perplexity

LLM Research & Analysis Prompt

Project Context: We are developing an educational application. The current AI logic for lesson progression, answer validation, and user state management is failing. Core issues include:

1. **Inconsistent Correction:** Failure to reliably validate user answers and provide corrective feedback.
2. **State Management Failures:** Inappropriate rollbacks (e.g., reverting to Lesson 1) and failure to correctly track user progress.
3. **Unpredictable Lesson Flow:** Poorly designed logic for branching, sequencing, and adapting content based on user performance.

Your Objective: Act as a research and systems analysis consultant. Your goal is to provide a comprehensive reference on functional architectures and logic patterns for educational applications. This reference will be used to diagnose flaws in our current system and design a more robust, predictable, and educationally sound architecture.

Research Tasks:

1. Analyze Core Educational Logic Patterns:

- Investigate and explain well-established pedagogical models for digital systems (e.g., **Mastery Learning**, **Adaptive Learning**, **Spaced Repetition**, **Microlearning**).
- Focus on the **concrete system logic** behind each model. For example:
  - **Mastery Learning:** *What are the exact criteria for "mastery"? How many consecutive correct answers? At what difficulty? How does the system handle failure to master? What is the fallback path?*
  - **Adaptive Learning:** *Based on what data points (response time, accuracy, confidence) does the system decide the next item? Describe a simple decision matrix or algorithm.*

2. Deconstruct System Architecture Components:

- Map the essential **stateful entities** and their relationships in a typical learning app. Provide a clear model for:
  - **User Profile & Progress State:** How is progress persisted? (e.g., `{user_id, skill_id, current_lesson, mastery_score, last_accessed}`).
  - **Lesson & Content Graph:** How are lessons/modules structured? Is it a linear sequence, a graph, or a tree? How are prerequisites and dependencies defined in the data schema?

- **Assessment Logic Module:** The isolated component responsible for evaluating inputs. What are its inputs (user_answer, question, acceptable_variants, hints) and outputs (is_correct, feedback, confidence_score)?
- **Session Manager:** The component that holds the user's state *during* a learning session and orchestrates calls to the Assessment Logic and Content Graph.

3. Investigate Robust Answer Validation Strategies:

- Research and summarize technical approaches for evaluating open-ended and structured responses beyond exact string matching.
  - Techniques: **Semantic similarity** (using embeddings), **keyword extraction**, **mathematical equivalence evaluation**, **parsing for conceptual understanding**.
  - **Rule-based feedback generation:** How to map specific error types to constructive feedback messages stored in a database.

4. Provide Reference Implementations & Case Studies:

- Find and summarize **publicly documented architectures** from reputable educational platforms (e.g., **Khan Academy**, **Duolingo**, **Coursera**, **Brilliant**). Focus on their technical blog posts or engineering talks.
- Analyze **open-source educational tools** (e.g., **Moodle**, **Oppia**, **Adapt**). Describe their data models for tracking and their question/answer engine design.

Deliverable Format:
Please structure your response as a **Technical Reference Document** with the following sections:

1. **Executive Summary:** Brief diagnosis of likely root causes for our stated problems based on common pitfalls.
2. **Core Pedagogical Logic Patterns:** A table comparing models, their logic rules, and best-use cases.
3. **Recommended System Architecture Blueprint:** A proposed component diagram (describe in text) with clear data flow (e.g., `User Input -> Session Manager -> Assessment Engine -> Progress Update -> Next Item Decision`).
4. **State Management Model:** A proposed schema for key databases/entities to prevent rollbacks and track progress reliably.
5. **Answer Validation Framework:** A decision tree or flowchart for how the `Assessment Logic Module` should process a user's answer.
6. **References & Further Reading:** Links to technical articles, research papers, and open-source repositories that serve as concrete examples.

Goal: The output must be **actionable, specific, and technical**. It should provide our engineering and product teams with a clear roadmap to audit our current logic and implement a stable, scalable, and educationally effective system.

A robust learning app needs (1) deterministic state transitions, (2) a dedicated assessment/validation service with typed outputs, and (3) a content graph + policy engine that decides “what’s next” from persisted mastery signals rather than ad-hoc chat context. The

symptoms described (inconsistent correction, rollbacks, unpredictable branching) are classic outcomes of mixing evaluation + progression inside an LLM without a strict state machine and immutable attempt records.

Executive summary

Common root causes behind "random rollbacks" and inconsistent feedback are: non-idempotent updates (recomputing progress from scratch each turn), missing append-only attempt logs, and letting the LLM both *judge* and *mutate state* without guardrails.

Typical failure mode: the system stores only "current_lesson" and overwrites it on each turn, so any mis-parse or LLM hallucination can reset the pointer; the fix is to persist **events** (attempts, decisions) and derive the current view from them, while also storing a canonical "current node" with monotonic constraints (can't go backwards unless an explicit remediation policy fires).

A second frequent cause is answer checking that depends on a single brittle method (exact match or single-shot LLM judge) rather than a layered validator that can return: `correct`, `incorrect_known_misconception`, `needs_clarification`, or `ungradable`, with deterministic handling for each.

Core pedagogical logic patterns

Below is a "system-logic" comparison (rules you can implement without guessing).

Model	What to persist	Promotion rule (example)	Failure/remediation rule (example)	Best use
Mastery learning	`mastery_score`, `streak`, `attempt_history`, `objective_ids_mastered`	Promote when score ≥ threshold (e.g., 0.85) **and** last N items include K correct at target difficulty	Assign targeted practice on failed objectives; allow multiple attempts and re-quiz variants (Coursera highlights multiple attempts + practice as part of mastery-style design) [1] [2]	Foundational skills, compliance training, math/grammar
Adaptive learning	`skill_mastery` per skill, item params, time-on-task, hint usage	Select next item by estimated mastery and uncertainty (e.g., BKT/IRT-style approaches are commonly referenced as adaptive frameworks) [3]	If repeated failure, reduce difficulty, add scaffold/hints, or switch to prerequisite skill	Mixed-ability cohorts, personalized pacing

Model	What to persist	Promotion rule (example)	Failure/remediation rule (example)	Best use
Spaced repetition	per item: `stability/half-life`, `last_seen`, `recall_prob`	Schedule review when predicted recall drops below target	If failed recall, shorten interval and adjust model parameters (Duolingo's Half-Life Regression models item "half-life" to predict recall vs time-lag) [4] [5]	Vocabulary, facts, definitions, formula recall
Microlearning	`module_completion`, `objective_tags`, `session_context`	Promote when micro-objective complete + quick check passed	If failed check, replay micro-lesson + one targeted item	Mobile-first, low attention windows, onboarding
Spiral review (hybrid)	`mastery_level` + `time_since_last_practice`	Periodically mix older skills into new practice	Level up/down based on short review sets (Khan Academy "Mastery Challenges" are a review mechanism with fixed-size sets and level-up/down rules) [6]	Prevent forgetting while progressing

Implementation note: Khan Academy's published Mastery Challenge mechanics include a 6-question challenge reviewing 3 skills (2 questions per skill) with deterministic level-up/down rules based on both answers for each skill. Their mastery levels are described as progressing through named levels (e.g., Familiar → Proficient → Mastered).[6] [7] [8]

Recommended architecture blueprint

Goal: keep the LLM out of authoritative state mutation; use it for explanation, hints, and (optionally) "grader-of-last-resort" with tight schemas.

Components (text "diagram")

- **Client (Web/Mobile)**
  - Sends: `{session_id, user_id, activity_id, user_answer, ui_lang, metadata}`
- **Session Manager (orchestrator)**
  - Owns in-session state cache; never invents progress.
  - Calls:
    1. Content Graph to fetch the current node + allowed transitions
    2. Assessment Engine to grade/diagnose the response
    3. Progress Service to write immutable attempt + update mastery aggregates

- **Assessment Engine (isolated, typed I/O)**
  - Input: `{question_id, question_type, rubric, acceptable_variants, user_answer, locale}`
  - Output: `{verdict, score_0_1, misconception_code?, feedback_key?, confidence, extracted_answer?, needs_human_review?}`
- **Progress Service (source of truth)**
  - Writes append-only **AttemptEvent**
  - Updates derived tables (mastery aggregates, "current node pointer")
- **Content Graph Service**
  - Stores lessons/modules as a DAG (recommended) with prerequisites.
  - Returns candidate next nodes given policy constraints.
- **Policy/Next-Item Decision Engine**
  - Pure function: `f(user_state, last_assessment, graph_candidates) -> next_activity + rationale_code`
- **LLM Tutor (non-authoritative)**
  - Consumes: `{question, rubric_summary, misconception_code, user_answer}`
  - Produces: explanation/hints, but cannot change progression.

Data flow (deterministic)

```
User Input -> Session Manager -> Assessment Engine -> Progress Service (AttemptEvent write) -> Policy Engine -> Content Graph -> Session Manager -> Next Item
```

This separation mirrors mature LMS/quiz engines: Moodle's core question engine explicitly orchestrates question attempts/behaviours and handles storing progress in the database behind a uniform API. Moodle also describes the control flow "question engine → behaviour → question type," emphasizing isolation between orchestration and question-type logic.[9] [10]

State management model

Design principle: **event-sourcing for attempts + monotonic progress pointers**.

Core entities (minimum viable)

- `users`
  - `user_id (pk)`, profile fields, locale, tz
- `skills`
  - `skill_id (pk)`, metadata/tags
- `content_nodes`
  - `node_id (pk)`, type (`lesson|practice|quiz|review`), `skill_targets[]`
- `content_edges`
  - `(from_node_id, to_node_id, condition_code, prerequisite_skill_id?, min_mastery?)`

- `attempt_events` (append-only)
  - `attempt_id (pk)`
  - `user_id, node_id, question_id`
  - `timestamp`
  - `submitted_answer (raw)`
  - `assessment_verdict`, `score`, `confidence`
  - `misconception_code`, `feedback_key`
  - `latency_ms`, `hints_used`
  - `engine_version` (critical for reproducibility)
- `user_skill_state` (derived, updatable)
  - `user_id, skill_id`
  - `mastery_score` (0–1), `mastery_level`, `last_practiced_at`
  - `streak_correct`, `ewma_accuracy`, `review_due_at` (if spaced repetition)
- `user_path_state` (derived "pointer")
  - `user_id`
  - `current_node_id`
  - `path_version` (increments)
  - `locked_until` (optional, if you need cooldown rules)

Anti-rollback invariants (practical rules)

- **Never** set `current_node_id` directly from the LLM output.
- Update `current_node_id` only via Policy Engine result, and only if:
  1. The decision references an existing `attempt_id`
  2. The transition exists in `content_edges`
  3. The transition passes prerequisite checks
- Allow backward moves only through explicit remediation transitions (e.g., `condition_code = "REMEDIATE_PREREQ_X"`), not by resetting to a default lesson.

Concurrency and idempotency

- Make `attempt_events` writes idempotent by accepting a client-generated `submission_uuid`.
- Use optimistic locking on `user_path_state.path_version` so out-of-order requests cannot revert progress.

Answer validation framework (decision tree + references)

Decision tree (implementable)

1. **Normalize input**
   - Locale-aware normalization (casefold, punctuation, whitespace), tokenization.
   - If the question expects a structured response: parse first (JSON, multiple-choice ID, numeric, equation AST).
2. **Fast deterministic checks (by type)**
   - Multiple choice: exact option ID match.
   - Numeric: parse number + tolerance/units.
   - Math equivalence: parse to AST/CAS simplification (if available); if equivalent → correct.
3. **Rule-based pattern checks (diagnostics)**
   - If matches known misconception patterns → `incorrect_known_misconception` + `misconception_code`.
4. **Semantic grading (short answer / open response)**
   - Compute embeddings similarity against 1..N reference answers; combine with keyword constraints ("must mention X").
   - Research on automated short-answer grading often combines semantic similarity via embeddings with syntactic analysis to score responses beyond exact match.[11] [12]
5. **LLM judge (bounded)**
   - Only if semantic score falls in an "uncertain band" (e.g., 0.55–0.75) or parsing fails.
   - LLM must output a strict JSON schema: `{verdict, score, rationale_code, extracted_answer}`.
6. **Post-processing + safety**
   - If `confidence < min_confidence`: return `needs_clarification` (ask a follow-up) or `needs_human_review`.
7. **Feedback selection**
   - Map `(question_id, misconception_code, locale)` → `feedback_template_id` stored in DB.
   - Render feedback; optionally ask LLM to rewrite *only the tone/phrasing* of an already-selected template.

Why this fixes "inconsistent correction"

- Deterministic layers catch most cases and produce repeatable outcomes.
- The LLM becomes a controlled fallback, not the first (or only) grader.
- Storing `engine_version` and rubric hashes makes grading auditable and reproducible.

Concrete platform references (useful patterns to copy)

- **Duolingo (spaced repetition):** Half-Life Regression models an item's memory "half-life" to predict recall probability as a function of time lag and practice history, enabling data-driven scheduling.[4] [5]
- **Khan Academy (mastery + spiral review):** Mastery Challenges are unlocked under explicit conditions and use deterministic level-up/down rules based on answers in a fixed question set; Khan also documents mastery levels as discrete states (Attempted/Familiar/Proficient/Mastered).[7] [8] [6]
- **Coursera (mastery-style attempts + feedback):** Coursera discusses allowing multiple quiz attempts and using practice + benchmarks (high quiz score) as part of mastery learning in course design.[1] [2]
- **Moodle (robust assessment engine):** Moodle documents a "core question engine" that orchestrates question behaviours and question types while handling storage of attempt progress in the database, and explains the key flow of "question engine → behaviour → question type".[10] [9]
- **Adapt (tracking state via xAPI):** The Adapt xAPI plugin exposes concepts like sending statements (attempted/passed/completed, etc.) and optionally letting an LRS manage course state via the xAPI State API.[13]

Further reading (starter set)

- Duolingo HLR paper (ACL): "A Trainable Spaced Repetition Model for Language Learning"[4]
- Duolingo HLR code: `duolingo/halflife-regression`[5]
- Khan Academy Mastery Challenges mechanics and mastery levels overview[8] [6]
- Coursera mastery learning in course design (practice + feedback)[2]
- Moodle Question Engine / Questions API and architecture notes[9] [10]
- Automated short-answer grading with semantic similarity/embeddings[12] [11]
- Adapt xAPI plugin (state + statements)[13]

If the current system can share a redacted sample of: (1) your stored "progress" records, (2) a lesson graph representation, and (3) a few real answer/feedback traces, a precise audit checklist can be produced against the blueprint above.

⁂

1. https://blog.coursera.org/5-tips-learn-more-effectively-in-class-with/
2. https://blog.coursera.org/how-to-integrate-mastery-learning-into-course-design/
3. https://learn.g2.com/adaptive-learning
4. https://research.duolingo.com/papers/settles.acl16.pdf
5. https://github.com/duolingo/halflife-regression

6. https://support.khanacademy.org/hc/en-us/articles/360037127892-What-are-Mastery-Challenges-in-course-mastery
7. https://blog.khanacademy.org/why-khan-academy-will-be-using-skills-to-proficient-to-measure-learning-outcomes/
8. https://support.khanacademy.org/hc/en-us/articles/5548760867853--How-do-Khan-Academy-s-Mastery-levels-work
9. https://tjhunt.blogspot.com/2010/10/new-question-engine-how-it-works.html
10. https://moodledev.io/docs/5.0/apis/subsystems/question
11. https://doaj.org/article/1e97df8b626d4fc09cd56ef9c3411463
12. https://ijtech.eng.ui.ac.id/download/article/4651
13. https://github.com/adaptlearning/adapt-contrib-xapi
14. https://www.coursera.support/s/question/0D51U00003BlYzDSAV/check-number-of-attempts-on-past-quizzes?language=en_US
15. https://blog.coursera.org/ai-grading-in-peer-reviews-enhancing-courseras-learning-experience-with-faster-high-quality-feedback/
16. https://blog.coursera.org/category/educators/page/13/
17. https://blog.coursera.org/coursera-launches-academic-integrity-features-to-help-universities-verify-learning-in-an-age-of-ai-assisted-cheating/
18. https://pmc.ncbi.nlm.nih.gov/articles/PMC12453766/
19. https://ijtech.eng.ui.ac.id/article/view/4651
20. https://blog.coursera.org/coursera-coach-leveraging-genai-to-empower-learners/
21. https://brilliant.org
22. https://www.facebook.com/Coursera/posts/read-about-5-tips-from-our-coursera-pedagogy-team-on-how-to-learn-more-effective/391632427621418/
23. https://brilliantlearningsystems.com
24. https://educationaldatamining.org/edm2025/proceedings/2025.EDM.short-papers.124/2025.EDM.short-papers.124.pdf
25. https://www.coursera.support/s/question/0D51U00003KTuOzSAL/how-many-times-can-we-attempt-the-graded-questions-quizs-is-there-any-maximum-attempt-count-curious-to-increase-the-grading-over-time?language=en_US
26. https://brilliant.org/about/
27. https://ieeexplore.ieee.org/abstract/document/10690067/
28. https://zerotomastery.io/blog/udemy-vs-coursera-vs-zero-to-mastery/
29. https://www.reddit.com/r/MachineLearning/comments/5icbpj/d_duolingos_halflife_regression_method_for/
30. https://www.repository.cam.ac.uk/bitstream/1810/305124/1/Adaptive_Forgetting_Curve_for_Spaced_Repetition_Language_Learning_.pdf
31. https://research.duolingo.com
32. https://github.com/oppia/oppia/wiki/Creating-Interactions
33. https://pmc.ncbi.nlm.nih.gov/articles/PMC6410796/
34. https://www.khanacademy.org/khan-for-educators/resources/teacher-essentials/safety-privacy-and-additional-resources/a/mastery-challenges-course-mastery

35. https://github.com/oppia/oppia/wiki/Tutorial-Learn-how-to-write-a-TDD
36. https://repositori-api.upf.edu/api/core/bitstreams/657b96a3-acab-48f2-9106-2055394ef712/content
37. https://www.youtube.com/watch?v=el8lUMrbHvk
38. https://www.scribd.com/document/954895206/Oppia-2020-Nishant-Mittal
39. https://escholarship.org/content/qt05t949b2/qt05t949b2_noSplash_83cfc3562493152e67742f7d0c92466b.pdf
40. https://www.youtube.com/watch?v=YUS8ggPkqZg
41. https://docs.openedx.org/en/open-release-sumac.master/educators/references/course_development/exercise_tools/oppia_exploration.html
42. https://www.politesi.polimi.it/retrieve/b39227dd-0963-40f2-a44b-624f205cb224/2022_4_Randazzo_01.pdf
43. https://www.cultofpedagogy.com/khan-mastery-learning/
44. https://oppia.github.io
45. https://papousek.github.io/analysis-of-half-life-regression-model-made-by-duolingo.html
46. https://www.open.edu/openlearncreate/mod/oucontent/view.php?id=52747&printable=1
47. https://teachinghub.bath.ac.uk/guide/how-to-set-up-a-moodle-quiz/
48. https://fossies.org/linux/moodle/public/question/engine/questionattempt.php
49. https://rsisinternational.org/Issue7/306-309.pdf
50. https://www.adaptlearning.org/index.php/plugin-browser/
51. https://blog.khanacademy.org/khan-academy-efficacy-results-november-2024/
52. https://wimski.org/api/3.8/dc/d34/classquestion__engine__unit__of__work.html
53. https://xapi.com/adopters/
54. https://phpdoc.moodledev.io/4.1/da/d46/classquestion__engine__data__mapper.html
55. https://www.adaptlearning.org/index.php/adapt-framework/
56. https://support.khanacademy.org/hc/en-us/articles/115002552631-What-are-Course-and-Unit-Mastery
57. http://xref-diff.mukudu-dev.net/moodle403/question/engine/diff_310-403__questionattempt.php.html
58. https://adapt.tips/xapi-guide/
59. https://www.khanacademy.org/khan-for-educators/k4e-us-demo/xb78db74671c953a7:getting-to-know-khan/xb78db74671c953a7:introduction-to-mastery-learning/v/course-mastery
60. https://phpdoc.moodledev.io/4.3/da/d46/classquestion__engine__data__mapper.html