder reranker for top-k results[2] - Agent orchestrator (retriever → answer composer) to ensure	**Low initial cost:** Uses pre-trained LLM and existing content. Build vector index, no custom model training initially. Standard cloud VMs for embedding & LLM calls; no special GPUs needed unless heavy traffic.	**Moderate:** Retrieval (~100 ms per query), rerank (~10–50ms for cross-encoder on top-10), plus LLM generation (~seconds depending on model). End-to-end typically a few seconds per question.	**Cold-start ready:** Requires *no* historical training data for recommendations – uses semantic similarity and keywords to find relevant snippets. Content corpus (videos, PDFs) and embedding model needed upfront.	**Good baseline learning:** Delivers relevant, *grounded* answers to student queries, reducing misinformation. Learning impact initially comes from giving correct info quickly. Less personalization in this	- **Knowledge gaps:** Depends on existing content; if knowledge base lacks an answer, the LLM may still falter or omit (mitigated by saying "I don't know"). - **Not personalized:** Doesn't adapt to student's prior knowledge or	**Strong:** Team has NLP and "agentic AI" skills. Setting up retrieval and prompt orchestration is straightforward. Leverages LLM expertise without requiring new model training.

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
	grounded answers			 **Data needs:** minimal for start; improves with feedback but not required to function.	option – all users get similar content for a given query.	difficulty needs. - **Maintenance:** Content updates must be re-indexed, but this is manageable.	
B) Lightweight Fine-Tuning (LoRA/ PEFT) + RAG *Train small adapters for pedagogical style or domain, layered on RAG*	- RAG pipeline as above for factual grounding. - Add LoRA adapters on the LLM for specific abilities: e.g. more explan	**Moderate cost:** Requires training small adapter (~<1% of model parameters[4]). One-time fine-tuning per task (questi	**Same or Slightly higher:** Inference uses base LLM + adapter merge; negligible overhead (just adds a few matrix ops). No	**Cold-start:** Adapters need examples. If no real data, can use *few-shot prompt* or synthetic data to do a "proto-fine-tune." Cold-st	**Higher learning relevance:** Fine-tuning can enforce pedagogical best-practices (e.g. always include a hint or ensure explanations	- **Data dependency:** Requires quality training data for the specific style/skill (which we may need to create). - **Risk of minor**	**Good:** Team can handle fine-tuning on domain data. PEFT techniques are well within data scientists' expertise. Leverages

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
pipeline	atory tone, Socratic questioning style, or domain-specific vocabulary. - Possibly fine-tuned modules for distractor generation or hinting.	on style, etc.) on either domain data or synthetic examples. Compute: a single GPU can fine-tune a LoRA on a 7B–13B model in a few hours.	extra network calls. Latency remains dominated by LLM runtime (seconds per answer).	art viability is okay if using general model behavior initially, then fine-tune when data available. **Data needs:** Low/medium – on the order of hundreds of example Q&A or conversations in desired style. Can incorporate	are at the right level). This yields more *consistent, student-friendly outputs* than prompt-tuning alone. Likely boosts student engagement and comprehension (e.g. clearer explanations, better-aligned questions).	**drift:** Poor fine-tuning could degrade base model correctness (mitigated by LoRA freezing most weights)[11]. - **Maintenance:** If base model is updated, adapters might need retraining or adjustment for compatibility.	our strength in model fine-tuning while keeping costs manageable.

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
C) RL/Bandits for Content Selection on RAG baseline *Contextual bandit policy learns which content piece (or strategy) yields best learning outcome, on top of retrieval results*	- Use RAG retrieval as default candidate generator. - Train a bandit or RL policy to choose the *best* content snippet or next action for a student given context (user history, question difficulty, etc.)[12].	**Upfront cost:** Higher complexity. Need to log interactions, maintain a policy model. Infrastructure for online learning (continuous training) or at least periodic policy updates. **Compute:** Bandit algorithms	**Low runtime overhead:** Policy lookup is fast (milliseconds to select an action). The main latency remains the content retrieval and LLM. **Potential delay:** Slightly longer decision-making if exploring (e.g.	**Cold-start:** *Challenging.* Needs some initial policy – likely we'd start with heuristic (e.g. always pick highest-ranked minimal resource) and a small exploration rate. As data accrues, the bandit learns. **D**	**High adaptive impact:** Over time, the system will *learn which content leads to the greatest learning gains*. E.g., it might learn that for concept X, a 3-minute video yields higher quiz scores than a text snippet	- **Complexity:** Requires careful reward design (to truly reflect learning, not just short-term answers). Mis-specified rewards could cause the policy to exploit the wrong behavior (e.g. always show easiest conten	**Strong:** The team's reinforcement learning experts can design and tune the bandit. They have experience with contextual bandit algorithms and can manage the engineering of an online learning loop.

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
	 - Reward defined by student's improvement (quiz correctness, time-on-task) instead of clicks. - Could start with epsilon-greedy or LinUCB bandit, later a policy-gradient RL as data grows.	are lightweight (linear models), but full RL (if used) might require GPU for training simulations.	computing scores for multiple options). Negligible for user.	**data needs:** Continuous interaction data (which we will get as students use the system). Off-policy learning possible from historical Q&A logs (if any reward proxies exist there). Initially, can simulate or use expert judgments as reward	, adjusting recommendations accordingly. Prior studies show bandit policies can boost educational outcomes by personalizing content sequencing[9][13]. Bandits also enable exploration of new content, finding better resources	t to get quick correct answers – we will mitigate by rewarding mastery gains, not raw score alone). - **Data hunger:** Until sufficient student interactions are collected, policy may be suboptimal or unstable. Cold-start	This fits our ambition to push an *adaptive* AI tutor.

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
				signals.	than a static rank might choose.	period must be managed so students aren't harmed by random exploration (use safe initial policy + high exploration decay). - **Evaluation:** Off-policy evaluation is non-trivial; we'll use techniques like IPS to validate the	

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
						policy offline[14]. Safety and fairness of learned policy must be monitored (ensuring it doesn't, for example, favor certain groups).	
D) Fully Fine-Tuned Task-Specific Models (small LMs) *Replace large LLM + prompts with*	- e.g. Use a Encoder-decoder model (like T5 or smaller GPT) fine-tuned for question genera	**High upfront cost:** Each model needs training data and pipeline integration. Fine-tuning	**Potentially faster inference:** Each small model is optimized for its task and could be	**Cold-start:** *Poor.* This approach requires substantial labeled data to train each compo	**Moderate impact (if achieved):** Specialized models can outperform prompts in narrow	- **Long development cycle:** Building each component from scratch increases integration points	**Feasible but heavy:** The team has the skills to train models, but doing many at once

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
an ensemble of smaller models each fine-tuned for a specific subtask (retrieval, question generation, grading, etc.)	tion on our content. - A BERT or RoBERTa fine-tuned for short answer grading against rubrics [15]. - Perhaps a smaller transformer for difficulty estimation or hint generation. - Essentially, a pipeline where each	small models is cheaper than a giant model, but doing many of them (QG, grading, etc.) adds up. Infrastructure becomes a collection of services (one per model) – more complex to maintain. In deployment, smaller models can run on	very fast (milliseconds) compared to a big LLM taking seconds. Pipeline parallelism could further speed it up (e.g. generating question and evaluating previous answer concurrently). However, pipeline overhead (data	nent before it's useful. We lack these at project start (no "correct question → best video segment" training pairs, etc.). Would need a major data labeling effort or rely on synthetic data to train models. **Data needs:**	tasks once fully trained. For example, a fine-tuned question generator might consistently produce well-structured, curriculum-aligned questions, and a grading model can reliably score answers like a human. This could improv	(potential bugs) and slows iteration. Any change in one task (e.g. new question style) requires collecting new data and retraining that model. - **Maintenance burden:** Multiple models to update as content or objectives	spreads our effort thin. It diverts focus from leveraging powerful existing LLMs. Given our size, a fully bespoke model stack is risky. We can pilot one small model (e.g. a BERT grader) later on, but not make the entire

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
	stage is a custom model trained on labeled data for that task (instead of prompting a big general model).	CPU or single GPU, which might reduce per-request cost if optimized. But the cost lies in development and maintenance of multiple models.	passing between models) and serial stages might negate some gains. Worst-case latency is sum of all stages. Still, likely slightly better than using one large model for everything.	High – for each subtask, hundreds or thousands of examples. E.g., to train a question generator to cover Bloom's levels, we'd need a dataset of content passages with questions at various cognitive levels[16]. For grading,	e the consistency of the system's teaching and assessment. However, the *ceiling* of each small model may be lower than a big LLM, especially for creative tasks. So while reliability goes up, the richness or adaptivity might	change. Ensuring they all remain compatible (e.g. question generator's style matches the grader's expectations) is challenging. - **Quality risk:** Smaller models may not capture the full nuance. E.g., a 110M param	system from only custom models initially.

Option	Components & Approach	Cost & Infra	Latency	Data Needs & Cold-Start	Expected Learning Impact	Risks/ Cons	Team Fit
				need student answers with scores. These are non-trivial to obtain.	be limited to what the model was trained on.	eter model fine-tuned for grading might not understand complex reasoning as well as GPT-4 with a prompt. Could lead to lower-quality feedback unless data is extremely good.	

Recommendation: We choose a **hybrid of A, B, and C**, leveraging RAG as the backbone, adding fine-tuned pedagogical adapters as needed (B), and introducing a bandit-based adaptive scheduler (C) once a baseline is in place. This hybrid maximizes immediate usefulness and accuracy (thanks to RAG grounding[1]) and then continuously learns to optimize for learning outcomes[12]. Option D (fully task-specific models) is not chosen as our primary route due to its high data and maintenance demands, but we may incorporate small specialized models in supporting roles if beneficial (e.g., a lightweight formula extractor or a math solver module) down the line.

3. Final Recommended Architecture

Architecture Overview: The system will be composed of modular layers, each responsible for a part of the learning loop, orchestrated by an agent controller. Below is a breakdown of the components (see Figure 1 for a schematic diagram):

Retrieval Layer (Knowledge Base Indexing & Search): We maintain an indexed repository of all learning content (video transcripts, textbook PDFs, lecture notes, etc.). Each document is chunked into small, semantically coherent pieces (e.g. ~300 tokens with overlap) to ensure relevant matches[17]. We use a **hybrid search** strategy: combine lexical search (BM25) with semantic vector search. This ensures that if the student's question uses different wording from the content, we still find relevant passages via embeddings, while exact keywords also get high priority[10]. A bi-encoder embedding model (e.g. Instructor-XL or OpenAI text-embedding-ada) converts text into vectors; we may use domain-specific embeddings for math or code if needed. Metadata filters (content type, length, difficulty tags) pre-filter the corpus (for instance, exclude full 1-hour videos). The top-k retrieved chunks are then re-ranked by a **cross-encoder** that scores each (question, chunk) pair in context[2], providing a more precise relevance ordering. We also apply Maximal Marginal Relevance (MMR) to diversify results[18] – this avoids redundant snippets and ensures we cover different aspects if the query is multi-faceted. The output of this layer is a set of candidate *answer resources* – e.g. three short text excerpts or video segments likely to contain the answer.

Content Minimization & Snippet Extraction: To enforce the micro-learning constraint, a dedicated sub-module trims content to the minimal necessary segment. For **videos**, we utilize Automatic Speech Recognition (ASR) to get transcripts and then perform *topic segmentation* on the transcript[19][20]. Using techniques like semantic similarity or slide change detection, we identify chapter or scene boundaries. Rather than ever recommending a full video, the system selects a segment (e.g. timestamp 2:10–4:00 of a video) that has high semantic overlap with the query. If a relevant segment is still long (say 8 minutes), we apply summarization to compress it further or find a sub-clip containing the answer. For **PDFs/text**, we chunk by headings and paragraphs, and if a chunk is too large (e.g. a whole chapter), we narrow to the subsection or even generate an extract. We define a *"sufficiency score"* for a candidate snippet: a measure of semantic coverage of the learner's question versus the snippet's length. The system favors the snippet with the highest coverage per unit length (essentially maximizing information density). Hard constraints will be set (e.g. discard any snippet >5 minutes or >3 pages to uphold brevity[3]). This component may also highlight the exact part of the text that answers the question (to enable quick skimming). By the end of this stage, the system has one or a few **"micro-nuggets"** of learning content – self-contained, concise resources likely to answer the question.

Pedagogical Layer (Adaptive Questioning & Guidance): This layer handles all learner interactions after delivering the initial resource. It includes:

- **Automated Question Generation:** Based on the content the student just consumed, the system generates a formative question to test their understanding. We use the LLM (or a fine-tuned smaller model) to create questions aligned with various *cognitive levels of Bloom's taxonomy*[21]. For example, if the student's query was factual, we might generate an application or analysis question to deepen understanding. The prompt to the LLM includes the content snippet and an instruction like "Generate a question that requires the student to apply this knowledge." Few-shot examples (or a LoRA fine-tune) ensure the style and difficulty are appropriate. Research indicates LLMs can produce high-quality educational questions at different Bloom levels with proper prompting[16], so we'll leverage that. Questions may be open-ended or multiple-choice depending on context. If multiple-choice, a **distractor generator** will create plausible incorrect options (drawing on common misconceptions or related concepts, possibly using a smaller model fine-tuned for distractor generation).

- **Hint and Explanation Generation:** In parallel, the system prepares hints and worked solutions. Using the content and the generated question, the LLM can derive the correct answer and a step-by-step explanation. This uses a *self-consistency* approach: have the LLM generate multiple reasoning paths/answers and verify agreement[22], to reduce the chance of error in the provided solution. The first hint might be subtle (pointing the student to revisit a specific part of the snippet), and if the student is still stuck, more direct hints or even a fully worked example can be revealed. The style is encouraging and Socratic where possible, per pedagogical best practices (we can fine-tune the LLM on a few example tutor-student dialogues to nail this tone).

- **Rubric-based Grading:** When the student submits an answer, the system evaluates it. For objective questions this is straightforward (match the correct choice). For open-ended responses, we employ a **rubric grader**. This could be an LLM prompt that checks the student's answer against the solution and a rubric of key points, or a smaller model (like a fine-tuned RoBERTa) that outputs a score. The rubric is derived from the content's learning objectives: e.g., "Answer should mention concept A and B and explain relationship C." The grader model uses this rubric to assign full, partial, or no credit. Automated short answer grading with transformers has shown effectiveness in assisting educators[15], and we will incorporate those techniques – e.g., using SBERT to compare student answer with ideal answer[15]. The grader also identifies specific missing pieces in the student's answer to inform feedback.

- **Feedback Generation:** Based on the grading, the system provides feedback. If the answer is incorrect or partially correct, the feedback module (likely the LLM again) uses the rubric to generate an explanation of why the answer was wrong and what the correct reasoning is. It may also point back to the resource snippet ("Re-watch 2:30–3:00 of the video where this concept is explained.") ensuring the

feedback is constructive and rooted in the content (grounded feedback reduces hallucinations in explanations[1]).

This pedagogical layer ensures the system doesn't just answer the student's question, but actively engages them in learning through questioning, hints, and explanations, mimicking a human tutor. All of these capabilities (QG, hints, grading) can be improved over time via fine-tuning. Initially we rely on prompting GPT-4 (for example) with careful instructions and examples; by Milestone 5 we may introduce LoRA-tuned models for these tasks to standardize output style and quality.

Assessment & Learning Analytics Layer: This is the brain that tracks learning over time. It maintains a model of the student's knowledge state and updates it after each interaction. We will use an Item Response Theory (IRT) approach to estimate the student's *ability θ* and each question's parameters. Concretely, each question (or formative assessment item) will have a difficulty parameter *b* (how hard it is) and discrimination *a* (how well it differentiates high vs low ability students) in a 2- or 3-parameter logistic model[23]. The student's ability θ is updated using their answer (e.g., via Bayesian update or maximum likelihood on the IRT model). Correct answers raise θ (more if the question was high difficulty), wrong answers may lower θ. Over time, the standard error on θ decreases as we collect more data, giving us confidence in their proficiency. We also calibrate question difficulty on the fly: if many students consistently get a question wrong, the system may adjust that item's *b* parameter upward to reflect it's harder than initially thought. This layer also computes aggregate *learning gains* for the student: for example, the change in their θ between the start and after several sessions, or normalized gains if a pre-test was given[8]. We break this down by topic if tagging is available (so we know, say, the student has mastered algebra basics but still struggles in word problems). The analytics layer thus provides a continuously updated portrait of learning. It also informs the planning layer – e.g., if a student's estimated ability in a topic is low, the system might inject a foundational content piece or an easier question next.

Additionally, this layer monitors **item quality metrics**: discrimination (a) of each question (does it get answered correctly much more by high-ability students than low-ability? If an item has a very low discrimination, it might be a bad question or ambiguous), and guessing factor (c) for multiple-choice (if even low-ability students get it right at a high rate, perhaps they can guess it easily – item may need revision)[24][23]. These statistics guide content improvements and ensure our assessments are effective.

Agentic Orchestration (Controller): At the top level, an **agent orchestrator** oversees the entire workflow per student query. This can be thought of as a policy engine or planner that decides which step to execute next, conditioned on the context. For instance, when a learner asks a question: 1. The agent triggers the *Retrieval Layer* to fetch a relevant snippet. 2. It then calls the *Pedagogical Layer* to pose a follow-up question. 3. Depending on the student's answer (captured by the *Grading* module), the agent decides: did learning occur? If the student got it right with high confidence, maybe move to a more advanced question or conclude the session. If the student is wrong, maybe

the agent decides to give a hint and ask a simpler question next (remediating a gap). 4. The agent also interacts with the *Assessment Layer*: it queries the student model to decide difficulty of next question ("student's θ is low, let's not jump too high in Bloom's taxonomy yet"). 5. It may loop back: e.g., if the student still fails after hints, the agent might select an *alternative resource* (perhaps the second-best snippet retrieved, or a different modality) and then continue questioning.

This orchestration can be implemented as a **finite state machine** or via an LLM agent that is given a schema (allowed tools: {Retrieve, AskQuestion, GiveHint, EndSession}, and it uses natural language reasoning to choose actions). Given our expertise, a structured approach (like a Python policy class or a small planning model) will ensure reliability. The orchestration ensures **fallbacks** are in place: e.g., if retrieval finds nothing with high confidence (low similarity), the agent will not hallucinate an answer – it will instead respond with a clarification question or a polite deferral ("I'm not sure I have the best explanation for that. Let me get an instructor to help."), which is a safety guardrail.

Every step the agent takes and the outcomes (right/wrong answers, etc.) are logged, feeding back into the learning analytics and (eventually) the reinforcement learning algorithm that will refine the policy of content selection.

Diagram: *The architecture can be visualized as follows:*

```
[User Query]
   ↓
[Orchestrator Agent] --(consult student model)--> Decide action
   ↓ (first action: retrieve content)
[Retrieval Layer: BM25 + Embeddings Search]
   ↓
Cross-Encoder Re-Ranker + MMR (diversify)
   ↓
Filtered Top Result (short snippet or clipped video)
   ↓
[Content Presented to Learner]
   ↓
[Pedagogical Layer] generates formative question about the content
   ↓
Student answers question
   ↓
Grading module (rubric/LLM) evaluates answer
   ↓
[Assessment Layer] updates mastery (IRT model)
   ↓
[Orchestrator Agent] decides next step (e.g. more hints/questions or end if
mastered)
   ↲ (feedback loop to Pedagogy or Retrieval as needed)
```

The layers communicate via the orchestrator, which carries the context (the specific question asked, the content shown, the student's performance so far, etc.). This modular setup allows us to improve each part independently (e.g., swap in a better

retriever or a more advanced student model down the road) without altering the whole system.

Crucially, the recommended architecture is **grounded in real data** at every turn – content comes from a vetted library and answers are verified against that content[1] – and **adaptive to the learner** – questions and next actions are chosen based on demonstrated knowledge, using bandit policies and IRT estimations to maximize learning gains.

4. Data Plan: From Cold-Start to Flywheel

Launching an adaptive learning system without historical recommendation data requires bootstrapping with intelligent heuristics and rapidly leveraging any user interaction to improve. Our data plan addresses the cold-start in content recommendation and then outlines how we evolve into a virtuous flywheel of data collection and refinement.

Cold-Start Strategies for Content Recommendation (Zero/Low-Shot):

- **Heuristic Initial Ranking:** At day 0, with no usage data, the system will rely on content metadata and semantic similarity to select resources. For example, if a learner asks *"What is Newton's second law?"*, we filter for content tagged "Newton's Laws" or with those keywords in titles. Among those, prefer the shortest video or document section that matches (to avoid overkill). We use ASR transcripts to ensure even spoken content is searchable. The idea is to apply human-like heuristics: prefer content labeled as introductory if question seems basic, or advanced if question contains advanced terms, etc. This can be encoded as metadata ranking rules until the learning-to-rank model can be trained.

- **Semantic Coverage via ASR+Text:** For video content, our system can on-the-fly transcribe and index speech (if not pre-indexed). We then do a semantic similarity between the question and different time segments of the transcript. The segment with the highest similarity score is a strong candidate snippet (this is essentially a weakly supervised alignment of question to content). This way, even without any prior pairing data, we exploit the content's text to locate answers.

- **Weak Labels from Overlap:** We will generate **pseudo-labels** for training a reranker by using overlap of content and Q&A. For instance, we have historical Q&A data (without content). If some question's answer text is known, we can search our content corpus for that answer text. If a particular document contains a sentence very similar to the known answer, we label that document as relevant to the question. This gives us (question → relevant content) pairs cheaply. They are noisy, but as a starting training set for a learning-to-rank model, they provide signal.

- **Teacher-in-the-Loop Cold Curation:** For the most common 50–100 questions (we can guess these from search query logs or known FAQs), we will ask subject matter experts (or experienced teachers) to manually pick the "best minimal

resource" that answers each. This small curated set serves multiple purposes: it seeds the initial recommendation logic (we can hard-code these as overrides), and it provides high-quality training examples for our models (especially to fine-tune the cross-encoder reranker or to evaluate our system against human judgment). We will supply the teachers with a rubric emphasizing *minimality and sufficiency* ("Choose the shortest piece of content that fully answers the question" and "if none exists, break a longer content into a segment"). Inter-rater agreement will be measured to ensure consistency. These human labels are gold-standard data that we can upweight in training.

- **Metadata and Taxonomy Filters:** If our content is organized by a taxonomy (say, by subject → topic → subtopic), we will use the classification of the question (via an NLP classifier or keyword mapping) to restrict the search space. E.g., detect that the question is about *Physics → Mechanics*, then only search within that content subset. This improves precision dramatically in cold-start when we can't rely on collaborative filtering. As we get more data, we can relax this if needed or let the learned model figure it out.

Rapid Dataset Creation & Active Learning:

Once the system is running, every interaction is data. We implement a data pipeline to capture: the question asked, the resource shown, whether the student clicked it or how long they spent (dwell time), whether they got the follow-up question right, etc. These will be logged (with user privacy in mind; identifying info is hashed or not stored). With these logs:

- **Continuous Label Generation:** Each time a student rates a resource as helpful or answers a quiz question after a resource, we treat that as a label of that resource's usefulness for that query. For example, if the student quickly and correctly answers the follow-up after watching the video segment, that's a positive label for (query, segment). If they seem confused or rate it poorly, that might be a negative label. Over a few weeks, we will accumulate a non-trivial labeled set of query → content → outcome. This can directly feed training of a learning-to-rank model to replace initial heuristics. Essentially, we transition from heuristic to data-driven ranking by retraining the cross-encoder on real feedback data.

- **Offline Labeling Protocols:** In parallel, we design an annotation task to enlarge our training data. We might use crowdworkers or junior teachers to label whether a given content piece answers a question. But to make this efficient, we can use *active learning*: run our retrieval on a large set of sample questions, gather the top 3 results, and have annotators label those results as "directly answers the question", "partially answers", or "not relevant". We specifically focus on cases our current model is unsure about (e.g., the cross-encoder scores are borderline) to get the most informative labels[14]. This process will quickly highlight any systematic mismatches and give us data to train the reranker to fix them.

- **“Best Minimal Resource” Rubric:** For consistency, any human labeling follows a rubric: the best resource is one that (a) contains the answer or enables the student to derive the answer, (b) has no extraneous information beyond what’s needed, and (c) is as concise as possible while still clear. We’ll include examples in the guideline (like a 2-page excerpt that answers the question is better than a full 10-page chapter even if both contain the answer). By explicitly defining “overkill rate” (e.g., mark if a suggestion is over, say, 3× longer than needed), we generate data for our minimality metrics and training a model to detect overly long content. Labelers will also flag if content is *technically correct but pedagogically inappropriate* (too advanced or assumes knowledge the learner might not have) – these notes help us add metadata (like reading level) to content and incorporate that into the recommendation logic.

- **Dataset for Question Generation & Grading:** We will mine the existing Q&A/assessment logs to create datasets to refine the pedagogical modules. If the logs contain student answers and scores from past homework or quizzes, we have a ready-made grading dataset. We’ll fine-tune a grading model (or at least validate the LLM’s grading against this data). For question generation, we can use any available set of teacher-created questions in our domain: fine-tuning a model like T5 on those can give it more grounding in domain terminology and typical phrasing. Even a small set of 500 curated Q&A pairs can significantly steer the style of the questions the LLM generates (making them more exam-like or more clear, depending on what we want).

- **Bootstrapping Difficulty Calibration:** If historical logs have per-question success rates, we’ll use those to initialize IRT difficulty values. E.g., if question Q was answered correctly by 80% of past students, it’s easy (low b); if only 30% got it right, it’s hard (high b). This saves time in cold-start for our mastery model to know rough difficulties. As new questions are added by the system, we’ll start all at medium difficulty and adjust based on early usage data.

- **Flywheel of Improvement:** As more students use the system, our **bandit exploration** will occasionally try alternative content (to learn if perhaps a different snippet works better). These explorations, combined with outcome tracking, feed back to improve the recommendation policy. The system effectively becomes self-improving: better recommendations → better learning → more engagement and trust → more data → better models. Our role is to carefully monitor and intervene (via model updates or new data collection) in the early phases to ensure the flywheel spins in the right direction (toward learning, not just clicks).

In summary, the data plan starts with intelligent bootstrapping using domain knowledge and weak supervision, then quickly ramps up to **learning from real interactions**. By Milestone 2–3, we expect to have enough data to retrain the retrieval ranking model and to begin policy learning for personalization. By Milestone 4+, the system should be generating substantial proprietary datasets: e.g., a growing pool of Q&A pairs it handled, which can be reused to train or fine-tune our models (this proprietary data

becomes a competitive moat and allows us to gradually reduce reliance on generic base models, if desired, by fine-tuning them more to our domain).

5. Metrics & Evaluation Plan

To ensure each milestone delivers "better learning outcomes," we define a comprehensive set of metrics. These cover retrieval performance, content minimality, pedagogical quality, assessment reliability, learning gains, and safety. We will evaluate these in offline tests and, where possible, with live A/B experiments. Below are the key metrics and how we will measure them (with target improvements per milestone):

Retrieval & Recommendation Metrics:

- **Recall@K and nDCG@K (Normalized Discounted Cumulative Gain):** Measures the quality of our content retrieval. We'll maintain a test set of queries with known relevant resources (from our labeled data or known good answers). *Recall@3* or @5 should be high – i.e., the correct answer snippet is in the top results X% of the time. *nDCG@3* weighs the ranking of relevant items; we want to see nDCG improve as we add reranking. For example, after adding the cross-encoder, we expect nDCG@3 to increase significantly (as seen in similar RAG pipelines[2]). We set an initial acceptance criterion (M1) like "Recall@3 >= 0.8 on the test set" and tighten this over time.

- **Coverage of Questions:** The percentage of user queries for which the system found at least one potentially relevant resource. If coverage is low, users get "no answer" responses – we track this to ensure our content base and retrieval are adequate. By milestone M2, coverage should be >95% for in-scope question types (assuming our content library covers the curriculum).

- **Time to First Useful Resource:** How long (in seconds) from user question to the system displaying a useful snippet. This includes system latency and any extra steps. We aim for, say, <3 seconds average. More interestingly, if the user looks at multiple suggested resources, we measure the rank of the one they actually found useful. We want that to be ideally 1 – meaning the first suggestion was useful. So *time-to-first-useful* is effectively a user behavioral metric: if the user had to try three suggestions, that's slower. Our goal is to reduce this by improving ranking. Success is, e.g., 90% of the time the first suggestion suffices by M3.

Content Minimality Metrics:

- **Median Resource Length:** We will track the median duration (for videos) or length in pages/words (for text) of the resources provided. We expect this median to drop as we implement segmentation and summarization. For instance, baseline might be median video length 8 minutes (if initially we sometimes return a full video); after M2's segmentation, median might drop to ~3 minutes[6]. We'll set a target like "Median content length <= 3.0 minutes by M2" to align with microlearning best practices. A similar target for text: median < 2 pages.

- **Overkill Rate:** The percentage of recommendations that exceed the target length (say >5 min or >3 pages). We want this near zero. If at M0 the overkill rate is 30% (just hypothetical if initial system gives some long items), we aim to cut this to <10% by M2, and <5% by M3. Overkill instances will be analyzed qualitatively to see why they happened (did we lack a shorter content? Did our reranker mistakenly prefer a longer item that had slightly higher relevance score?). This metric directly measures how well we're enforcing the "minimal" in minimal resource.

- **Compression Ratio:** For any content longer than the target that we do show (maybe unavoidable in some cases), we measure how much we summarize or skip. E.g., if we show a 5-minute excerpt from a 60-minute video, that's a compression ratio of 12:1 from the whole. Or if we summarize a 10-page chapter into a 2-page summary, that's 5:1. Tracking this helps quantify how much we're condensing information. A higher average compression (within reason) indicates we're delivering more succinct content. We'll use this internally to ensure our summarization component is pulling its weight.

Question & Pedagogy Quality Metrics:

- **Expert Rubric Score for Generated Questions:** We will have education experts review samples of the AI-generated questions for clarity, appropriateness, and alignment with the learning objective. They'll score each on a rubric (e.g., 1–5 scale for clarity, cognitive level appropriateness, lack of ambiguity). Our baseline (LLM with no fine-tune) might score, say, an average 3.5. We expect after adding few-shot exemplars or tuning (M3) to see this average score rise (target e.g. >4.0). We'll specifically check that we are getting a distribution of Bloom levels – not all questions should be low-level recall[21]. If initially 90% of questions are remember/understand, by M3 we want a more balanced mix (perhaps 50% lower-order, 50% higher-order) to ensure depth.

- **Answer Accuracy (Pass@1 or @k for open-ended):** For questions with known correct answers, we can measure how often the student (or our test harness) could answer correctly after using our resource. We simulate this by having a solved answer and seeing if our provided content plus question leads to that answer (for example, we might use the system on some known QA pairs to see if it guides to the right answer). This is tricky to automate, but for multiple-choice we can check if the correct answer is achievable from the content. Essentially, we want to ensure our *questions are answerable from the provided content* – a factuality check. A question failing this (no info to answer it in snippet) is a flaw. We target 100% answerability (0 hallucinated or ungrounded questions). We'll use a *reference-grounded check*: compare the question and reference content via an LLM to see if the answer is contained[25]. Any flagged by this as lacking support will be counted as a defect.

- **Distractor Quality:** For multiple-choice, good distractors should be tempting yet clearly wrong if you know the material. To evaluate, we'll measure how often students choose incorrect options *when they actually didn't know the answer* versus when they did (discriminatory power of distractors). This may require live testing. Alternatively, we have experts rate distractors on plausibility. If early on some distractors are obviously irrelevant (e.g., a physics question with a distractor from chemistry), that's a quality issue. We aim to minimize those. A metric could be "<5% of distractors are rated as implausible or silly by experts".

- **Student Engagement with Questions:** Not exactly a direct learning metric, but we track if students attempt the questions or skip them. High attempt rate and reasonable time spent suggests the questions are engaging and clear. If we see skips or very quick random answers, it might indicate poor question quality (or too hard). We set a goal that >85% of questions posed are answered by students (i.e., they don't give up).

Assessment Quality Metrics:

- **IRT Item Parameters Stability:** As we collect response data for items, we can estimate their discrimination (a) and difficulty (b). We want these estimates to be **stable** (not fluctuating wildly with more data). If an item's parameters swing a lot, it might mean the item is inconsistently interpreted or our model is off. We'll hold out a portion of data and see if estimated *a* and *b* on one sample predict performance on another. By M3, after enough student answers, we expect to have reliable estimates for frequently used questions (with standard errors below a threshold). We might report average standard error of difficulty across items – expecting it to drop as data grows.

- **Item Discrimination (a):** Ideally, our assessment items have a high discrimination (say a > 0.7 in 2PL). We will monitor the distribution of discrimination. If many items have a ~0 (bad items that don't tell us about ability), we'll remove or fix them. So a metric: % of items with a > 0.5 (for example) – we want this high.

- **Test Information / Reliability:** Using IRT, we can compute how much information (in the psychometric sense) our whole set of questions provides at various ability levels. We want the test (or sequence of questions the student gets) to be informative around the student's ability. Also, if we give a subset again, we'd like to see similar scores – that is test-retest reliability. Concretely, we might do an A/B where some students get two interleaved sets of questions targeting the same concept; we expect their mastery score from set A and set B to correlate strongly (this is a form of reliability check). Or if a student's θ is 1.2 today, after a short interval (with no learning in between) it should be similar.

- **Adaptive Assessment Efficiency:** Compare with a fixed quiz of N questions. If our system uses IRT to achieve the same confidence in ability estimate with fewer questions, that's a win. We'll simulate some student profiles through our

questioning strategy and count how many questions until the ability estimate's standard error falls below X. This expected number of questions is a metric – hopefully dropping as our item bank and selection improve. Target: achieve same precision in 3 questions that a traditional 5-question quiz would (just an illustrative goal).

Learning Outcome Metrics:

- **Δθ Over Time:** The change in student ability (θ) between the start and after a learning session or series of sessions. This is our primary metric for learning. We will compute average θ gain for students after using the system for, say, a month, controlling for starting level. We might present this as "students improved by 0.6 logits on average in topic X after completing the recommended microlearning modules" – and compare to a control or baseline (maybe those who just read textbook without our adaptive layer). Normalizing by maximum possible gain (like if pre θ was -1 and post is 0, that's significant) is also useful. We strive for statistically significant gains. For example, an ambitious target: students achieve *normalized learning gains* of >50% on post-tests, which would exceed typical lecture gains (~23% in traditional settings)[8].

- **Mastery Progression:** The fraction of skills or topics mastered per unit time. If we define mastery as reaching θ above a threshold for a concept, we can track how many topics a student "checks off" after using the system for some period. Faster progression indicates efficiency. We'd compare this to expected progression in a classroom. Our goal might be that *80% of active users master at least one new subskill each week* as measured by our mastery criteria. Also, time-to-mastery for a given concept (from first attempt to reaching threshold) is a metric – we want to minimize that without sacrificing retention.

- **Normalized Gain (Hake's g):** If pre/post tests are available for a module, we compute Hake's normalized gain = (Post % - Pre %)/(100% - Pre %)[26]. This tells what fraction of the "learning gap" was closed. We aim for high normalized gains. For instance, interactive engagement methods in physics average ~0.48 (48%)[27]. We'd like our approach to meet or exceed that, indicating we're as effective as good interactive classes. If initial pilot shows g ~0.3, we know to iterate on content or adaptivity.

- **Downstream Performance:** Ultimately, does using the system help in real evaluations? We will track, where possible, correlations between system usage and performance on external metrics (exam scores, course grades). While many factors affect these, a significantly higher exam score for frequent users vs occasional users (controlled for prior ability) would be a strong validation. This might be long-term to measure, but we include it in our evaluation plan (likely as part of a research study or A/B test in a course).

- **Engagement vs. Learning Distinction:** We will explicitly measure click-through, session length, etc., but we treat them separately from learning metrics. For transparency, we'll report both. For example: a feature might increase time-on-platform by 20% (engagement up) but if learning gain doesn't improve, we won't count it as success. We'll use **engagement metrics** (like daily active users, questions answered per session, etc.) as health indicators but optimize primarily for the learning metrics above. This ensures our system doesn't devolve into a clickbait recommendation engine but stays true to educational efficacy.

Safety & Accuracy Metrics:

- **Hallucination Rate:** The percentage of system answers or explanations that contain unsupported or incorrect information. We will test the system on a set of queries where we know the correct answer and see if the system ever presents something else as fact. RAG should minimize this[1], but we quantify it. Ideally, with proper retrieval grounding, hallucination should be <5% of responses. Any hallucination is taken seriously; we do adversarial testing with edge queries to find failure modes. As we add features, we ensure this rate does not creep up.

- **Refusal/Deferral Accuracy:** In cases where the system should not answer (outside scope or low confidence), does it handle appropriately? We measure how often it correctly refuses or defers. For example, if asked a question that our content doesn't cover, the system should say it can't help (and maybe suggest asking an instructor) rather than guess. We'll create some stumpers to test this. A 100% safe refusal on out-of-scope questions is the goal. Similarly, if a question is potentially harmful or a request for personal advice outside our domain, the system should refuse (we'll incorporate OpenAI safety best practices). We'll internally track any incidents of inappropriate content or violations (none expected since domain is education and we'll filter inputs).

- **Bias and Fairness Checks:** While harder to quantify, we will simulate users of different demographics in content and see if outcomes differ. For instance, if we have a mix of male/female voiced videos, do students consistently rate one higher? (If so, we ensure our retrieval isn't biased or we provide options.) Or ensure examples in content aren't culturally biased. These are qualitative but can be put into metrics like "Representation score of examples (scale of 0–10 by rubric)", aiming for inclusive content. Also, fairness of the adaptive policy: use evaluation data to confirm students of different initial ability all see improvement (no group is systematically shortchanged by the policy's choices).

Each milestone will have a **Evaluation Report** documenting these metrics. For example, after M2 we expect to report something like: *nDCG@3 improved from 0.65 to 0.78 with cross-encoder; median snippet length down from 4.5m to 3.0m; hallucination rate remained <1% in tested queries.* By M4, we'll report *learning outcome proxies*: perhaps in an A/B test, users on adaptive mode answered 20% more post-questions correctly

than those on non-adaptive mode (a significant Δθ uplift). These concrete metrics keep us accountable to the core goal: better learning, not just better click metrics.

6. Stepwise Roadmap (12-Week Plan with Milestones)

We propose a ~12-week roadmap, broken into 6 bi-weekly milestones (M1–M6). Each milestone delivers a functional increment of the system, along with evaluation against the metrics above and clear acceptance criteria. This ensures iterative progress and early risk mitigation. Below is the timeline with objectives and exit criteria:

- **M1 (Weeks 1–2): RAG Baseline with Strict Minimality** – *Scope:* Implement the basic retrieval augmented QA system. This includes setting up the vector database with our content, a simple BM25+embedding search, and prompt-based answer generation grounded on top result. We will enforce manual constraints on content length (e.g., if a top result is a 10-min video, the system instead summarizes it or says "too long, skipping"). Basic safety filters in place (no unsupported answers). *Deliverables:* a working prototype that given a question returns a short snippet answer with a cited source. We'll test it on a set of sample queries. *Acceptance Criteria:* Achieve at least **nDCG@3 = 0.7** and **Recall@3 = 0.85** on a small benchmark of queries (likely using the heuristic ranking). "Overkill rate" should be under **20%** (meaning at most 1 in 5 answers is longer than desired). No blatant hallucinations in a 50-question test (target 0% hallucinatory answers – system should defer if unsure). We will also conduct a **red-team test** of out-of-domain or malicious queries to verify the system refuses appropriately. If any unsafe outputs occur, we fix filters. M1 is essentially the alpha system that answers student questions with one best resource.

- **M2 (Weeks 3–4): Enhanced Retrieval – Cross-Encoder Reranking & Segmenter Integration** – *Scope:* Upgrade the retrieval and presentation. We integrate the cross-encoder reranker to improve relevance[2]. We also plug in the **video/PDF segmentation** pipeline: the system can return a timestamped video clip (with an accompanying transcript snippet) or a specific PDF section rather than entire files. We introduce output in a structured JSON format (for front-end consumption), containing fields like `{"snippet_text": ..., "source": ..., "estimated_length": ..., "confidence": ...}`. The system now actively refuses to return non-minimal resources: a new component (evaluator) checks any candidate's length, and if it's above threshold, it must either summarize it or choose the next best candidate. *Deliverables:* Updated prototype where queries return JSON with either a direct answer snippet or a safe failure mode if nothing short and relevant is found. Include the cross-encoder model deployed (possibly via HuggingFace or on our infra). *Acceptance Criteria:* Notable improvement in retrieval metrics: e.g., **nDCG@3 >= 0.8** (due to reranker) and **overkill rate < 5%** (the few long answers should now be eliminated or summarized). Median resource length should drop (goal: <3 minutes for videos, <2 pages for text). Also measure **response time** – we target still under ~5 seconds despite extra rerank step. Additionally, **hallucination checks** on a new set of questions should show no

increase (still ~0%). We expect some improvement in "time-to-first-useful-resource" as well, since reranking promotes the truly best snippet to position 1. We will also do a small pilot with internal users: have a few colleagues ask questions and give qualitative feedback, especially verifying that the returned segments indeed answer their questions without extraneous info.

- **M3 (Weeks 5–6): Pedagogical Layer v1 – Question Generation, Grading & Hints** – *Scope:* Add the interactive learning loop. The system, after giving the resource, now asks a follow-up question. We develop the question generator using few-shot prompts (with examples for different Bloom levels) and integrate it such that every answer snippet is followed by an automated question. Build the basic grading logic: likely start with a simple string match or regex for expected keywords for short answers, and correctness check for MCQs. Also implement hint logic: if a student's answer is wrong, the system can provide a hint (we'll craft a prompt for the LLM like "Given the content and the correct answer, provide a helpful hint to get there"). We start calibrating difficulty: incorporate a simple 2PL IRT model. Initially, we might estimate abilities and difficulties in a rudimentary way (even if data is sparse, we can simulate or use default values). *Deliverables:* An end-to-end demo: a user asks a question, gets a snippet, and is then presented with a follow-up quiz question; the system accepts an answer and gives feedback (correct or a hint and another try). All interactions are logged in a structured way. We also deliver initial **calibration results**: using historical Q&A data, we fit a preliminary IRT model for a subset of questions (or use known question difficulties if available) – essentially demonstrating our ability estimation working on past data. *Acceptance Criteria:* **Pedagogy quality** should exceed a baseline: e.g., when comparing our auto-generated questions to the content they came from, experts rate at least **70%** of them as on-topic and appropriately challenging (versus maybe 50% for baseline zero-shot LLM). At least **X** (we might set X = 3 out of 5) in average expert rubric score for these questions, which should be an improvement from baseline (we'll have scored baseline in M2). The grading should correctly mark at least, say, 90% of a test set of student answers (we can simulate with some known correct/incorrect answers). Item parameter estimates (a, b) on the historical data should make sense (e.g., high difficulty questions have lower % correct in logs) and be stable on a hold-out subset – we want, for example, a correlation of >0.9 between difficulties estimated on two separate samples from our log (indicating consistency). Essentially, by M3 we want to show that the system can not only answer questions, but also quiz the student and measure their learning with reasonable accuracy.

- **M4 (Weeks 7–8): Adaptive Content Selection – Contextual Bandits & Policy Tuning** – *Scope:* This is the introduction of the learning loop optimization. We define the multi-objective reward (see Section 7) and implement a bandit algorithm (likely a form of contextual Thompson Sampling or LinUCB) that will decide which content snippet to show when multiple are plausible, and potentially other decisions like whether to show a video vs text if both have equal relevance.

We will likely start with **off-policy evaluation** using historical data or simulation to ensure the bandit's policy would have improved something. For example, we can simulate students with certain learning parameters going through content chosen by random vs. chosen by a greedy difficulty-match policy, and see that our algorithm would choose the one leading to better score gains. During these weeks, we might not fully deploy the bandit live (unless we have enough live traffic to learn), but we set it up and run it on collected data in batch mode (training on logs up to week 6 and then seeing how it would behave). We also implement guardrails in the policy (don't choose content above a certain length, etc., as described in RL design). *Deliverables:* A trained initial bandit policy (possibly just a weighted linear model over features like content length, difficulty match, past success with that content, etc.) and a report on its evaluation. E.g., off-policy evaluation (IPS or doubly robust) showing the policy's reward would have been higher than baseline policy's reward on our logged dataset[14]. We also deliver the integrated system where the orchestrator can call the bandit to select among, say, the top 3 retrieved snippets which one to show (introducing adaptivity beyond pure relevance). *Acceptance Criteria:* We need to demonstrate a **proxy uplift in learning outcome metrics**. For instance, using a holdout set of student simulations or real data, the contextual bandit's choices yield a higher average $\Delta\theta$ or quiz score than always taking the top-ranked snippet. If our reward is composite, we might show that *expected reward per interaction increased by >=10%* with the bandit compared to a non-adaptive baseline. Also crucial: this improvement must not come at the cost of content length or accuracy. So we check **minimality** and **hallucination** remain at least as good as before. If the bandit suggests longer content for marginal learning gains, we'll have a penalty in reward to avoid it. So by acceptance, we might state "Bandit policy improves estimated learning gain by +15% while keeping overkill rate < 5%". We'll also verify the bandit is *exploring properly but safely*: e.g., no student in the trial got an obviously irrelevant resource. Off-policy evaluation with techniques like IPS should show statistically significant better reward for the learned policy (with $p < 0.05$ if possible, to be rigorous). This milestone is critical as a proof-of-concept that adaptive selection can move the needle on learning proxies.

- **M5 (Weeks 9–10): Refinement – LoRA Fine-Tuning for Pedagogy & A/B Testing vs Prompt Baseline** – *Scope:* Now that the core loop is in place, we go back and fine-tune some components to improve quality and efficiency. We'll pick one or two areas where prompt engineering hit a ceiling and apply LoRA fine-tuning. Likely candidates: the question generation style (to consistently produce, say, a higher cognitive level or a more concise question than the raw GPT might) and the grading rubric model (to reduce instances of mis-grading). We create small fine-tuning datasets from our accumulated interactions and earlier labeling. For example, compile 200 examples of content → question pairs that are high quality (perhaps some from our experts and some we cherry-picked) and fine-tune an LLM's outputs toward those. We also might fine-tune a smaller model (like

Llama-2 13B with LoRA) to act as the grader, which could cut down the reliance on GPT-4 for that task. These fine-tunes will be done with careful evaluation – we'll run the tuned model vs. the original prompt approach on a test set of prompts. We also set up an A/B experiment in the system: some fraction of users get the new fine-tuned question generator, others get the original prompt-based, and we compare outcomes (like user satisfaction, learning gains in that period, etc.). *Deliverables:* Updated models (LoRA adapters) and documentation (model cards describing what data we used and how it alters behavior). We'll also produce an **A/B test plan** detailing how we will measure the impact of these fine-tuned components on real users (e.g., measure expert rubric scores as we did offline, but now on questions actually posed to users, or track if fine-tuned grader leads to fewer hint requests because it gave more accurate feedback). *Acceptance Criteria:* The fine-tuned components must show **quantitative improvement** over prompt-only. For question generation, perhaps the average expert score goes from 4.0 to 4.5 out of 5 after fine-tuning (or the proportion of higher-order questions increases). For grading, maybe agreement with human grading on a test set goes from 85% to 92%. These improvements should come **without a performance penalty**: latency should remain similar (our LoRA-loaded model shouldn't be slower than the API call by a big margin) and cost per call likely drops if we move some tasks to an open model. We require that any fine-tuned model's outputs pass our safety checks (no regression in appropriateness – we'll run the same red-team prompts to ensure nothing odd). Essentially, acceptance is "Fine-tuned modules outperform the prior implementation on their respective quality metrics, confirmed by both offline evaluation and a small-scale user A/B (if available), at no more than 1.2x the inference cost." If, hypothetically, prompt-based QG vs tuned QG have no difference in quality, we might decide not to deploy the tuned one and stick with prompts for now (thus we'd fail the acceptance unless tuning clearly helps). This milestone is about validating that targeted fine-tuning yields worthwhile gains.

- **M6 (Weeks 11–12): Production Hardening & Continuous Learning Framework** – *Scope:* The final stretch focuses on making the system robust, secure, and ready for scale. We implement **guardrails and monitoring**: e.g., integrate an open-source tool or custom checks for any LLM output to catch toxic or biased language (should be rare in our domain, but we add it). Ensure PII handling – if a user question includes personal info ("My grade is low in class, what should I do?"), the system should not leak that or process it inappropriately. We'll put in place data anonymization for logs (to comply with privacy). Set up **telemetry dashboards** to live-monitor metrics like: overall correctness rate, average content length served, any spikes in errors or refusals, etc. This includes monitoring the bandit policy's performance in real-time – e.g., we can log the reward components for each recommendation and track a rolling average (so we can quickly detect if a change caused a drop in, say, learning gains). We also plan for **continuous recalibration**: design how the IRT model will update item params as new data comes (perhaps a weekly batch job refitting difficulties, with anchors to

avoid drift). Similarly, how the bandit will retrain (maybe it updates nightly on the day's data). We also conduct a **bias audit**: review content and model outputs for any inadvertent bias (ensuring examples used aren't marginalizing any group, etc.). And ensure compliance with educational data regulations (FERPA in US) – e.g., define data retention policies, and expose a way to delete a user's data if requested. Essentially, get all non-functional requirements in place. *Deliverables:* A deployment-ready system with documentation for devops (how to deploy new model versions, how to rollback if an issue, etc.), a set of **guardrail tests** (like unit tests for safety: feed some problematic queries and assert the system refuses), and a **dashboard** screenshot showing key metrics trending. We will also have a plan written for ongoing item parameter recalibration (maybe included in model card or separate doc). If applicable, a compliance report or checklist (covering GDPR if any EU users, etc.). *Acceptance Criteria:* Stakeholders sign off that the system meets organizational standards for release. Concretely: **Safety** – internal red-team testing reveals no unhandled safety issues (any found are fixed or mitigated). **Privacy** – a review with our privacy officer (if we have one) shows we log no sensitive PII in an unsafe manner (e.g., any user-provided name or ID is either not logged or properly anonymized), and we document data flows. **Performance & Scaling** – load test results should indicate the system can handle the expected concurrent users (we'll set a number, say 100 concurrent sessions, with response p95 < 5s). **Monitoring in place** – team can observe real-time and be alerted on anomalies (like if hallucination rate goes above some threshold or if any API fails). Also, by this point, we should have demonstrated in a pilot that learning outcomes are trending positive (even if not yet a formal study). We might include as acceptance: at least **X% normalized gain** on a pilot group or a significant improvement in post-test scores for users. While this may extend beyond 12 weeks to fully verify, by M6 we expect preliminary signals. Lastly, a **go/no-go meeting** is held where we present all evaluation results from M1–M6; a "go" decision means we're ready for wider rollout.

Throughout the roadmap, at each milestone we produce detailed evaluation reports. If a milestone's acceptance criteria are not met, we will allocate a hardening sprint or adjust before moving on – this ensures we don't pile on features without solid foundation. By the end of week 12, we aim to have a system that is not only functional but shown to improve student learning in measurable ways, with all necessary safeguards, ready to deploy to a broader audience or test in a classroom setting.

7. Reinforcement Learning Design

Designing the reinforcement learning (RL) component for our system requires a clear reward definition and a practical, safe training approach – especially given limited initial data. We opt for a *contextual bandit* framework as a first step towards full RL, focusing on the immediate outcome of each content recommendation. Here's our RL design:

Reward Definition (Multi-Objective): We propose a reward signal that balances multiple objectives, formulated as:

$$R = w_1 \Delta\theta + w_2 \cdot (Minimality\ Bonus) - w_3 \cdot (Hallucination\ Penalty) - w_4 \cdot (Latency/Cost\ Penalty)$$

Breaking this down: - The primary term $\Delta\theta$ is the estimated *learning gain* from the action – for instance, the increase in the student's ability θ after answering the follow-up question (if the student got it right, especially if they previously struggled, that's a positive gain). This could also be proxied by the **improvement in quiz correctness**: e.g., reward 1 if the student answers the quiz correctly (meaning the content helped them learn/recall), and perhaps an extra reward if this question was one they previously got wrong (indicating overcoming a misconception). We weight this by w_1 , the most important factor. - A *minimality bonus* encourages brevity: e.g., we give +1 if the content delivered was under the target length (or more granularly, $+\ 0.1$ for each minute under the cap up to some limit). This ensures the bandit doesn't start favoring longer content just because sometimes it has higher chance of answer (we explicitly reward keeping it short). We set w_2 such that a minimal snippet gets a notable boost in reward. - A *hallucination/irrelevance penalty:* if the student's quiz answer was wrong and our system evaluation indicates the content might not have covered the answer, that triggers a penalty. Similarly, if the student quickly abandons the resource (say, they closed it in 5 seconds, perhaps indicating it was off-target), that could incur a negative reward. This term (with weight w_3) penalizes selecting content that turned out not useful or confused the learner. - A *latency/cost penalty:* to gently discourage very heavy actions (like fetching a huge LLM response unnecessarily or using an extremely high-res video). If one action type is significantly slower or costlier (say, using a big model for a trivial task), we assign a small negative for that to push the policy towards efficiency. Weight w_4 will be small (we don't want to degrade learning for cost too much, but if two choices are equal in learning value, it will prefer the cheaper/faster one).

These weights $w_1 \ldots w_4$ can be tuned. Initially, w_1 (learning gain) is by far the highest, reflecting our priority. w_2 ensures tie-breaking in favor of brevity. w_3 is also high because hallucinations or irrelevant suggestions are quite bad (they waste student time and trust). For example, we might start with $w_1 = 10$, $w_2 = 1$, $w_3 = 5$, $w_4 = 1$ (scale is arbitrary, these would be normalized or adjusted as we see the magnitude of each component). Over time, we can adjust these via multi-objective optimization or even let an algorithm tune them if we set a higher-level objective (but likely we'll do manual/heuristic tuning at first).

Context and Action Space: The *context* for the bandit at decision time includes features of the student (e.g., current ability θ, maybe a one-hot of their identified knowledge gaps or learning style if known), the question asked, and features of each candidate content snippet (length, content type, difficulty level of content, whether the snippet directly contains the known answer keywords, etc.). We will likely engineer a feature vector combining these – some features continuous (differences between content difficulty and student ability, etc.) and some categorical (content modality, source quality rating, etc.). Since our action is basically "choose which content snippet (or which strategy) to show",

the arms correspond to the top N retrieved snippets, plus maybe a special action like "ask for clarification" or "defer". We include a "do nothing" or "cannot find good answer" action as well, which leads to the system refusing to answer (with a safe message). That action's reward structure is such that it's neutral or slightly negative (we prefer to give an answer if we have one, but refusal is better than a wrong answer, so a wrong answer would incur a larger penalty anyway making refusal relatively better in those cases).

Algorithm – Contextual Bandits First: We will implement a **Thompson Sampling** or **Upper Confidence Bound (UCB)** contextual bandit as our initial algorithm. This is simpler than full RL (no long-term credit assignment, just each interaction). Bandits are known to work well for sequencing and recommendation problems in education[9] and require less data to start making reasonable decisions (since they optimize immediate reward)[28]. Concretely, we might use a linear model for expected reward: $\hat{R} = \theta_{policy} \cdot x_{context,action}$ (with x being feature vector for a context-action pair), and TS or UCB to handle exploration. We'll start with relatively aggressive exploration (to learn what works) but clip it to safe options as noted. The bandit will learn, for example, if videos or texts yield higher quiz success for different types of students, or if a certain content source consistently yields better outcomes (it will shift probability mass to those).

Off-Policy Evaluation: Since we can't risk a completely untested policy on real students (especially early on), we will leverage **Inverse Propensity Scoring (IPS)** and **Doubly Robust** estimators on our logged data[14]. We have logs from the heuristic policy (which is like a uniformly random or top-1 always selection, depending on how we did cold-start). Because we know which content was shown and the outcome, we can retrospectively ask "if the bandit policy had been in place, what reward would we have gotten?" IPS will weight those logged events where the bandit would have made the same choice, adjusting for probability. We'll use this to get an unbiased (though high variance) estimate of new policies before deploying them. Over time, as we collect data with the bandit running (with some exploration), we'll continuously evaluate new candidate policies offline with these techniques – essentially A/B testing policies in simulation before committing them.

Safety Constraints: We incorporate **hard constraints** into the policy. For instance, the bandit's action set can be pruned of any content snippet above X length (so it simply cannot choose an overlong video – that enforces minimality from the get-go). Also, we'll restrict actions that conflict with safety; e.g., if a snippet hasn't been vetted or has some flag, the agent can't choose it. If uncertainty is high (the bandit hasn't seen enough of a certain type of content), we might have a rule that requires a minimum of evidence before letting that content be shown frequently (maybe initially throttle novel content suggestions).

We also plan for **uncertainty-aware deferral**: if the top N snippets all have low predicted reward (maybe the bandit model is uncertain or thinks none are good), the agent can

defer (choose an action like "I'm not confident – escalate to human/expert or give a generic safe answer"). This is important in education because giving wrong info can be worse than giving none. We will incorporate model uncertainty (like the variance in Thompson sampling or confidence intervals in UCB) – if the best arm's lower confidence bound is below some threshold, we trigger a deferral or a high-level hint rather than potentially misleading the student.

Transition to Full RL (if needed): If we find that immediate reward optimization isn't enough – for example, maybe the bandit learns to give easy questions to get quick reward but that could slow long-term learning – then we might consider a full RL approach (MDP) where the state includes the student's knowledge state and the goal is to maximize total learning over a sequence (like maximizing final θ after a series of interactions). However, full RL is data-hungry and complex (the "curse of dimensionality" for states in personalized learning is known[29]). We'll only escalate if bandit policies plateau or start showing undesirable myopic behavior. We would then leverage simulators or use the bandit policy as a baseline to train a policy-gradient agent (maybe using *reinforcement learning from human feedback (RLHF)* style – where human experts could occasionally compare two learning trajectories and choose which is better, feeding a reward model). That's an advanced step likely beyond our 12-week horizon, but the architecture leaves room for it.

In summary, the RL design starts simple: **multi-objective contextual bandit** focusing on immediate post-question success and brevity, evaluated offline carefully, deployed with safety in mind. This aligns with prior research that bandits can effectively personalize learning sequences with much less data than full RL[9]. Over time, as we get sufficient data (say thousands of interactions), we can enrich the state (maybe include a short history of past successes/failures – effectively making it a limited-memory MDP) and try more sophisticated approaches. But the bandit will already give us adaptive content selection, balancing exploration of new content strategies with exploitation of what seems to work best for learning gains[13].

8. Fine-Tuning Policy (When & How to Fine-Tune Models)

Our strategy is to be **judicious** about fine-tuning any large models – especially early on. We start with prompt engineering and small adaptations, and only fine-tune when evidence shows it's necessary and beneficial. Key principles and triggers for fine-tuning in our plan:

- **No Full-Model Fine-Tune Initially:** We explicitly avoid fine-tuning the foundation LLM (like GPT-4 or Llama-2) on our domain out of the gate. The base models are very powerful general reasoners; we don't want to risk *catastrophic forgetting* or narrowing their knowledge base by overfitting to our relatively small domain data. Initially, prompts (possibly with few-shot exemplars) will suffice. For style or specific behaviors, we prefer prompt techniques (system messages, etc.). This keeps us agile to swap out the underlying model (e.g., if a new better LLM comes along, we can move to it without re-training from scratch).

- **Identify Task-Specific Modules for later FT:** As the system evolves, we isolate which tasks could benefit from a tailored model. These likely are:

- (a) **Question Generation module:** Perhaps a smaller language model fine-tuned on educational Q&A pairs to consistently produce well-structured questions at the right difficulty. E.g., fine-tune Flan-T5 or Llama-2 on a dataset of textbook questions labeled by Bloom level, so it internalizes educational phrasing and avoids common mistakes (like avoiding hints of the answer in the question).
- (b) **Rubric-Based Grader:** A model (maybe a classifier or a T5) fine-tuned to score short answers against a reference. This could even be a BERT-based classifier for each rubric dimension. The goal is consistency – ensure the same answer always gets the same score, matching expert graders. Fine-tuning here can dramatically improve consistency over few-shot prompting.
- (c) **Distractor Generator:** A specialized model that, given a question and correct answer, generates plausible distractors. This can be trained on existing multiple-choice question banks (where distractors are known good ones). A fine-tuned model might produce more pedagogically-sound wrong answers than a generic LLM (which sometimes either gives obviously wrong or too tricky ones).
- (d) **Style/Explanation fine-tune:** We might train an adapter on the large LLM to speak in a certain pedagogical style – e.g., always use encouraging tone, avoid jargon – if we find prompts alone don't reliably enforce this. Also, an adapter that compresses explanations (for the "short-explainer" style) could be useful: e.g., fine-tune a model to summarize answers in <100 words for hint purposes.

We will use **Low-Rank Adapters (LoRA) or other PEFT methods** for these fine-tunes whenever possible. This means we add a few trainable matrices to the model (which is memory-efficient) and train on our task-specific data while the original weights stay frozen[4]. The result is an *adapter* that can be merged or activated during inference to modify the model's behavior. The beauty is that these adapters are small (often <0.1% of model size[30]) – we can even maintain multiple (like one adapter for "explain like a patient tutor" and another for "succinct answer") and toggle them as needed.

When to Fine-Tune (Entry Criteria): We set concrete criteria to decide a fine-tune is warranted: - We have at least **N_k high-quality examples** for the task k. For instance, to fine-tune question generation, maybe we decide we need 500+ good question-answer pairs with context and difficulty labeled. Or to fine-tune grading, we need a few hundred student answers with scores. These ensure the model has enough signal to learn from without overfitting severely. We won't fine-tune if we only have, say, 50 examples – then we stick to prompting. - The current prompt-based approach has **plateaued** in performance or has an identified limitation. E.g., if after many prompt tweaks, our grader still mismarks certain edge cases or has variance, that indicates a fundamental limit of zero-shot/few-shot approach on that model – fine-tuning could help it generalize more consistently on those patterns (as known from literature that fine-tuning can solidify behaviors). - We project that fine-tuning will yield a **net positive in efficiency or consistency**. For example, if we fine-tune a 7B model for grading, we could use it

instead of calling a 175B model via API for each answer – huge cost and speed savings if it maintains accuracy. Or if fine-tuning yields +10% higher rubric scores for generated questions, that consistency gain justifies it. Essentially, we do a cost-benefit analysis: training time and potential model maintenance vs. improvements. - **Governance approval:** Especially for using student data in fine-tuning, we likely need to get stakeholder consent (and perhaps student consent if these are personal answers). We'll have a model governance process (perhaps internal review board) to ensure fine-tuning won't encode biases or privacy issues. Only after that approval do we proceed. Also, if using a licensed model (like GPT-4 via API), we need to consider if fine-tuning is allowed or whether we switch to an open model for that component.

Migration Plan for Model Upgrades: As foundation models evolve (say GPT-5 or a new open model with better capabilities appears next quarter), we want to **retain our fine-tuned knowledge**. Our approach: - Use adapters that are ideally model-agnostic or easily transferable. For example, LoRA applied to Llama-2 won't directly plug into GPT-4 obviously, but if we stick to open models for fine-tuning tasks, we can upgrade within that family (like moving from Llama-2 13B to Llama-3 13B later, we might need to re-tune but can use the same training data and perhaps even initialize using the old adapter weights if the architecture is similar). We keep our training scripts and data versioned so that we can recreate adapters for new base models quickly. - Decouple skills: Each fine-tune is on a smaller model for a narrow task, not on the all-purpose LLM. This means if our main LLM gets upgraded, we likely continue to use it with prompts for general QA and perhaps retrain only the small modules if needed. For example, suppose we fine-tuned a 7B grader model and a new 7B model comes that's much better; we can fine-tune the new one on the same grading dataset and slot it in, with minimal changes to other parts. - Maintain fallback to prompting: For any fine-tuned piece, keep a prompt-based fallback available. For instance, if our fine-tuned question generator model performs poorly on an edge case, the system could detect it and use the general LLM as backup. This ensures robustness during the transition period of any model migrations. - **Adapter versioning:** When the base model updates (say Llama-2 to Llama-3), we might need new adapters. We version them (e.g., QGen_v1_LLama2, QGen_v2_Llama3). We'll run parallel tests to ensure the new base+adapter works at least as well as old base+adapter before fully switching. - **Continuous fine-tuning process:** As more data comes (flywheel), periodically we will refresh the fine-tunes. Instead of a one-off event, we plan perhaps every quarter to check if we've amassed significantly more training data for a module, and then update it. We use *adapter fusion* or *incremental training* to add new data without forgetting old performance (or simply include all data in a new fine-tune from the original base if that's safer). This controlled schedule prevents constant churn but keeps models up-to-date with evolving usage.

In summary, **fine-tuning is applied sparingly**, on smaller models or via adapters on big models, only when we have enough data to do it well and when it clearly addresses a shortcoming. This ensures we remain agile: we're not locked into an old fine-tuned giant model while the world moves to new ones. Adapters give us plug-and-play upgrades – for example, if a new version of our base LLM supports the same adapter technique, we

can often reuse or lightly retrain adapters and get improved base performance plus our customizations.

To illustrate, imagine by week 8 we have 1000 good Q&A pairs. Instead of fine-tuning GPT-4, we fine-tune Llama-2 13B with LoRA on those Q&A. If it matches GPT-4's question quality, we use it for question gen. If next month a Llama-3 13B comes with better language skills, we quickly fine-tune Llama-3 on the same data and swap it in – minimal disruption. Meanwhile, our core GPT-4 (if we still use it for answering or such) remains untouched and can also be upgraded by OpenAI without breaking our system – since we only ever interacted via prompts.

Checklist for Fine-tuning Decision: We will formalize a checklist that must be greenlit: - Data volume & quality sufficient (check). - Offline metric plateau with prompting (check). - Potential benefits outweigh maintenance (check). - Bias/ethical review passed (check). - Model choice (which base to fine-tune) made with long-term in mind. If all good, we proceed to fine-tune and then do **extensive testing** of that model in sandbox before deploying.

By following this cautious, data-driven fine-tuning policy, we ensure that we leverage fine-tuning when it truly adds value, and we do so in a way that doesn't compromise our ability to move with the tide of rapidly improving foundation models.

9. Risks & Mitigations

Implementing an adaptive learning system at scale comes with various risks. We enumerate the key risks and our plans to mitigate each:

- **Cold-Start Content Recommendations:** *Risk:* At launch, with no prior pairing of questions to resources, the system might suggest suboptimal or tangential resources, frustrating early users. *Mitigations:* We use the heuristic and weak-label strategies described in the data plan to bootstrap relevance. Additionally, we will have a **human-in-the-loop** override for initial users: for example, in a beta launch, we could have an expert quickly review the top suggestion for each question asked and intervene if it looks off. This "teacher review" safety net (even if it's sampling 10% of queries) can catch egregious misses early. We also plan to start with a smaller scope of topics, ensuring content coverage is good for those before expanding – thus reducing the chance of an unanswerable question. As data grows, the risk diminishes, but the first few weeks are crucial; hence we focus heavily on the M1/M2 evaluation to ensure decent recommendations out of the gate.

- **Over-long or Over-complex Resources (Violating Minimality):** *Risk:* The system might frequently default to long videos or entire chapters (especially if those have high keyword overlap) because it hasn't learned to choose shorter segments, leading to information overload for simple questions. *Mitigations:* We set **hard caps** on content length at the retrieval stage – any document chunk beyond X length is split or not considered. We also bias the retriever via a

“coverage-per-minute” scoring: effectively re-rank by (relevance score / length). This directly promotes concise sources[3]. The cross-encoder can be trained or prompt-engineered to favor passages that answer succinctly (we can include a feature for length in its input). The system’s evaluator will outright reject content beyond policy limits (by design at M2). If there’s truly no short answer available, the agent might instead break the question into sub-questions or ask the user to clarify, rather than dump a huge resource. In user interface, if a longer video is given (in rare cases), we’ll provide an estimated watch time and maybe an option like “too long? click to get a summary” to manage expectation. So, product design plus algorithmic bias will mitigate this.

- **Hallucinations or Inaccurate Answers:** *Risk:* The LLM might generate explanations or answers not supported by the retrieved content, or the system might mis-grade a correct student answer as wrong, etc., thereby misleading or discouraging learners. *Mitigations:* We lean on **retrieval-grounding** heavily – the LLM is instructed to only use provided sources. We will implement a **verification step**: after the LLM generates an answer or explanation, we can have another pass (or another model) check it against the content. For example, use a smaller model to do an answer comparison: does the answer contradict the content? If yes, we refuse and instead present the content directly or escalate to a human. This reduces authoritative hallucinations. For grading, we mitigate errors by having rubrics and potentially multi-model agreement (e.g., only mark wrong if both the rubric check and an LLM check agree it’s wrong; if there’s ambiguity, maybe ask the student to clarify rather than outright mark incorrect). Additionally, initial red-team testing focuses on factuality: we’ll test known difficult questions where the LLM might be tempted to fill gaps, and ensure our pipeline catches that. If the retrieval fails (no good source found), the orchestrator will choose to say “I’m not confident” rather than letting the LLM just guess – that path is explicitly built in as a guardrail (with a slight negative reward so it’s used when needed). In summary, by design, *no final answer or explanation is given without being either directly extracted or cross-checked with source material*, minimizing hallucinations[1].

- **Misaligned Difficulty (Too hard or too easy content/questions):** *Risk:* The system might give resources or ask questions at the wrong level – e.g., showing an advanced proof to a novice, or conversely, boring the student with simplistic material they already know. This could hurt engagement and learning. *Mitigations:* We address this via our difficulty calibration in the student model. The orchestrator uses the student’s estimated ability (θ) and the content’s difficulty metadata (or the difficulty of the quiz question) to try to match level. We will maintain **prerequisite relationships** between topics – the planner won’t jump ahead unless mastery of prior topics is indicated. If we detect a student consistently getting things right quickly, the agent can skip ahead or offer a more challenging extension question. Conversely, if wrong twice, agent can drop to an easier sub-concept. To keep the calibration honest, we’ll **recalibrate the IRT**

model periodically, using anchor items (some questions that have known stable difficulty, given occasionally to serve as reference). If drift is detected (maybe our difficulty estimates shift because student population changed), we update the model. The bandit's context also includes past performance, which helps it avoid giving too difficult content (since that would lead to low immediate reward). Lastly, we'll incorporate user feedback: if a student can rate a resource or question as "too easy/hard", we feed that back as a signal (possibly adjusting θ or marking that resource's difficulty differently for future).

- **Privacy and Security (Student Data & PII):** *Risk:* Handling education data (which might include personal identifiers, performance records, etc.) comes with legal and ethical obligations (FERPA in US, GDPR in EU). A breach or misuse could be very damaging to students and the organization. *Mitigations:* We implement **strict data governance** from day one. All personal data (names, emails, etc.) will be separated from usage data. For instance, when logging interactions, we use a random student ID and not their real name. We'll store any mapping of ID-> identity in a secure, access-controlled system, separate from the main data. Only aggregated or de-identified data will be used for model training (this is important for privacy compliance – e.g., we can use students' answers to fine-tune a grader only if de-identified and with consent). We will provide an option for students to opt-out of data collection beyond what's necessary for functionality. For compliance: we ensure data is stored encrypted, and if using any cloud services, that they are within allowed regions (for GDPR, EU servers if needed). On the model side, we will instruct the LLM never to request personal info. If a student asks something revealing (like "I have dyslexia, how should I study?"), the system should help but carefully (this crosses into possible sensitive data territory – our policies will likely treat that as within scope because it's educational, but we must handle it sensitively and not log the condition with identity attached). In summary: minimum data retention, anonymized analytics, and possibly a **FERPA compliance check** with legal – e.g., our system could be offered as an on-premise solution for schools so that student data never leaves the school's environment, if needed. Technically, we'll also implement role-based access control in our dashboards – only authorized team members see student-level data, and even then no PII. All these reduce the privacy risk. We also plan periodic **security audits** of the system, especially if it's storing assessment results (to prevent hacking or leaks of exam info).

- **Bias in Recommendations or Content:** *Risk:* The system might inadvertently favor certain content providers or types, not due to efficacy but due to some artifact (e.g., always recommending videos from one source because they have high production value but maybe they cater to a certain demographic). It could also present examples that reinforce stereotypes (if underlying content has bias). *Mitigations:* We'll monitor recommendation patterns. Because we have a multi-objective reward that focuses on learning gain, the system inherently tries to be fair (it doesn't care about source except for outcome). But we will explicitly

enforce diversity if needed (e.g., if two resources are equal, rotate sources to give a breadth of perspectives). Also, in content ingestion, we aim for a diverse library that represents various backgrounds (especially in examples and contexts). For instance, if math problems all feature male names, we'd adjust that content. This is more content curation than algorithm, but our system can highlight if maybe students of a certain group consistently get lower reward – which could hint the content isn't inclusive. We'd then investigate and adjust. In bandit terms, we could add fairness constraints (like ensure each content provider gets at least some exploration, or ensure that the policy's outcome is not discriminatory – which is complex, but we can measure performance by demographic if data is available in aggregate). Our diversity-aware design from the start (the references like DABSEC bandit that clusters diverse user traits[31] could inspire future improvements to ensure the system personalizes without bias).

- **Model or Policy Drift:** *Risk:* Over time, as models are updated or the user population changes, the performance could degrade. E.g., a fine-tuned model might become misaligned if curriculum changes (say, definitions change or new convention, the model might give outdated info). Or the bandit policy might exploit some weird loop (like it finds showing a certain easy question yields good immediate reward but in long run not ideal). *Mitigations:* Continuous monitoring via our dashboards will catch drifts – e.g., if overall post-quiz scores begin to drop or certain content starts getting bad feedback suddenly. We'll have **alert thresholds** (like if mastery gain in a topic falls below historical baseline, flag it). For model drift, we largely control our models, but if using external ones (OpenAI), changes can happen. We mitigate by regularly re-evaluating the system on our test sets whenever the external model changes (OpenAI often informs if a significant change, and we'd run a regression test). Keeping human oversight in the loop, especially for any major system decisions (like releasing a new policy or model) is essential: e.g., an educator panel periodically reviews system outputs to ensure continued quality.

- **Student Frustration or Dependency:** *Risk:* If the system is not clearly improving learning, or if it's too intrusive (quizzing too much) or not aligned with teacher's curriculum pacing, students might disengage or even be misled in their studies. Conversely, they might over-rely on it for answers instead of thinking (if not well designed). *Mitigations:* We will refine the UX such that the system feels like a helpful guide, not a strict examiner. E.g., ensure the tone is encouraging (which we do via style guidelines). We allow students to skip or say "I want to just see the answer" – giving them autonomy to avoid frustration. We will also integrate with teacher oversight: providing teachers a dashboard (Stakeholder view) to see what the AI is suggesting and maybe moderate it. Teachers can input learning goals for each student or class so the system aligns with those (ensuring, for example, it doesn't jump to content that teacher hasn't covered yet in class – that alignment reduces confusion). Also, we'll gather user feedback systematically

(thumbs up/down on resources, etc.) and quickly react if patterns of dislike emerge for certain features. Regarding dependency: by focusing on *formative assessment* (the system always nudges the student to answer questions, not just giving direct answers), we mitigate the risk of the AI becoming an answer vending machine. The student must do some work, during which learning happens.

In essence, our risk management is a combination of **design-time safeguards** (hard constraints, policies), **real-time monitoring** (dashboards, user feedback loops), and **iterative improvement** (using data to detect issues early and fix them). By anticipating these risks and addressing them in each milestone (e.g., M6 is explicitly for hardening and checks), we aim to ensure the system is not only effective but also trustworthy and equitable.

10. Deliverables per Milestone

To maintain transparency and reproducibility, we will produce a set of concrete deliverables at each milestone, ensuring all stakeholders (developers, educators, management) can validate progress. Below is what we'll deliver at the end of each milestone:

- **M1 Deliverables:**
- *Prototype Notebook/Pipeline:* A Jupyter notebook or script demonstrating the RAG baseline – given a sample question, it performs retrieval and outputs an answer snippet. This will be fully reproducible (with maybe a subset of content).
- *Preliminary Data Card:* Documentation of the initial content dataset (what content we loaded, sources, any preprocessing). And a simple Model Card for the base LLM usage (describing GPT-4 or whichever used, and our prompt format).
- *Evaluation Report M1:* A report (likely a PDF or markdown) showing the offline retrieval metrics (nDCG, recall on a small set) and examples of system answers for a list of test questions, including any mistakes and analysis. It will also include results of red-team tests (like a table of problematic queries and how the system responded).
- *Decision Memo M1:* A short memo for internal record stating whether acceptance criteria of M1 were met and lessons learned, plus go/no-go for M2 development. For instance, if recall was low, note plan to improve in M2.

- **M2 Deliverables:**

- *Updated Code Pipeline:* The code integrated with cross-encoder and segmentation. Possibly a Docker container or a REST API exposing the Q&A system endpoints returning JSON. This is now beyond a notebook – could be a small microservice running locally for test.
- *JSON Output Examples:* A set of example JSON outputs for test queries (to show the format and contents like snippet, source citation, etc.).

- *Segmenter Module Documentation:* A brief doc on how we do video/PDF segmentation (e.g., “we use YouTube’s API for captions, then split by time or topic; for PDFs we split by headings etc.”).
- *Evaluation Report M2:* Updated metrics (retrieval improved, median lengths, etc.) and analysis. Include a section on how the cross-encoder re-ranked some examples (like “question X: originally a long PDF was rank1, now a shorter paragraph is rank1 after rerank – and indeed it answered the question”). Show before/after content length stats. Also any continued hallucination tests.
- *Red Team Log M2:* If any new issues found with the more complex pipeline (e.g., the cross-encoder might erroneously bump something irrelevant if it had weird semantics – we’ll report that if happened and how we fixed it).
- *Acceptance Test Results:* A checklist of acceptance criteria from M2 and whether each was met (for traceability).

- **M3 Deliverables:**

- *Pedagogy Module Code:* The new components (question generator, grading function, etc.) likely in modular form (maybe as Python classes or functions). Also the prompt templates or few-shot examples used for QG and hints, included in a “prompt library”. We’ll maintain a prompt library file that contains all the prompt patterns for various tasks (for version control and easy editing).
- *Rubric and Hints Library:* Documentation of how we generate hints and what the rubrics are for sample questions (this could be a set of JSON or YAML defining rubrics for topics – if we did that).
- *Data Artifacts:* If we used existing logs to calibrate IRT, deliver the dataset and the resulting item parameter estimates (perhaps in a CSV or so).
- *Evaluation Report M3:* This focuses on pedagogical quality. It will include samples of AI-generated questions alongside the content they’re based on, with expert commentary or scores. It will include grading accuracy results (maybe a confusion matrix comparing AI grade vs. human grade on some sample answers). It also documents initial IRT calibration: e.g., a table of a few questions with their difficulty and discrimination values that make sense (and possibly a plot of an ICC – Item Characteristic Curve – to show we did it).
- *Learning Analytics Dashboard (prototype):* Perhaps a screenshot or live demo of a simple dashboard showing a student’s θ over time or question outcomes. By M3, we might have a basic internal dashboard to visualize data (for developers/educators).
- *Updated Model/Data Cards:* Model card for the question generation approach (documenting we used GPT-4 with X prompt, known limitations like it sometimes asks too hard questions, etc.), and data card updates if we created new datasets (like “Bloom questions dataset v1” if we curated some).
- *M3 Decision Memo:* Summarizing whether the pedagogy features are meeting quality bars and if any pivot needed (e.g., if automated grading isn’t reliable enough, plan to involve teacher verification at first, etc.).

- **M4 Deliverables:**

- *Bandit/RL Module Code:* The policy implementation (e.g., a Python class that given context returns action). This includes the training code that produced the current policy. We will likely also deliver a saved model of the policy (parameters) for transparency.
- *Off-policy Evaluation Notebook:* A notebook showing how we estimated the new policy's performance using logged data (with IPS or other). This is important for stakeholders to trust the bandit – we can share "look, on historical data, our policy would have improved quiz success by X%" with details.
- *Policy Evaluation Report:* Essentially, a detailed section in the evaluation doc describing the reward function design, the values chosen for weights, and showing key results like "policy chooses shorter content in 70% of cases where baseline chose longer, and those cases had better outcomes Y". It will also list any simulations done (maybe a synthetic student simulation and how the bandit adapted).
- *Safety Tests Report for RL:* We'll specifically report on exploration safety – e.g., verify that even with exploration the system never chose a content above length threshold (we enforce that, but we'll log it to show compliance). And that it didn't, say, starve a certain type of content completely (we'll check fairness of distribution).
- *User Feedback from Pilot:* If by M4 we run a small pilot with actual students using the adaptive version, we'll include any feedback or results (like "5 out of 5 students in pilot preferred the adaptive hints vs. static ones" or such).
- *Go/No-Go for RL Expansion:* Possibly a memo if we decide at this point whether to roll out the bandit broadly or keep it shadow mode until more data. That decision would be documented.

- **M5 Deliverables:**

- *Fine-tuned Model Artifacts:* The LoRA adapter files for each fine-tuned model (question generator, etc.), along with training scripts. We'll also include sample outputs from before vs after fine-tune for a few prompts to illustrate the change.
- *Updated Model Cards:* For each fine-tuned model, a model card describing training data, epochs, any bias evaluation, performance metrics, etc. For example, "QGen-Adapter-v1: fine-tuned on 500 QA pairs, achieves 4.5/5 avg clarity score, uses base Llama-2 13B."
- *A/B Test Report:* If we ran an A/B with and without the fine-tune for some users or content, we present the results (like a table or plot showing the differences in outcomes or feedback).
- *Expert Evaluation of v/s Baseline:* Perhaps a small panel of educators reviewed outputs from both versions blindly – we'd deliver the data and results of that (to strengthen evidence that fine-tune is better).

- *Integration Notes:* Documentation on how the new fine-tuned models are integrated into the pipeline (e.g., “we now call our local QGen model instead of GPT-4 for that step, here’s how to switch back if needed”), so ops team knows.
- *Cost/Latency Analysis:* A brief analysis showing the cost savings or latency improvement from any model switch. E.g., “Grader: moved from API costing $0.03 per call to local model costing essentially $0, and latency down from 2s to 0.5s.”
- *M5 Acceptance Checklist:* Listing target improvements (like rubric scores improved, etc.) and confirming each.

- **M6 Deliverables:**

- *Complete System Documentation:* This includes architecture diagram, instructions to deploy (in staging/production environment), and an **API spec** if we have endpoints. Also, an end-user documentation draft (help guide for students/instructors) if relevant.
- *Safety & Compliance Report:* A formal report enumerating safety tests done, biases checked, privacy measures, and compliance mapping (like “we comply with FERPA by doing X, Y, Z”). This is for stakeholders/trust (could be needed if we’re going to deploy in a school district, for example).
- *Telemetry Dashboard & Alerts:* We will provide access (or screenshots) of the live dashboard we set up showing key metrics. Also a document describing what alerts are configured (e.g., “if hallucination rate >2% in 1 day, send email to team” or “if any content above length threshold is delivered, log it as error”).
- *Continuous Learning Plan:* A short document describing how we plan to retrain/calibrate on an ongoing basis – e.g., “IRT parameters will be updated monthly with a rolling window of recent data; bandit policy retrains weekly with latest log, using hyperparameter XYZ; new content added will be indexed daily.” Essentially our maintenance plan so it’s clear this isn’t one-and-done.
- *Final Evaluation Report:* A comprehensive report collating all key results, especially focusing on learning outcomes. If we have results from a controlled study or pilot showing learning gains, this will be highlighted. We’ll also include student engagement metrics here for completeness (though not optimizing for them, stakeholders will want to see usage stats, etc.).
- *Production Readiness Checklist:* Ensuring we tick off things like load testing done, failover in place (e.g., if one model fails, do we degrade gracefully?), monitoring set, team on-call schedule prepared, etc.
- *Go-Live Launch Memo:* A memo summarizing that we have achieved the objectives of Phase 1, and recommending go-live (or further limited beta if that’s our plan), plus any last-minute risk assessment.

Each milestone’s deliverables ensure that by the end we have not just code, but understanding and evidence. Notably, the *model cards* and *data cards* ensure we document what data went into models (for transparency, crucial in education for trust).

The *evaluation reports* at each stage keep the focus on learning efficacy. By delivering things like prompt libraries, we ensure reproducibility – someone else could take our prompts and re-create the behavior (important if migrating to a new LLM, etc.).

By the end of M6, the collection of deliverables constitutes a full technical package for the system: code, documentation, evaluation, and governance artifacts – everything needed to maintain and extend the system beyond the initial development cycle.

In addition, throughout development we will maintain a **task tracking board** (like Jira or Trello) which is not exactly a deliverable to users but an internal living document of tasks, bugs, improvements aligned with these milestones – so we can trace any feature to when/why it was done (useful for later audits or iterations).

Finally, a **"First 14 Days" Task List** as requested:

First 14 Days Task List:

Week 1: 1. **Project Setup & Content Ingestion:** Assemble the content corpus. Write scripts to ingest and chunk videos (via transcripts) and PDFs. Validate chunk sizes (target ~300-500 tokens with overlap)[17]. Index in a simple vector store (e.g., FAISS). 2. **Baseline Retrieval Implementation:** Implement BM25 and embedding search over the corpus. Test with a few example queries to ensure we get reasonable results. 3. **LLM Q&A Prompting (Initial):** Write an initial prompt for the LLM to answer questions with provided context. Test it by manually providing a snippet and seeing if it uses it. 4. **Evaluation Dataset Prep:** Draft a small set of QA pairs (maybe 20) covering various topics to use as a baseline test. 5. **Metric Calculation Code:** Write code to compute nDCG@3, Recall@3 using the small test set for retrieval results. 6. **Red-Team Planning:** Brainstorm some potentially problematic queries (nonsense, outside scope, etc.) for later safety tests. 7. **Meeting with Educators:** Quick sync with a few teachers to define what a "good" answer resource looks like (to inform our rubric for minimality etc.).

Week 2: 8. **Prototype End-to-End (M1):** Integrate retrieval with LLM: input a question, retrieve top 1 snippet, feed to LLM, get answer. Build this in a notebook for clarity. 9. **Content Length Filter:** Implement a simple check: if snippet length > threshold, truncate or mark it. For now, maybe just note it in output. 10. **Run Baseline Evaluation:** Use the test queries on the prototype. Compute retrieval metrics, note any overlong outputs, and qualitatively assess answer correctness. 11. **Iteration on Retrieval Parameters:** If recall is low, adjust embedding model or chunking parameters. Re-run until baseline metrics reach acceptable ballpark (not all acceptance criteria, but reasonable). 12. **Document M1 Results:** Compile the Evaluation Report for M1 (with metrics and two or three example Q&A outputs). 13. **Plan M2 Work in Detail:** Break down tasks for cross-encoder integration (find a suitable model, e.g. "cross-encoder/ms-marco-MiniLM"), and video segmentation approach (maybe using YouTube timestamps or a simple text splitting). 14. **Team Review & Handoff:** Present the M1 prototype to team/stakeholders for feedback. If acceptance criteria not fully met, identify quick fixes (maybe add a second retrieved snippet, etc.) and schedule those in early Week 3 before diving into M2.

This two-week plan gets us through Milestone 1 with a functioning RAG baseline and initial evaluation, establishing a foundation to build upon.

---

[1] Reducing Hallucination in Structured Outputs via RAG | Prompt Engineering Guide

https://www.promptingguide.ai/research/rag_hallucinations

[2] [10] The aRt of RAG Part 3: Reranking with Cross Encoders | by Ross Ashman (PhD) | Medium

https://medium.com/@rossashman/the-art-of-rag-part-3-reranking-with-cross-encoders-688a16b64669

[3] Just How Micro Is Microlearning?

https://www.td.org/content/atd-blog/just-how-micro-is-microlearning

[4] [5] [11] [30] Efficient Fine-tuning with PEFT and LoRA | Niklas Heidloff

https://heidloff.net/article/efficient-fine-tuning-lora/

[6] Microlearning Videos: Finding the Ideal Length

https://www.compozer.com/post/how-long-should-microlearning-videos-be

[7] Multi-objective contextual bandits in recommendation systems for smart tourism | Scientific Reports

https://www.nature.com/articles/s41598-025-89920-2?error=cookies_not_supported&code=05024641-61ce-4f42-982a-043c5a79e655

[8] [26] [27] Standardized assessments developed by physics education researchers can help tell an important story: Why I use these assessments in my own classes | Bridge to Tomorrow

https://bridgetotomorrow.wordpress.com/2022/04/15/standardized-assessments-developed-by-physics-education-researchers-can-help-tell-an-important-story-why-i-use-these-assessments-in-my-own-classes/

[9] [12] [28] [29] people.umass.edu

https://people.umass.edu/~andrewlan/papers/16edm-bandits.pdf

[13] [31] Sequencing Educational Content Using Diversity Aware Bandits

https://educationaldatamining.org/EDM2023/proceedings/2023.EDM-posters.57/index.html

[14] Reinforcement Learning for Recommendation Systems

https://applyingml.com/resources/rl-for-recsys/

[15] Exploring Automatic Short Answer Grading as a Tool to Assist in ...

https://pmc.ncbi.nlm.nih.gov/articles/PMC7334737/

[16] [21] [25] [2408.04394] Automated Educational Question Generation at Different Bloom's Skill Levels using Large Language Models: Strategies and Evaluation

https://ar5iv.labs.arxiv.org/html/2408.04394v1

[17] Evaluating Chunking Strategies for Retrieval | Chroma Research

https://research.trychroma.com/evaluating-chunking

[18] Building a Smarter RAG Application with Reranking: An End-to-End Tutorial Using a Safaricom Use Case | by Herman Wandabwa | Data Science Collective | Medium

https://medium.com/data-science-collective/building-a-rag-system-with-mmr-for-safaricoms-smart-assistant-1e9ba91b9bfe

[19] [20] new.videopoints.org

https://new.videopoints.org/public/vpresearch/resources/papers/Topic_Based_Segmentation_of_Classroom_Videos.pdf

[22] Self-Consistency | Prompt Engineering Guide

https://www.promptingguide.ai/techniques/consistency

[23] [24] Understanding Item Response Theory

https://www.edisonos.com/online-teaching/understanding-item-response-theory