# Real-Time LLM-Powered Psychology Tutor: System Design Guide

This guide outlines the design of an AI-driven tutoring system for undergraduate psychology. It covers the full stack architecture, how to structure psychology content into a skill map, and the adaptive logic for questioning, grading, feedback, and scheduling. The focus is on building a **real-time tutor** that generates questions on the fly, tracks detailed user progress (accuracy, timing, misconceptions, mastery), and adapts to the learner. We emphasize clear structure and educational best-practices so a developer can implement the system (no code needed, just design and logic).

## System Architecture Overview

A high-level architecture for the LLM-powered tutor consists of a **frontend**, **backend**, **database**, and **LLM integration** component, plus supporting services for performance and scalability:

* **Frontend:** The user interface (web or mobile) where students interact with the tutor. It can be a chat-style UI or a quiz interface that displays questions (multiple-choice or short-answer input) and shows feedback. For example, a web app could present a chat conversation or quiz screen for each lecture or topic[[1]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=1,specified%20for%20the%20lecture). The frontend handles user login, question display, answer input (selection or text), and shows explanations/next questions. It communicates with the backend via API calls (e.g., REST or WebSocket for real-time updates).
* **Backend:** The server side application that contains the logic of the tutor. This can be built with a web framework (for instance, the CS Tutor system used a Django backend to manage sessions, format prompts, and handle authentication[[2]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=2,for%20a%20better%20user%20experience)). The backend receives user actions (e.g. answer submitted), then uses the LLM and the user model to decide on feedback and next questions. It orchestrates calls to the LLM (for question generation or grading), applies business rules (like updating mastery), and interfaces with the database. This layer may also include **middleware** for caching and asynchronous tasks. For example, expensive operations like generating a batch of quiz questions or analyzing a PDF can be offloaded to a task queue (as done with Celery in the CS Tutor project[[3]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=,and%20Q%26A%20about%20the%20document)). A caching layer (e.g. Redis) can speed up responses by storing recent Q&A pairs; repeated or similar queries could retrieve a cached result instead of querying the LLM again[[4]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=6,hit%20and%20uses%20the%20answer). However, caching should be carefully managed for dynamic, personalized questions (the tutor mostly generates new questions each time, so caching is more relevant for repeated explanatory queries or reused content).
* **Database:** A persistent store for user data, progress, and content. A relational database (like PostgreSQL) is suitable to store structured information: user profiles, the psychology skill/topic map, question logs, and performance history[[5]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=Main%20chat%20screen%20). The database will have tables for users, skills (topics), questions (or at least records of generated questions and answers), and user interactions (responses, scores, timestamps). By logging all interactions, the system can analyze progress over time and personalize the tutoring. The database also stores any static content, like a library of common misconceptions or a bank of example questions if using them. All sensitive data should be stored securely (with encryption for personal info) – more on privacy below.
* **LLM Integration:** The core AI component is the Large Language Model that generates questions, evaluates answers, and provides explanations. The LLM can be accessed via an API (e.g., OpenAI GPT-4) or hosted on-premise. In either case, the backend formats prompts and sends them to the LLM, then receives the output and interprets it. Integration considerations include model choice (balance quality vs. latency), prompt design (to get the desired output format), and rate-limiting or cost control. In a real-time tutor, latency is critical – calls to the LLM should be optimized (possibly using smaller models for simpler tasks or caching results of previous calls). Some designs use a local model for speed; for example, one tutor system integrated a local LLM (Gemini 1B via Ollama) for conceptual explanations and quiz generation[[6]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=3,and%20Q%26A%20about%20the%20document). In other cases, one might use a cloud API with careful prompt engineering and possibly few-shot examples to ensure the LLM’s outputs are of *exam-quality*. The backend might also incorporate a vector database or knowledge base for retrieval-augmented generation if needed (e.g. to ground questions in specific textbook content), but for a well-covered domain like undergraduate psychology, a sufficiently advanced model may handle it with its internal knowledge.
* **Additional components:** For a production system, other elements support the core. Containerization (Docker/Kubernetes) can be used to deploy the services in a scalable way[[7]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=7,or%20ease%20in%20usage). Logging and monitoring services will track system performance and errors. If speech or multimedia are involved, additional modules would handle those (not our focus here). Also, content management interfaces might be provided for educators to review or input questions/objectives, but in this design the LLM generates content dynamically so content management is more about maintaining the curriculum structure (the topic map and any curated materials).

**Workflow Summary:** The typical request flow is: the frontend sends the backend a request (e.g., “user answered question X with choice B”). The backend checks the answer (possibly calling the LLM for grading if it’s a short text answer), updates the user’s mastery and progress in the database, then uses the LLM to generate the next question (prompted by the relevant skill and desired difficulty). This next question and any feedback are returned to the frontend to display to the user. This loop continues in real-time for the tutoring session. Fast database lookups and occasional cache use ensure responsiveness, while heavier LLM calls happen asynchronously if needed to keep the UI snappy. For instance, one architecture routes simple queries directly to the LLM, but offloads longer operations to a task queue and checks a cache before calling the model[[8]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=7,or%20ease%20in%20usage).

## Psychology Topic Map (Structured Learning Objectives)

To drive an adaptive tutor, we need to represent the *curriculum* of psychology as a structured **topic map** or skill graph. This map breaks down the subject into hierarchical **learning objectives** and skills that the system can track and teach individually. In other words, it’s an **expert model** of the domain knowledge, organizing what an undergraduate psychology student should learn.

**Organizing the Content Domains:** At the top level, psychology can be divided into major domains commonly taught in an undergraduate program. For example, a typical introductory psychology course (or series of courses) would include domains such as **biological psychology**, **cognitive psychology**, **developmental psychology**, **social psychology**, **personality and individual differences**, **abnormal psychology (clinical disorders and treatment)**, and **research methods**. Each of these broad areas then breaks into finer topics. For instance, *Cognitive psychology* can include subtopics like memory, learning theories, perception, and language; *Developmental psychology* might split into cognitive development, social development, etc.; *Social psychology* includes topics like conformity, group behavior, and attitudes; *Abnormal psychology* covers various disorders (anxiety, mood, schizophrenia, etc.) and therapeutic approaches. A course alignment document might list these as *“major content domains of psychology”* such as social, developmental, learning, memory, sensation & perception, cognitive, personality, and abnormal psychology[[9]](https://www.coursesidekick.com/psychology/2663557#:~:text=between%20the%20two%20on%20behavior,Students%20will%20applycritical%20thinking%20skills). These will serve as the high-level modules in the tutor’s curriculum.

**Skills and Subskills:** Within each topic, we identify granular *knowledge components* or skills that can be individually assessed. A **concept map** approach is useful: each domain is broken down into key concepts or learning objectives, which can be further subdivided into sub-concepts. For example, under *Memory* (a cognitive psychology topic), subskills might include distinguishing short-term vs. long-term memory, understanding working memory model, knowing types of memory (episodic, semantic, procedural), etc. Each of these is a specific learning objective the student should master. The tutor’s content map should list all these fine-grained skills, which are sometimes called **knowledge components (KCs)** in intelligent tutoring systems[[10]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=the%20probabilistic%20and%20deep%20learning,intuitive%20notions%20of%20ideas%20that). The **granularity** should be such that one question can reasonably test a component. If a skill is too broad (“understand all of cognitive psychology”), it’s not useful for targeted questioning; if too narrow (“recall the exact date of a specific experiment”), it may be trivial. The idea is to have *relatively fine-grained subskills* that together cover the whole domain[[11]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=The%20concept%20map%20or%20otherwise,the%20exercise%20instructs%20and%20assesses).

**Prerequisites and Relationships:** The topic map isn’t just a list – it should encode relationships between concepts. Some skills are **prerequisites** for others. For example, understanding classical and operant conditioning is a prerequisite for more advanced topics in learning or behavior therapy. Or knowing the parts of a neuron and their functions is prerequisite to understanding how neurotransmitters affect behavior (biopsychology). We can represent this structure as a directed graph or a hierarchy: nodes are concepts, and edges indicate a prerequisite or dependency. This allows the tutor to maintain a **learning path** – typically a student should master foundational topics before moving to more complex ones. In psychology, many topics are interrelated but not strictly linear; the map might be a web. However, for implementation, we can impose a sensible ordering (e.g., cover research methods early so that later topics can reference study findings, or cover basic biology before cognitive topics). The skill map might also group concepts by difficulty level or Bloom’s taxonomy level. For example, *basic recall* of a definition is one level; *application* of a concept to a scenario is a higher level. We might label objectives with their expected cognitive level so the question generator can adjust question complexity accordingly[[12]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20for%20generating%20MCQs,levels%20of%20Bloom%E2%80%99s%20Revised%20Taxonomy).

**Use in the System:** This topic map is stored in the database (e.g., a table for Topics/Skills with fields like id, name, parent\_topic\_id, difficulty\_level, description). Each question generated will be tagged to one or more of these skills. The user’s mastery will be tracked per skill (detailed in the next section). By having the map, the system knows *where to go next*: if a student masters one subtopic, the tutor can suggest the next one (perhaps unlocking the next “branch” of the map). This is analogous to a **skill tree** in Duolingo or other learning apps, where you must achieve a certain level in earlier skills to progress[[13]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=2,level%20Spanish%20in%02struction%20%28Vesselinov%20and)[[14]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=one%20lesson%20remaining,1849). The topic map also helps ensure coverage: the tutor aims to help the student eventually master all the key objectives by navigating the map, rather than randomly drilling questions.

In summary, the psychology topic map is the knowledge backbone of the tutor. It **breaks down the subject matter into chunks of knowledge and skills**[[10]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=the%20probabilistic%20and%20deep%20learning,intuitive%20notions%20of%20ideas%20that). Each chunk is a target for questioning and mastery tracking. This structured approach allows the tutor to generate relevant questions (it knows what content to draw on), and to adaptively select the next topic based on what the student has or hasn’t mastered. It’s important to design this map in collaboration with psychology educators or based on curricula (textbook chapters, APA guidelines, etc.) so that it’s pedagogically sound.

## User Model: Mastery, Memory, Misconceptions, Personalization

The **user model** represents everything the system knows about the learner’s current state of knowledge, skills, and needs. It is updated continuously as the student interacts with the tutor. Key components of the user model include:

* **Mastery Estimates per Skill:** For each knowledge component in the topic map, the system maintains an estimate of the student’s mastery or proficiency. This can be a numeric score or probability (e.g., a value between 0 and 1 representing the probability the student will get a question on that skill correct). Initially, these can be set to a default (for a new user, perhaps 0.2 or 20% mastery assumption, or could be initialized via a diagnostic quiz). As the student answers questions, the system updates the mastery level for the relevant skill(s). If the student answered a question on, say, *Piaget’s stages of development* correctly, the mastery for that concept increases; if they answered incorrectly, it decreases. A common approach is Bayesian Knowledge Tracing (BKT), where each skill’s mastery is modeled as a probability that gets Bayesian-updated with each answer (with parameters for learning rate, guess, and slip)[[15]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=represent%20the%20available%20knowledge%20and,is%20a%20more%20recently%20developed)[[16]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=observation%20pairs%20X%20%3D%20%7B%28,Similarly%2C%20the%20objective%20of). In BKT, the model would use the sequence of correct/incorrect responses for a skill to update the probability the student *knows* that skill[[16]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=observation%20pairs%20X%20%3D%20%7B%28,Similarly%2C%20the%20objective%20of). Alternatively, a simpler heuristic approach could be used (e.g., an exponential moving average of performance on recent questions for that skill). The goal is to have a dynamic measure of how well the student knows each subtopic.
* **Memory Decay (Forgetting Curve):** Human memory is not static – mastered knowledge fades if not practiced. The user model should account for **time** since last practice by implementing a forgetting model. In practice, this means that the mastery score for a skill will **decay downward over time** if the student doesn’t review it[[17]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=knowledge%20at%20any%20given%20time,average%20probability%20that%20the%20student)[[18]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=exercises%20are%20taken%20from%20those,methods%20in%20language%20learning%20technology). Systems like Duolingo explicitly model this: they have a “strength” meter that decreases as days go by without practice, reflecting the probability of recall at that moment[[19]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=a%20key%20component%20of%20the,word%20from%20the%20lessons%20in)[[20]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=student%20model%2C%20as%20seen%20beneath,bars%2C%20the%20skill%20has%20grown). We can incorporate a similar decay function (for example, an exponential decay where mastery drops towards some baseline over a period of days/weeks). When the student reviews the material (answers a question on that skill correctly), the strength resets or increases. This is tied to spaced repetition scheduling (discussed later), but the decay is represented in the user model. The model essentially answers: *“How likely is the student to recall concept X right now?”*[[21]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=Duolingo%20uses%20strength%20meters%20to,bars%2C%20the%20skill%20has%20grown). Initially right after learning, it’s high; as months pass, it drops, unless refreshed by practice. By modeling memory this way, the tutor knows when to prompt a review question for earlier topics to reinforce them (preventing long-term forgetting).
* **Misconceptions and Error Patterns:** The user model should not only track right/wrong counts, but also identify **why** the student might be getting something wrong. In psychology, as in other subjects, learners can have specific misconceptions. For example, a common misconception might be confusing negative reinforcement with punishment, or believing that correlation implies causation. If the system can detect these, it can tailor feedback and future questions to correct them. How to detect misconceptions? One way is through **analysis of wrong answers**. In multiple-choice questions, the incorrect options (distractors) can be designed to represent specific misconceptions[[22]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding). If the student consistently picks a particular distractor, the system infers they have the misconception that distractor represents. For instance, if a question asks about a neurotransmitter’s effect and one choice is a popular myth, selecting that choice reveals the student holds that myth as true. The system would record this in the user model (e.g., a boolean flag or counter for the misconception). For short answers, the LLM’s evaluation can also pinpoint misconception patterns by comparing the student’s explanation to the correct one – the LLM might detect if the student reasoning aligns with a known incorrect theory. The system can maintain a catalog of common misconceptions for each topic to check against (this could be curated from educational research or even generated by the LLM by asking it for common misunderstandings of a concept). Storing these helps the tutor avoid presenting similar misleading choices or conversely deliberately present them to test and then correct the student’s understanding. Essentially, beyond just “mastery low or high,” the user model might say *“User tends to mix up concept A and B”* or *“User holds misconception X.”* This allows more precise remediation.
* **Personalization Factors:** Every student is unique. The user model can include personal data or preferences that influence the tutoring strategy. This might include the student’s **learning goals** (e.g., preparing for an exam vs. general learning), their **preferred learning style or pace**, and even aspects of their personality or interests if available. For example, some students might prefer more examples and analogies, others might prefer straight factual explanations – the tutor could adapt if it has that info. Recent research (like the PACE project) has looked at adjusting to learning styles and personas[[23]](https://arxiv.org/html/2502.12633v1#:~:text=significance%20of%20recognizing%20and%20adapting,develop%20individualized%20teaching%20strategies%20that)[[24]](https://arxiv.org/html/2502.12633v1#:~:text=restrict%20their%20ability%20to%20address,In). In our context, we might not have an explicit measure of learning style, but we can observe behavior: if a student often asks for clarification or seems to struggle, perhaps give more scaffolding; if they get bored with easy questions (perhaps answering all instantly and correctly), maybe increase challenge or speed up progression. Personalization might also involve **contextualizing content** to the student’s interests. For instance, if we know the student enjoys sports, the tutor’s examples or analogies could involve sports psychology. Large language models are capable of weaving in such context if instructed (e.g., “when possible, relate questions to sports”). While this is an enhancement, it can boost engagement[[24]](https://arxiv.org/html/2502.12633v1#:~:text=restrict%20their%20ability%20to%20address,In). Lastly, the user model can store any **accessibility needs** (for example, if a student needs simpler language or has a reading difficulty, the system could adjust the phrasing of questions).
* **Response History and Timing:** The model also records **how the student interacts**, not just what they answer. This includes the **response time** for each question and whether they requested hints or used any help features. Response time is an important signal: extremely fast and incorrect responses might indicate *rapid guessing* (low effort or disengagement)[[25]](https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full#:~:text=Soland%20,a%20test%20item%2C%20while%20non). If a student clicks an answer in 1 second without reading the question fully, the system might flag that attempt as a guess rather than a serious try. Research on testing behavior calls this **rapid-guessing behavior** and suggests filtering it out or addressing it because it can bias the assessment of proficiency[[26]](https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full#:~:text=Soland%20,responding%20would%20suggest%20that%20a)[[27]](https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full#:~:text=Kuhfeld%2C%202020%3B%20Wise%20et%20al,negative%20impacts%20on%20estimated%20performance). So the tutor could detect if the student consistently answers too fast and perhaps gently prompt them to slow down and think, or treat those answers differently in mastery calculations (maybe not lowering mastery as much for a fast wrong guess, assuming it was a fluke). Conversely, very long response times might indicate confusion or difficulty; the system might proactively offer a hint if it notices the student hasn’t answered for a while (assuming a mechanism to detect “stalling”). All these interaction details (time taken, attempts made if multiple tries allowed, etc.) enrich the user model’s understanding of how the student learns.

In summary, the user model is **multi-faceted**. It keeps track of *what* the student knows (mastery per skill), *how that knowledge changes over time* (practice history and forgetting), *what mistakes or misconceptions* the student has, and *individual traits or behaviors* that inform the tutoring approach. This model is continuously updated: after each question, the system recalculates mastery for the involved skills (perhaps increasing the probability of mastery if correct, or lowering if incorrect, with a larger adjustment if the question was answered confidently and wrong). The model’s data drives the adaptive decisions: which question to give next, when to review old material, how to phrase feedback, and so on. By maintaining a detailed user model, the tutor can implement **adaptive mastery-based learning**, much like a human tutor who remembers which areas a student needs to work on and which they have mastered.

## Question Generation with LLM: Dynamic MCQs and Short Answers

One of the core capabilities of the tutor is to **generate questions on-the-fly** using the LLM. We need the questions to be high-quality (akin to those an expert teacher or exam might pose) and appropriately targeted to the student’s current learning needs. The system will generate two main types of questions: **multiple-choice questions (MCQs)** and **short-answer questions**, each with potentially different prompts and logic.

**Prompt Design for Generation:** To get the LLM to produce a good question, we craft a prompt template that provides context and instructions. The prompt typically includes: - The **target skill or topic**: e.g. “Generate a question to test the concept of classical conditioning in psychology.” - The **question type**: MCQ or short-answer (the LLM needs to know the format). - The **difficulty level or cognitive level**: e.g. “an application-level question” or “an easy recall question” depending on the student’s mastery and progress. This ensures the question has the right level of challenge – not too trivial, not too advanced. As research suggests, adaptively adjusting difficulty based on performance is important for effective learning (making questions harder if the student is doing well, or easier if they struggle, to hit the sweet spot of *desirable difficulty*[[28]](https://arxiv.org/html/2507.05629v1#:~:text=Another%20challenge%20is%20to%20dynamically,more%20challenging%2C%20and%20vice%20versa)). - **Output format instructions**: for MCQ, we might ask the model to produce the question stem and a list of, say, 4 options (labelled A, B, C, D) with exactly one correct answer. We also secretly want to know which is correct (the model can be instructed to indicate the correct one in some way that the user won’t see – perhaps by outputting the answer separately or by putting an asterisk next to the correct option, which the backend can parse and remove before showing to the student). - **Style guidelines**: “Make it an exam-style question, clear and unambiguous, using professional language.” We also might instruct “avoid giving away the answer” and for MCQs “ensure the wrong choices are plausible, not silly.” If needed, include a brief context scenario to make the question more engaging (e.g., describing an experiment and asking the student to identify the independent variable).

Using such a prompt, the LLM can generate questions on any concept. Studies have shown that modern LLMs like GPT-4 can produce MCQs of comparable quality to human-written ones in many subjects[[29]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20have%20been%20utilized%20to,49). For example, an LLM was able to generate valid anatomy questions that experts rated similarly to textbook questions[[29]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20have%20been%20utilized%20to,49). LLMs can also align questions to specific learning objectives or difficulty levels given proper instruction[[30]](https://arxiv.org/html/2507.05629v1#:~:text=medical%20experts%20for%20an%20anatomy,promising%2C%20some%20generated%20MCQs%20still).

**Generating Multiple-Choice Questions:** For an MCQ, the LLM needs to provide a correct answer and several **distractors** (wrong answers). A good MCQ requires distractors that are *plausible and relevant* (so that a student can’t guess the answer just by elimination) and ideally that reflect common errors or misconceptions. Simply asking the LLM “give 3 wrong options” might yield implausible or trivial wrong answers if not guided. We should prompt the LLM to produce **high-quality distractors**. Research indicates this is a challenge but feasible: one study found that providing the model with in-context examples led to better distractors for math questions[[31]](https://arxiv.org/html/2507.05629v1#:~:text=Addressing%20the%20generation%20of%20plausible,fewer%20%E2%80%9Dnonsense%20distractors%E2%80%9D%20compared%20to), and another showed ChatGPT can produce far fewer “nonsense” distractors if guided, compared to traditional algorithms[[32]](https://arxiv.org/html/2507.05629v1#:~:text=hallucination%20and%20bias%20persisting,based%20models)[[33]](https://arxiv.org/html/2507.05629v1#:~:text=shot%20baselines%20for%20math%20MCQs,based%20models). We can explicitly instruct: *“The wrong choices should be based on common misconceptions or typical mistakes related to this topic.”* For instance, if the question is about a research method, a distractor might be a method that sounds similar but is incorrect. By doing so, we both catch misconceptions and ensure the distractors are not random. An example prompt snippet for MCQ might look like:

"Topic: Operant Conditioning (Psychology).   
Generate a 4-option multiple-choice question that tests the concept.   
One option should be correct, the others should reflect common misconceptions (e.g., confusing negative reinforcement with punishment).   
Label options A, B, C, D.   
Also provide the correct answer label at the end."

The LLM might output a question stem about, say, a scenario of a child’s behavior being reinforced, and options including the common mistake of calling negative reinforcement punishment, etc. We then parse out the correct answer (the LLM might say “Correct answer: C” which we don’t show to the user but use for grading). Including misconceptions in distractors is a known suggested improvement for LLM-generated questions[[22]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding), as it increases their pedagogical effectiveness by tempting students who have those misconceptions and allowing the tutor to diagnose them.

We also ensure the MCQ has *only one unambiguously correct answer*. Sometimes LLMs might produce ambiguous questions or multiple correct answers if the prompt isn’t specific. We test and refine the prompt with examples to minimize this. If needed, the backend can double-check the generated question by asking the LLM or using a script to verify that only one answer is correct (for example, re-query the LLM: “Which option is correct?” to see if it’s consistent).

**Generating Short-Answer Questions:** Short-answer questions require the student to type a phrase or a few sentences (not just pick an option). These are useful to test recall or understanding in the student’s own words, but they are harder to grade automatically. For generation, the LLM can produce a question prompt that asks for an explanation or definition. For instance: “In one or two sentences, explain the difference between classical conditioning and operant conditioning.” We would instruct the LLM to also provide an **expected answer or rubric** for this question (which will be used by the grading logic). This could be done by a two-step prompt or a single prompt that yields both question and answer. One approach: prompt the LLM to output in a JSON or structured format containing the question and a “model\_answer” or “solution”. For example:

"Topic: Classical vs Operant Conditioning.  
Generate a short-answer question asking the student to explain a concept or difference.  
Also, provide a brief ideal answer or key points for the answer."

The LLM might return: - Question: “What is the primary difference between classical conditioning and operant conditioning?” - Ideal answer: “Classical conditioning associates an involuntary response with a stimulus, while operant conditioning links a voluntary behavior with a consequence (reward or punishment)...” (and so on).

We will use that ideal answer as the basis for grading. If the LLM doesn’t support multi-output easily, an alternative is to first generate a question, then in a follow-up call ask the LLM for the answer to that question for use as reference. Since we constrain to undergraduate-level content, the LLM’s own knowledge should suffice to generate correct answers (though we must be cautious of any hallucination; however, factual hallucination in well-known psych concepts is less likely with top models).

**Ensuring Quality and Variety:** We should vary the question formats to keep the student engaged. The LLM can generate not only standard questions but also scenario-based questions (e.g., describing a hypothetical experiment or case study and asking which principle applies). It can generate questions at different Bloom’s levels: knowledge, comprehension, application, analysis, etc., as appropriate for the student’s progress[[12]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20for%20generating%20MCQs,levels%20of%20Bloom%E2%80%99s%20Revised%20Taxonomy). The tutor might start with basic definition questions and progress to ones requiring application of concepts to examples (e.g., given a case of a patient’s symptoms, which disorder fits?). We instruct the model accordingly.

Also, the generation can be adaptive: if a student got a question wrong, the next question could be on the *same topic* but phrased differently or easier to help them learn (a remediation question). If the student got it right easily, the next could be a bit harder or move to the next subtopic. The backend selects the target skill for the next question based on the user model, then calls the LLM to generate it.

**Speed and Caching Considerations:** Calling an LLM for every single question could be costly or slow. If latency becomes an issue, one strategy is to *pre-generate* a few questions ahead of time or during idle moments. For example, once the student starts a session on a topic, the system could ask the LLM to generate 3 questions of increasing difficulty on that topic and cache them. Then it serves them one by one without waiting each time. This pipelining improves responsiveness. Another strategy is to maintain a library of high-quality questions per skill (some pre-vetted questions) and use the LLM only for variations or new ones. However, given the power of on-demand generation, we might rely on it entirely but with careful prompt engineering to ensure consistency.

In summary, the LLM is leveraged to produce a steady stream of **exam-quality questions** tailored to the psychology curriculum. MCQs are generated with plausible distractors (avoiding trivial choices and addressing common misunderstandings)[[34]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding), and short-answer prompts are generated alongside expected answers for automated grading. The adaptivity is built into the generation process by conditioning on skill and difficulty. With a well-designed prompt template and iterative testing, the LLM can yield questions that are varied, challenging, and pedagogically sound, comparable to those a human tutor or textbook might provide[[29]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20have%20been%20utilized%20to,49).

## Grading Logic: Evaluating Answers with Accuracy and Fairness

Once questions are delivered and the student responds, the system must **grade** the responses to determine correctness (for MCQs) or give a score (for short answers), and provide the appropriate feedback. The grading logic combines straightforward rule-checking for objective questions with LLM-based evaluation for open-ended answers.

**Multiple-Choice Grading:** For MCQs, grading is mostly straightforward. The system knows which option is correct (from the question generation step). When the student selects an answer, the backend simply compares it to the correct option. If it matches, the question is marked correct; if not, it’s incorrect. There might be slight nuance if we allowed multiple-select questions (“Select all that apply”) – in such cases the logic would check if the set of selected options equals the set of correct answers, possibly awarding partial credit for partially correct selections. But assuming standard one-correct-answer MCQs, it’s a boolean right/wrong check. The system records the result (and which option the student chose, since if it’s wrong we might use that to identify a misconception tied to that specific distractor). Since MCQs are automatically graded with 100% reliability (no ambiguity), they are excellent for updating mastery – e.g., we can confidently boost the skill mastery if correct, or reduce it if wrong.

**Short-Answer Grading with LLM (Rubric-Based):** For short answers, grading is more complex. Students’ responses might be phrased in many ways, and we need to judge if they have the right idea. Here, we use the LLM in a different role: as an **assistant grader**. The recommended approach is to provide the LLM with a **reference answer or rubric** and the student’s answer, and ask it to assess the student’s answer against the reference. Essentially, we turn the grading task into a natural language comparison task for the LLM, guided by a rubric.

A robust method, inspired by research like the *SteLLA* system[[35]](https://arxiv.org/html/2501.09092v1#:~:text=the%20specific%20task%20of%20grading%2C,are%20generated%20to%20the%20user), is retrieval-augmented grading: when an instructor’s reference answer and a marking rubric are available, those can be fed into the prompt so the LLM has ground truth to compare with[[36]](https://arxiv.org/html/2501.09092v1#:~:text=%28NLP%29,an%20overall%20grade%2C%20the%20breakdown). In our automated context, we have the LLM’s own generated ideal answer (from when the question was created). That can serve as the reference answer. We can also design a simple rubric by breaking the ideal answer into key points. For example, if the question is "Explain the difference between classical and operant conditioning," the rubric might be: (a) mentions that classical conditioning involves associating two stimuli (involuntary response), (b) mentions operant conditioning involves consequences (rewards/punishments) for behavior (voluntary response), and (c) maybe an example of each or the role of the learner. We can prompt the LLM in a structured way: *Provide a score 0-2 and feedback. 2 if fully correct, 1 if partially, 0 if incorrect. The correct answer should include: [key point 1], [key point 2], etc. Now compare to the student’s answer and evaluate.* By giving these criteria, we *ground the LLM* in a specific grading rubric, reducing arbitrary judgments[[35]](https://arxiv.org/html/2501.09092v1#:~:text=the%20specific%20task%20of%20grading%2C,are%20generated%20to%20the%20user)[[37]](https://arxiv.org/html/2501.09092v1#:~:text=Our%20system%20is%20the%20first,points%20examined%20in%20the%20problem). Research has shown that LLM grading is much more reliable when using such structured, rubric-based prompts, rather than just asking “is this answer correct?”[[37]](https://arxiv.org/html/2501.09092v1#:~:text=Our%20system%20is%20the%20first,points%20examined%20in%20the%20problem). It forces the model to check for specific content.

Concretely, the backend could do: prompt = "Instructor answer: ...; Student answer: ...; Using the instructor's answer as reference, does the student answer cover the main points? Provide a grade (Correct/Partially Correct/Incorrect) and a brief justification." The LLM would then produce something like: *"Grade: Partially Correct. The student mentioned classical conditioning but did not clearly describe operant conditioning’s role of consequences..."*. The system would parse the grade (or score) and the explanation for feedback.

We should also configure the LLM for consistency here: use a deterministic setting (low temperature) so that the grading doesn’t fluctuate if asked multiple times[[38]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=,output%20in%20a%20deterministic%20way). During development, one would calibrate the rubric and possibly test the LLM on a sample of answers to ensure it aligns with human grading.

**Handling LLM Grading Challenges:** There are known concerns when using LLMs as graders. They might be biased or inconsistent, or even hallucinate flaws that aren’t there if not given a reference[[39]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=,output%20in%20a%20deterministic%20way)[[40]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=cut,also%20not%20be%20present). By always providing a reference answer/rubric, we mitigate hallucination (the model shouldn’t make up criteria beyond the rubric). We also avoid giving the model any information about the student (like their name or prior score) to avoid bias. If available knowledge could be an issue (for instance, very domain-specific answers the LLM might not know), we could supply relevant definitions or facts. In psychology, that’s usually not needed if the model is well-trained on general psych knowledge. Another issue: **prompt injections** – a savvy user could try to manipulate the grader by including something like “ignore the previous instructions” in their answer. We must guard against that by sanitizing the student’s input before putting it in the LLM prompt (e.g., by neutralizing prompt-break sequences or using the API’s tools to disable system override by user content). Essentially, treat the student answer as data, not as part of the instruction.

**Scoring and Mastery Update:** For MCQ, it’s binary correct/incorrect. For short answer, we might get a graded score or at least a correct/partial/incorrect judgment. This result is then used to update the user model. If partial, we might update mastery with a smaller increment than a full correct. The grade is also used for feedback to the student (discussed next).

One more aspect: **Open-Ended or Essay Questions** – beyond short factual answers, one could have the tutor ask more open questions like “How would you design an experiment to test X?” These are difficult to grade objectively. If included, we’d lean on the LLM heavily to provide an assessment, but likely in our system we stick to short answers to keep grading reliable. Any LLM-based grade can be double-checked: some systems run multiple LLMs or multiple prompts to see if they agree (due to time we assume one is fine).

**Storing Results:** Every graded result is stored in the database. For each question attempt, we save the question (or an ID/reference), what the student answered, whether it was correct, the score if numeric, and any particular flags (like which distractor was chosen for MCQ, or what misconceptions were detected). These records allow us to later analyze performance and are key for the adaptive scheduling.

In essence, the grading logic uses **exact matching for objective questions** and **LLM-powered rubric comparison for free responses**. This gives us both accuracy and depth: we not only mark right or wrong, but we can generate a **brief explanation or rationale** for why the answer was considered correct or not, using the LLM’s comparison output. Keeping the LLM grounded with a reference answer ensures the grading is aligned with the expected learning content and reduces randomness[[41]](https://arxiv.org/html/2501.09092v1#:~:text=%28NLP%29,are%20generated%20to%20the%20user). By addressing known pitfalls (consistency, bias, injection)[[38]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=,output%20in%20a%20deterministic%20way)[[40]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=cut,also%20not%20be%20present), we can harness the LLM to grade effectively, allowing immediate feedback to the student at scale.

## Feedback and Remediation: Closing the Loop with the Student

After grading a question, the system provides **feedback** to the student. Good feedback is crucial for learning – it should not only tell the student whether they were correct, but also **explain why** and help them learn from mistakes. Our tutor will deliver several types of feedback: the correctness outcome, the rationale or explanation, guidance on any misconceptions, and follow-up questions or prompts for recovery. The design follows the principle that *explanatory feedback* (explaining the why) is far more effective for learning than just saying right or wrong[[42]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=learners,of%20extraneous%20processing%20relative%20to)[[43]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=principle%20states%20that%20novice%20students,some%20of%20the%20boundary%20conditions).

Here’s how feedback works step-by-step:

* **Immediate Acknowledgment:** As soon as the student answers, the system should indicate whether it was correct or not. For a correct MCQ, a simple positive confirmation (“Correct!”) is given; for incorrect, a gentle indication (“That’s not correct.”). This instant correctness feedback serves as an anchor for the student – they know where they stand before moving on.
* **Rationale for the Correct Answer:** The tutor then provides an explanation of the answer. If the student was correct, this is a confirmation plus possibly an elaboration: *“Correct – classical conditioning is indeed about associating a neutral stimulus with an involuntary response. In this case, the bell and salivation... etc.”* If the student was wrong, the system explains the correct answer: *“The correct answer is C:* *Sensorimotor stage, because infants in Piaget’s first stage learn through direct sensory and motor interaction. Option A was incorrect because... ”*. Ideally, the explanation covers **why the correct answer is correct and why the main distractors are incorrect**. This approach helps address the student’s specific misunderstanding. In fact, when we generated the MCQ, we wanted distractors tied to misconceptions, so the feedback can explicitly refute them (e.g., “Answer B might seem right if you confuse negative reinforcement with punishment, but note that negative reinforcement *increases* a behavior by removing something aversive, whereas punishment decreases behavior[[34]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding).”). By providing this *principle-based explanation*[[42]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=learners,of%20extraneous%20processing%20relative%20to), we guide the learner to the proper concept and not just the item. The LLM can generate these explanations. We can either generate the explanation at question creation time (some systems have the LLM provide a rationale for each option when forming the question), or generate it on the fly after grading (prompt the LLM: “Explain why the correct answer is correct and why the student’s answer is wrong, referencing the concept”). Another approach is to have a pre-written explanation per question if using a static question bank. But with dynamic generation, it’s easiest to leverage the LLM. Given that the LLM *knows* the solution (we had it in order to grade), we can prompt it to output a concise teaching explanation. This explanation is then delivered to the user in the UI, often prefaced by something like “**Explanation:** …”.
* **Concept Reinforcement (Recovery Instruction):** If the student got the question wrong, the system may provide a brief **review of the underlying concept** to “recover” their understanding. This might blend into the rationale or come after it. For example: *“Recall that in classical conditioning, the response is involuntary. Pavlov’s dogs salivated involuntarily to a bell because it was paired with food. This is different from operant conditioning where a behavior is voluntary and influenced by consequences.”* Such restatement helps ensure the student gains the correct knowledge before proceeding. The user model can mark that a remediation was given. Sometimes, if the student’s answer shows a very specific misunderstanding, the tutor might give a targeted mini-lesson. For instance, “It looks like you might be confusing the sympathetic and parasympathetic nervous systems. Let’s clarify: the sympathetic nervous system triggers the fight-or-flight response (e.g., increasing heart rate), whereas the parasympathetic calms the body (rest-and-digest)…” Addressing the error directly solidifies learning. The LLM can produce these mini-explanations either as part of the feedback prompt or by having stored text for common misconceptions.
* **Praise and Encouragement:** The tone of feedback should be supportive. For correct answers, especially after difficulty, some praise is motivating (“Great job! You correctly identified the independent variable.”). For incorrect answers, encouragement to keep trying or that it’s okay to make mistakes keeps morale up (“Good effort – this was a tough question. Let’s review this and try another one.”). The system can modulate this based on the user’s performance history (if a student is getting many wrong, be extra encouraging and maybe lighten the load; if the student breezes through, still acknowledge their success but perhaps focus on challenging them further).
* **Follow-Up Question or Activity:** After giving the explanation, the tutor often will present a **follow-up question**. This is a critical part of the learning loop: practice and application. The follow-up could take several forms depending on the context:
* If the student was wrong, often the best follow-up is a *similar question* to test if they learned from the mistake. For example, a student messes up a question about reinforcement; the tutor explains it, then asks a new question on that concept to ensure the misconception is corrected. This follows the mastery learning idea of not moving on until the concept is understood. The new question might be slightly easier or more guided if we suspect the student is struggling, or it could be just another variant. Essentially, it’s a **second chance** on the same skill.
* If the student was right (especially if it seemed easy for them), the follow-up might escalate to a more challenging question on the same topic or a connected topic. For instance, after a correct basic question, the tutor can say “Great, let’s try a more challenging one:” and increase difficulty or move to the next subskill in that domain.
* Occasionally, the follow-up might not be a question but a prompt for reflection (especially in a one-on-one human tutor scenario). In our automated case, it’s likely always another question, but the system could also ask them to summarize what they learned if that suits the design, or give them an optional extra problem.
* The flow is continuous: The explanation and follow-up question together ensure the student has another opportunity to engage with the material and solidify their knowledge.

To illustrate a feedback exchange: 1. **Tutor:** “Your answer: *‘The dog salivates to the bell because the bell is a reward.’* — **Not quite.** The correct explanation is that in classical conditioning, the bell is initially neutral but becomes associated with the food, causing salivation. It’s not that the bell is a reward (that would be operant conditioning logic). In classical conditioning, the response (salivation) is involuntary, triggered by the bell after conditioning.[[42]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=learners,of%20extraneous%20processing%20relative%20to) Remember, negative reinforcement is not punishment; it *increases* behavior by removing something bad[[34]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding).

Now, let’s try another one: **What’s the difference between negative reinforcement and punishment?** (short answer)”

1. **Student:** (Writes an answer)
2. **Tutor:** (Grades and gives feedback accordingly.)

This example shows how the tutor corrects the specific misunderstanding and immediately prompts the student to articulate the corrected concept.

**Adaptivity in Feedback:** The system can also adapt how much feedback to give. If a student only slightly missed an answer, the tutor might say “Almost! You got the first part right but missed the second part.” If a student is completely lost, the tutor might give a more detailed explanation or even break down the concept further. The user model (mastery and prior attempts) can inform this: e.g., if the student has gotten this concept wrong multiple times, maybe the tutor goes more in-depth now or uses a different explanation approach (perhaps an analogy or a visual aid if possible). Since we are focusing on text, analogies in text can be powerful and the LLM is quite capable of generating analogies to explain (like comparing memory to computer storage, etc.). Personalization can come into play: e.g., *“Think of it this way: if you love video games – classical conditioning is like when a sound in the game makes you feel excited because it’s paired with a reward.”* Tailoring in such a manner uses info from the user model (interests) and can be done by including those cues in the prompt for the LLM when generating feedback.

Finally, we consider **timing** of feedback: The tutor is giving feedback immediately after each question (this is typical in tutoring systems and helps learning[[44]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=ASAG%20are%20manyfold%2C%20for%20both,1)). There is an educational debate about immediate vs delayed feedback, but for practice and tutoring (as opposed to a formal test), immediate explanatory feedback is usually best for learning, especially for novices[[42]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=learners,of%20extraneous%20processing%20relative%20to).

By delivering rich, concept-level feedback, the tutor helps the student not only see what the right answer was, but *why* it’s right and clears up any confusion surrounding it. The follow-up question then gives the student a chance to apply that new understanding, reinforcing the learning. This iterative loop of **question → answer → feedback → next question** embodies the tutoring process. It’s all automated but aims to feel to the student like they have a responsive tutor that corrects and guides them in real time, which is a major advantage of intelligent tutoring systems[[44]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=ASAG%20are%20manyfold%2C%20for%20both,1).

## Scheduling and Adaptivity: Mastery-Based Progression and Spaced Repetition

An effective tutor not only reacts in the moment but also plans *when* to revisit topics and *how* to sequence the learning over time. Our system will incorporate **adaptive scheduling** algorithms to decide what question to give next and when to review old material, using principles of spaced repetition and mastery learning.

**Mastery-Based Progression:** The student should ideally **master a topic before moving on** to a significantly harder one. The user model’s mastery estimates are used here. We can set a mastery threshold (for example, 0.9 probability or 90% success rate) for a skill to be considered “mastered.” Within a learning session, the tutor will focus on one or a few topics, giving questions and feedback until the student consistently demonstrates understanding. If the student keeps getting a particular type of question wrong, the system may stay on that subtopic longer, possibly switching question formats or giving a simpler intermediate question to build up to it. Conversely, if the student answers correctly several times in a row with confidence, the tutor may conclude the skill is mastered enough and introduce a **new topic or a more advanced concept**. This approach is akin to **mastery learning**, where the student must achieve a certain level of performance in one unit before proceeding[[14]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=one%20lesson%20remaining,1849). For example, the tutor might require the student to answer 3 questions in a row correctly on “operant conditioning” before moving to “observational learning.”

However, we also want to keep the sessions engaging – drilling one concept excessively can bore the student. So the system might mix in related topics or vary the question type to keep things fresh, as long as it doesn’t stray too far ahead. A strategy might be: work on a cluster of related skills (e.g., all within “Learning and Behavior” chapter) in parallel, so that if a student is stuck on one, we temporarily shift to another and come back later. But the exact strategy can be tuned. Simpler approach: sequentially go through the skill map, but allow *review* interleaving (see below).

**Adaptive Difficulty:** Within a topic, we adjust question difficulty based on the student’s performance as noted earlier. If a student finds everything too easy (quick correct answers), the system can skip ahead to more challenging material or perhaps reduce redundant practice. If the student is struggling (many errors), the system can drop the difficulty a notch (e.g., ask more recall-level questions or give partial hints in the question). This kind of *adaptive questioning* aligns with the idea of keeping the student in a productive zone of difficulty – not so easy that they’re not learning, but not so hard that they’re frustrated[[28]](https://arxiv.org/html/2507.05629v1#:~:text=Another%20challenge%20is%20to%20dynamically,more%20challenging%2C%20and%20vice%20versa). For instance, research on retrieval practice in real classrooms emphasizes adjusting difficulty: if a question was too easy, make the next one harder; if it was too hard, next one should be easier or reviewed[[28]](https://arxiv.org/html/2507.05629v1#:~:text=Another%20challenge%20is%20to%20dynamically,more%20challenging%2C%20and%20vice%20versa). Our system will implement that rule-of-thumb in how it selects or generates the next question.

**Spaced Repetition Scheduling:** Perhaps one of the most important aspects for long-term retention is scheduling **reviews** of previously learned material. The system should not just move linearly through topics and never return; instead, it should periodically give questions on older topics to strengthen memory (the testing effect). This is where the time-based decay in the user model comes into play. Each skill’s “strength” decays as time passes[[17]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=knowledge%20at%20any%20given%20time,average%20probability%20that%20the%20student). The scheduler should select topics for review that are getting weak. For example, if the student learned “Neuroscience basics” yesterday, by tomorrow it might be time for a quick review question on it, and then again next week, etc., with the intervals increasing if the student keeps showing they remember (this is classic spaced repetition – increasing intervals for items that are well remembered).

A possible algorithm is to maintain a **review queue** of questions/skills. Each time a student answers a question, we schedule that skill for its next review based on whether they got it right and how strong it is. If they struggled or got it wrong, we schedule it sooner (maybe later in the same session or next day). If they aced it, we schedule it further out (maybe in a few days or a week). This could be implemented similar to Leitner system or SM2 (SuperMemo) algorithms used in flashcard apps[[45]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=Leitner%20,The%20Leitner%20System%20for%20flashcards). For simplicity: have buckets of review intervals (1 day, 3 days, 7 days, 14 days, etc.). A skill moves to a longer interval bucket if answered correctly, or back to a shorter interval if answered incorrectly. The Duolingo half-life regression model does this more continuously, estimating probability of recall over time and scheduling when that probability falls to ~80%[[46]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=being%20taught%20in%20the%20lesson,strength%20meters%20to%20visualize%20the)[[19]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=a%20key%20component%20of%20the,word%20from%20the%20lessons%20in) – but that’s an advanced technique. We can approximate it with buckets.

In practice, when a student starts a study session (or even mid-session), the system will check for any **due reviews**. For example, “It’s been 3 days, and the mastery of Cognitive Psychology > Memory has decayed to 60%, let’s throw in a memory question now.” These review questions are interleaved with new material. This interleaving has been shown to solidify learning and also improve ability to discriminate between concepts. Our tutor might begin each session with a few review questions from past topics as a warm-up (much like a human teacher might start class with a quick quiz on last week’s material). Or it could periodically say, “Let’s take a short review break: [question on older topic].”

**Deciding What’s Next:** At any given point when the tutor needs to give a next question, it will decide among a few possibilities: 1. Continue with the **current topic** (especially if not mastered yet). 2. If current topic seems mastered or the student is fatigued with it, consider **moving to the next new topic** (following the curriculum order/prerequisites). 3. Check if any previously learned skills are due for **review** (based on time and decay). If yes, possibly serve a review question now (either as a break or at session end). 4. If the student made a specific error that should be addressed first, ask a **remediation question** on that now (basically still current topic).

The algorithm might weigh these factors. One approach: at regular intervals (like every N questions or every M minutes), force a review from past topics. Another approach: always maintain a mix like 70% new practice, 30% review.

For example, suppose a student has learned topics A, B, and is now on C. They answer a question on C correctly. The tutor sees that a topic A question is due for review (last seen 5 days ago). It might say: “Quick review: [Question from A].” After that, return to C. If the student got the review wrong, it may spend a bit giving feedback and maybe another follow-up on A (essentially a relearn), then resume C.

**Session Scheduling:** Beyond within a single session, if the platform supports scheduling study sessions or sending reminders, we could use the mastery data to suggest when the student should come back. For instance, if they stop now, the system might say “Good work! Make sure to review these 3 concepts tomorrow for retention.” or even send notifications when certain skills are decaying (like “Your Memory chapter is getting rusty, practice a few questions today!”). This is more of a feature outside the core tutor logic but leverages the same data.

**Adaptivity to User State:** Adaptivity isn’t just about knowledge; we can also adapt to **the student’s pace and attention**. For instance, if a student is getting many wrong in a row, maybe slow down and switch to an easier or previously mastered topic to rebuild confidence, then come back. Or if a student has been at it for a long stretch, maybe the system suggests a break or a different activity (could be a summary or a quick game question). Monitoring the student’s performance and even time on task can allow such adaptations to keep the student in an optimal state.

In summary, the scheduling system closes the loop on a macro level: it ensures **long-term retention through spaced reviews**[[18]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=exercises%20are%20taken%20from%20those,methods%20in%20language%20learning%20technology) and **efficient learning by focusing on weaknesses and moving on from strengths at the right time**. The result is a personalized learning trajectory through the topic map: each student might spend more time on what they personally find difficult and less on what comes easy, and everybody periodically revisits earlier topics so that knowledge sticks. This adaptive scheduling is key to the tutor’s effectiveness and sets it apart from a static quiz system.

## Data Schema and Storage Design

To support all the above functionality, we need to store data about users, content, and interactions in a structured way. A well-designed **database schema** will include tables for users, the curriculum (topics/skills), questions (or generation logs), and user performance history. Below is a conceptual schema with the main entities:

* **Users Table:** Stores user account information and profile details. Fields might include user\_id (primary key), name, email, etc., as well as any preferences (like preferred learning pace or whether they want daily review reminders). We might also store aggregate stats like overall progress or last login time here for convenience. If this is an institutional context, could include class or group info. (Of course, sensitive personal info should be minimized and protected.)
* **Topics/Skills Table:** Represents the nodes in the psychology topic map (the learning objectives). Fields: skill\_id, name (e.g., “Classical Conditioning”), parent\_skill\_id (self-referential foreign key if hierarchical), description (maybe a short definition of that concept), difficulty\_level or order index, etc. Another field or a separate table could define prerequisite relationships explicitly (e.g., a linking table prerequisite(skill\_id, prereq\_skill\_id)). This is our content structure reference. We may also maintain here any metadata like recommended number of questions to mastery or tags (e.g., which chapter or domain it belongs to like “Domain: Learning”).
* **User Mastery Table:** This table links users to skills and tracks their current mastery and practice stats. Fields could be: user\_id, skill\_id (together as a composite primary key or with its own id), mastery\_level (numeric score/probability), last\_practiced (timestamp), practice\_count, success\_count (how many times answered correctly), perhaps last\_interval (the current spaced repetition interval or bucket for that skill). Each time the user practices a skill, we update this record. If using a BKT model, mastery\_level updates based on the algorithm; if using simpler counts, we update success\_count, etc. This table is crucial for querying what a user knows and what’s due for review (e.g., we can find all records where mastery\_level is dropping below a threshold or where last\_practiced is a while ago).
* **Questions (Log) Table:** We don’t necessarily need a permanent store of every generated question text since the LLM can always make new ones. However, logging questions and responses is very valuable for analysis, and also needed if we want to avoid repeating the exact same question too soon. So we maintain a **Question/Response Log**. Fields: question\_id (if we want to reference it; even if not pre-stored, we can assign an ID when it’s asked), skill\_id (which concept it targeted), question\_text, possibly the options (for MCQ) and correct answer, or for short answer the expected answer. We also store which user got this question (or this could be part of the User Response table described next). We might separate **Question** and **Response** tables, or combine them. A possible approach:
* **QuestionTemplate Table:** If we predefine some templates or have common questions, store them here. But in our fully dynamic scenario, this may not be used.
* **UserQuestionHistory Table:** Each entry is one interaction (one question presented to one user and their answer). Fields: user\_id, question\_id or full question reference, skill\_id, the presented question text (we may store it for auditability), the user’s answer (text or chosen option), is\_correct (boolean or score), timestamp, response\_time (seconds taken), perhaps misconception\_tag if a known misconception was identified. This table becomes large, but it’s incredibly useful for analytics (e.g., to train improvements or to show the student a history of what they’ve done). It can also be used for detecting if the LLM ever gave a flawed question (we could flag weird logs for review).

In some systems like the CS Tutor example, they stored user chats and generated quizzes in the database[[5]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=Main%20chat%20screen%20). Similarly, we would log interactions for traceability.

* **Misconception Catalog Table (Optional):** If we explicitly manage a list of common misconceptions per topic, that could be a table: misconception\_id, skill\_id, description. For example, skill “Negative Reinforcement” might have misconception “thinking it means punishment.” We could tag distractors or wrong answers with these. The User Mastery/Skill table could then have a field or a related table UserMisconception that notes if a user has shown evidence of each misconception (e.g., if they selected that distractor, set a flag). This is optional; we could also just encode it in logic without separate schema, but storing it is helpful to ensure we address it later (we might specifically assign a remediation question to a user if we know they have a certain misconception).
* **Sessions or Progress Table (Optional):** We might have a table to track study sessions: when a user starts and ends a session, how many questions, etc. This could be useful for measuring engagement (like streaks, or average session length). Not strictly necessary for core functionality, but good for analytics and potentially gamification (like awarding points or badges for completing a session).
* **System Data Tables:** This includes any configuration or content needed. For example, if we have static definitions or reading material for concepts, a table could store paragraphs of explanation per skill that can be shown on demand. Or if we integrate any multimedia or external resources, their references might be stored. If using vector search for retrieval augmentation, we might have an index of text chunks from a psych textbook keyed by topic (though this might be outside the SQL schema, e.g., stored in a vector database or as files).
* **Security & Privacy Data:** If we have user consents, or parental consent forms for minors (COPPA compliance), a table might record that. Also, an admin or teacher account might have access to aggregated data – that might be separate tables or views.

From an implementation perspective, a relational database (PostgreSQL, MySQL) works, or a NoSQL store could also be used if we prefer flexibility (but relational fits well with the structured nature of curriculum and user scores). Each question attempt is an important event, so we might log it with an append-only approach for auditing. Storing every question text could be heavy, but text these days is not too big compared to multimedia, and the log could be pruned or compressed if needed (or just store a question ID referencing a template, but since we generate, there is no template – unless we hash the question text to see if it repeats).

**Citations in Data:** If we eventually want to show sources or have the LLM cite a reference for explanations (for factual accuracy), we might store references for content. For example, for a definition of a concept we might store a citation (like to a textbook or Wikipedia). But that is more an enhancement; not critical for internal design.

**Example Data Flow:** When a question is generated and shown, we create a UserQuestionHistory record with user, skill, question text, and maybe store the correct answer. When the user answers, we update that record with their answer, mark correct/incorrect, time taken. We then update UserMastery for that skill (and related skills if any). If wrong and a misconception matches the chosen answer, we could update a UserMisconception table entry. The next question selection will query these tables (for mastery and pending reviews).

By storing this data: - We enable the adaptive logic (the system queries what the user knows and what they struggle with). - The student’s progress can be displayed (like a dashboard showing mastery levels of each topic, perhaps a “skill tree” with percentages – similar to Duolingo’s strength bars[[19]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=a%20key%20component%20of%20the,word%20from%20the%20lessons%20in)[[47]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=Duolingo%20uses%20strength%20meters%20to,A%20student%20can%20tap%20the)). - Teachers or administrators could get reports if this is used in a class (like which areas the class finds hardest). - We can ensure continuity between sessions (when the user logs in tomorrow, the system knows exactly what they did before and where to resume). - Importantly, this data needs to be protected and used responsibly (which leads to privacy considerations, next section).

In summary, the database will store **users, the content structure (topics/skills), the user’s performance on each skill, and a log of questions and answers**. This schema supports the core functionality: quickly retrieving what to do next (via mastery and schedule info) and recording what happens. It’s generally a good practice to keep the data normalized (separate tables) but with necessary indices (like index by user\_id on history for fast lookup of a user’s past answers, etc.). The database becomes the memory of the tutor, enabling both real-time adaptivity and long-term tracking of educational progress.

## Interaction Flow: A Tutoring Session from Start to Finish

To illustrate how all these components come together, let’s walk through the **step-by-step flow** of a typical user’s tutoring session:

1. **Login and Session Initialization:** The user (student) opens the tutoring app and logs in. The frontend authenticates with the backend (which checks the Users table). Once authenticated, the backend retrieves the user’s profile and learning status (e.g., their mastery levels, what topic they were last working on, any scheduled reviews due). The frontend greets the user, possibly with a dashboard: “Welcome back! Last time you were studying *Social Psychology*. You have 3 review questions due from earlier units.” The user then clicks “Start Session” (or it might auto-start).
2. **Set Context/Choose Topic:** Depending on design, either the user selects a topic to work on (e.g., chooses from a list of psychology chapters), or the system decides the topic based on their progress. Let’s say the system recommends continuing with the next unit they haven’t finished. For example, if they last did Social Psych and mastered it, it might propose “Shall we start **Developmental Psychology** today?” The user can accept or pick something else. Once the topic (or set of topics) for the session is determined, the system is ready to pose a question. (If it’s the very first session for a new user, the flow might instead start with a quick assessment quiz spanning various basics to get an initial gauge.)
3. **Retrieve or Generate First Question:** The backend selects the first question of the session. Often, it’s good to do a quick **warm-up review**. Suppose the user has a couple of skills with decayed mastery – the system might first surface a review question: “Before we dive in, a quick review: [Question on a previously learned concept].” It calls the LLM to generate that review question (or pulls one from the question bank/log if reusing). The question is sent to the frontend and displayed to the user.
4. **User Answers the Question:** The student submits their answer. For MCQ, they click an option; for short answer, they type and hit enter. The frontend sends this response to the backend along with timing info (the UI can capture how long since the question appeared). For example, “User chose option B for question ID 123, took 15 seconds.”
5. **Backend Grades the Answer:** The backend now uses the grading logic. If MCQ, it checks if B is correct. If short answer, it formulates a grading prompt for the LLM with the student’s answer and the reference answer, and gets an evaluation. The result (correct/incorrect/score and possibly an explanation from LLM) is obtained. The backend updates the UserQuestionHistory record for this question with the user’s answer and correctness.
6. **Update User Model:** Based on the result, the backend updates the UserMastery for the associated skill(s). If correct, increase the mastery probability or mark knowledge as strengthened (and schedule next review further out). If incorrect, decrease mastery probability (and schedule a sooner review or keep it in current practice rotation). Also, record if a particular misconception was indicated (like if an incorrect choice corresponds to a known misconception, flag that in the user’s record). The response time could also adjust some confidence measure – for example, a very fast but wrong answer might be treated differently than a slow, thought-out wrong answer.
7. **Generate Feedback:** The backend prepares the feedback message. It likely calls the LLM to generate an explanation if it doesn’t already have one. For instance, if the grading LLM already returned a rationale, that can be formatted. Or a new prompt: “Explain why the correct answer is X and address why the student’s choice is incorrect.” Using the stored correct answer and the misconception info, the LLM crafts a response which includes the rationale and perhaps a tip. The backend then sends this feedback text to the frontend.
8. **Frontend Displays Feedback:** The user sees the tutor’s feedback on their screen. E.g., “**Incorrect.** The correct answer is B, *Variable Ratio schedule*, because in a variable ratio reinforcement, the reward comes after an unpredictable number of responses... [detailed explanation].” If the system provides the explanation in parts (like first just correct/incorrect then a “Show Explanation” button), the UI might handle that. But generally we can show the full feedback at once.
9. **Follow-Up/Next Question Decision:** Immediately after the feedback (or even alongside it), the system decides what the next question should be. If the answer was wrong, often the next question is a follow-up on the same concept to attempt correction (as described, maybe a rephrased question or a simpler one). If it was right, next might be a new concept or a harder question on the same concept if mastery isn’t confirmed with one question. The decision also considers if a review from another topic should be injected. Let’s say in our example the user got it wrong, so the system chooses to ask another question focusing on that same skill to give them practice. It might even say in the feedback “Let’s try another one: [next question]” as part of the same message. Or the UI could wait for the user to hit “Next” when ready.
10. **(Loop) Next Question Presentation:** The backend uses the LLM to generate the next question based on the chosen target skill and difficulty. This question is sent to the frontend and displayed. We are now back at step 4 with a new question. The loop of **question → answer → feedback** repeats.
11. **Mastery Achieved or Time Up:** This loop continues, possibly cycling through different topics as mastery and scheduling dictate. Suppose after a few tries the student is now consistently getting the developmental psychology questions right. The system sees mastery is achieved for those objectives, and might decide to **conclude the topic**. It could congratulate the student for mastering that topic. If there’s another new topic next in the curriculum, it might suggest moving on either in the same session or next time. Alternatively, suppose the user has been practicing for 30 minutes and decides to stop. They can end the session (or perhaps the system nudges that it’s a good stopping point). The backend would mark any unfinished topics as “in progress” and store the partial mastery levels.
12. **End of Session Summary:** When the session is ending, it’s nice to give the student a summary of what was accomplished. The backend can compile stats: “You answered 15 questions today (12 correct, 3 incorrect). You mastered **Operant Conditioning** and reviewed 3 previous topics. Great work! Here are a couple of concepts to review next time: [list].” This summary is sent to the frontend to display. It reinforces progress and can highlight improvements or lingering trouble spots (e.g., “Keep an eye on *Memory Encoding* – we’ll practice that again soon.”).
13. **Save State and Logout:** The user logs out or closes the app. All their progress is already saved incrementally in the database (mastery levels updated, history logged). If needed, the backend could do any cleanup. The next time they log in, the system knows where to pick up.

Throughout this session, **adaptivity** is in play: the content and pacing adjust based on the learner’s responses. The user is effectively in a conversation (though structured) with the tutor: question, answer, feedback, question, and so on, much like a human tutor would do interactively. The difference is it’s powered by the LLM and the rules we’ve set up.

**Error Handling:** If at any point the LLM fails (say the generation comes back nonsense or not at all due to an outage), the system should have fallbacks. For instance, it could have a small set of backup questions or a message: “Having trouble generating a question, let’s try again.” Robustness might include validating LLM outputs (ensuring an MCQ actually has the required format, etc., and regenerating if not).

**User Control:** The flow described is mostly system-led (tutor decides next steps). But we can allow the user some control, such as a *skip button* (“I want a different question”) or the ability to ask for a hint before answering. If a hint is requested, the backend could prompt the LLM for a hint (and log it) and show it. This wasn’t explicitly in requirements, but it’s a plausible extension to mirror human tutoring. For simplicity, our main flow assumed no manual hints.

**Multi-modal notes:** We assumed text Q&A. If images or charts were used (e.g., a question shows a brain diagram and asks the student to label something), the flow would incorporate media. That’s beyond our textual focus, but the system could host such content and refer to it.

Finally, this stepwise flow ensures that at every step the **student is guided, the system adapts, and the learning process is continuous** until either the material is learned or the session ends. The user always knows what to do next (answer the next question) and gets immediate feedback, which is a proven approach in intelligent tutoring to keep engagement and promote learning[[44]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=ASAG%20are%20manyfold%2C%20for%20both,1).

## Privacy and Security Considerations

Building a system that tracks detailed educational data means we must carefully handle **privacy and security**. Student data – including their personal info, performance history, and interactions – is sensitive and often protected by laws (especially if minors are involved or if used in schools). We outline measures to ensure data privacy and secure usage:

* **Compliance with Privacy Laws:** Depending on jurisdiction and user age, laws like **FERPA** (Family Educational Rights and Privacy Act in the US), **COPPA** (Children’s Online Privacy Protection Act, for U.S. children under 13), and **GDPR** (General Data Protection Regulation in the EU) will apply. Our system should be designed to comply with these. For example, **FERPA** requires that student educational records (which our tutor data would be) are protected and not shared without consent. If this tutor is used in a school context, the school can designate the system as a “school official” under FERPA’s exception to share data for educational purposes[[48]](https://nssa.stanford.edu/tutoring/data-use/measures-data-collection/student-data-privacy-guidance#:~:text=Accelerator%20nssa,volunteers%2C%20companies%2C%20or%20other), but we’d still need to ensure confidentiality. **COPPA** would require **parental consent** before collecting personal info from young children[[49][50]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=The%20Children%E2%80%99s%20Online%20Privacy%20Protection,providers%2C%20including%20online%20tutoring%20platforms). So if our user base includes sub-13 users, we must get verifiable parental consent during signup and allow parents to review/delete the data. **GDPR** mandates principles like data minimization, purpose limitation, and rights for users to access or delete their data[[51]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=,necessary%20for%20the%20stated%20purposes). We should only collect data needed for the tutoring purpose, clearly inform users (or parents) what we collect and why, and honor requests to delete data. For instance, the system could include a feature to delete an account which would erase or anonymize the user’s personal data and records (except maybe aggregate stats with no identifiers).
* **Secure Data Storage:** All user data in the database should be stored securely. That means using strong access control (only authorized services or personnel can access the database). Personal identifying information (PII) like names, emails should be encrypted at rest. Many systems also encrypt performance data, though the main risk is with PII. We should also encrypt data in transit – use HTTPS for all client-server communication so answers and feedback are not intercepted. Within the architecture, if the backend and database are separate, ensure connections are secured. Implementing regular security audits on the database for vulnerabilities is a good practice[[52]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=Another%20major%20concern%20is%20unauthorized,help%20detect%20and%20address%20vulnerabilities)[[53]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=including%20names%2C%20birthdates%2C%20and%20academic,help%20detect%20and%20address%20vulnerabilities).
* **Access Control and Authentication:** Users should have secure authentication (passwords hashed properly, or single sign-on if in an institution). If there are different roles (like teacher or admin who can see student progress), enforce strict access control. A teacher should only see data for their students, for example. The system might allow a student to share their progress with a mentor or parent – but that should be opt-in and clearly controlled.
* **Minimize Personal Data:** The tutor doesn’t actually need a lot of personal data to function – mainly a login and perhaps age (to comply with COPPA). We should avoid collecting extraneous PII. For example, we don’t need their address or phone for the tutoring function, so we shouldn’t ask for it. If any analytics are done, aggregate or anonymize data where possible. Under GDPR principles, we state the **specific purpose** (e.g., “data is collected to provide personalized tutoring and track progress”) and we don’t use it for unrelated purposes like marketing without consent[[51]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=,necessary%20for%20the%20stated%20purposes).
* **Data Retention Policies:** We shouldn’t keep user data forever by default. We might implement a retention schedule – e.g., if an account is inactive for a long time, we might purge detailed records after some years, or at least give users the option to delete old data. GDPR emphasizes not retaining data longer than necessary[[51][54]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=,necessary%20for%20the%20stated%20purposes).
* **Handling LLM Provider Data**: If our LLM is accessed via an external API (e.g., OpenAI’s servers), we are essentially sending user query data to a third-party. This raises privacy questions: user’s answers and the content of questions will be transmitted. We should review the LLM API’s data policy – for example, OpenAI allows opting out of data retention for API calls, which we should do if possible so that user data isn’t stored on their servers beyond processing. Also, avoid sending any user-identifiable info in prompts. The prompt to the LLM doesn’t need the student’s name or email; it just needs the question content. So sanitize prompts to only include necessary text. This limits exposure of PII. Ideally, if privacy is a top concern (like for a school), using an on-premises or open-source LLM that runs locally would be better so data doesn’t leave the system.
* **Security of the Application:** Beyond data storage, the application itself must be secure. Protect against common web app vulnerabilities (SQL injection, XSS, etc.) since we are collecting inputs (the student’s answers are inputs too). Also, consider the possibility of a user trying to abuse the system – e.g., one could try to input malicious text to confuse the LLM (prompt injection) or to break the UI. We should sanitize and validate user inputs. For prompt injection: when we put the student’s answer into the LLM prompt for grading, we can format it in a way that minimizes it being interpreted as instruction. For instance, always prepend “Student answer: [text]” and perhaps escape or remove problematic tokens. Newer LLM APIs also allow us to label user content vs system content to reduce injection issues.
* **Audit and Transparency:** It’s good to keep logs of who accessed data (especially if admins or multiple roles). If something goes wrong or a user has a concern, we have an audit trail. Also, being transparent with users about what is stored and how it’s used builds trust. Possibly provide a “download my data” feature if appropriate, aligning with user rights (especially under GDPR which gives a right to data portability).
* **Backup and Recovery:** A security consideration is also reliability – making sure data isn’t lost. Regular backups of the database should be taken (with encryption). But ensure backups are stored securely too (encrypted and with limited access) to avoid any breach through a stolen backup.
* **Protecting Educational Records:** In an educational context, student progress data is considered part of their educational record and needs protection similar to grades. If the system is used in a school, likely there will be data privacy agreements required by districts (in US, many states have additional student data privacy laws). The design we propose should be able to meet those: by limiting use of data, securing it, and giving control to the institution over it.
* **User Anonymity in Research:** If the system data is ever used for research or improving the system (say, analyzing answer logs to improve questions), ensure that identifying information is removed. It’s fine to analyze aggregate correctness or even misconceptions frequencies, but not to expose student identities.
* **Cybersecurity against External Threats:** There’s also the threat of external hacking attempts. We should host the system in a secure environment, use firewalls, keep software up-to-date, and possibly have DDoS protection if it’s a public site. For example, ensure the LLM API keys and database credentials are kept secret (not exposed in client code) and rotated if needed. Penetration testing can be done to find vulnerabilities. And in case of any data breach, have a plan (like notifying users, etc., as per legal requirements).

In conclusion, designing with privacy and security in mind is not just about compliance but also about maintaining user trust. Students (and parents and educators) need confidence that this sensitive data about their learning is safe. By following encryption practices, access controls, and legal guidelines like **obtaining necessary consent and limiting data use/retention**[[51]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=,necessary%20for%20the%20stated%20purposes), we create a secure environment. And by building the system so that it only uses data for the student’s own tutoring benefit and not exposing it elsewhere, we adhere to the ethical use of educational data. All of these measures ensure that the real-time LLM tutor is not only intelligent and helpful but also trustworthy and responsible in handling user information.

[[1]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=1,specified%20for%20the%20lecture) [[2]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=2,for%20a%20better%20user%20experience) [[3]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=,and%20Q%26A%20about%20the%20document) [[4]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=6,hit%20and%20uses%20the%20answer) [[5]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=Main%20chat%20screen%20) [[6]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=3,and%20Q%26A%20about%20the%20document) [[7]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=7,or%20ease%20in%20usage) [[8]](https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87#:~:text=7,or%20ease%20in%20usage) Design and Implementation of an LLM-Powered Intelligent Tutoring System for Computer Science Education | by Pelin Ece Burgun | Aug, 2025 | Towards AI

<https://pub.towardsai.net/design-and-implementation-of-an-llm-powered-intelligent-tutoring-system-for-computer-science-474d91542ccd?gi=a89e2a96bf87>

[[9]](https://www.coursesidekick.com/psychology/2663557#:~:text=between%20the%20two%20on%20behavior,Students%20will%20applycritical%20thinking%20skills) PSY 150 Alignment Chart(2) (2) (docx) - Course Sidekick

<https://www.coursesidekick.com/psychology/2663557>

[[10]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=the%20probabilistic%20and%20deep%20learning,intuitive%20notions%20of%20ideas%20that) [[11]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=The%20concept%20map%20or%20otherwise,the%20exercise%20instructs%20and%20assesses) [[15]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=represent%20the%20available%20knowledge%20and,is%20a%20more%20recently%20developed) [[16]](https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf#:~:text=observation%20pairs%20X%20%3D%20%7B%28,Similarly%2C%20the%20objective%20of) Microsoft Word - \_paged\_overlay\_134.doc

<https://personales.upv.es/thinkmind/dl/journals/intsys/intsys_v12_n12_2019/intsys_v12_n12_2019_9.pdf>

[[12]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20for%20generating%20MCQs,levels%20of%20Bloom%E2%80%99s%20Revised%20Taxonomy) [[22]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding) [[28]](https://arxiv.org/html/2507.05629v1#:~:text=Another%20challenge%20is%20to%20dynamically,more%20challenging%2C%20and%20vice%20versa) [[29]](https://arxiv.org/html/2507.05629v1#:~:text=LLMs%20have%20been%20utilized%20to,49) [[30]](https://arxiv.org/html/2507.05629v1#:~:text=medical%20experts%20for%20an%20anatomy,promising%2C%20some%20generated%20MCQs%20still) [[31]](https://arxiv.org/html/2507.05629v1#:~:text=Addressing%20the%20generation%20of%20plausible,fewer%20%E2%80%9Dnonsense%20distractors%E2%80%9D%20compared%20to) [[32]](https://arxiv.org/html/2507.05629v1#:~:text=hallucination%20and%20bias%20persisting,based%20models) [[33]](https://arxiv.org/html/2507.05629v1#:~:text=shot%20baselines%20for%20math%20MCQs,based%20models) [[34]](https://arxiv.org/html/2507.05629v1#:~:text=Extensions%20to%20theses%20studies%20include,students%20toward%20the%20correct%20understanding) Enhancing Student Learning with LLM-Generated Retrieval Practice Questions: An Empirical Study in Data Science Courses

<https://arxiv.org/html/2507.05629v1>

[[13]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=2,level%20Spanish%20in%02struction%20%28Vesselinov%20and) [[14]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=one%20lesson%20remaining,1849) [[17]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=knowledge%20at%20any%20given%20time,average%20probability%20that%20the%20student) [[18]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=exercises%20are%20taken%20from%20those,methods%20in%20language%20learning%20technology) [[19]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=a%20key%20component%20of%20the,word%20from%20the%20lessons%20in) [[20]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=student%20model%2C%20as%20seen%20beneath,bars%2C%20the%20skill%20has%20grown) [[21]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=Duolingo%20uses%20strength%20meters%20to,bars%2C%20the%20skill%20has%20grown) [[45]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=Leitner%20,The%20Leitner%20System%20for%20flashcards) [[46]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=being%20taught%20in%20the%20lesson,strength%20meters%20to%20visualize%20the) [[47]](https://research.duolingo.com/papers/settles.acl16.pdf#:~:text=Duolingo%20uses%20strength%20meters%20to,A%20student%20can%20tap%20the) A Trainable Spaced Repetition Model for Language Learning

<https://research.duolingo.com/papers/settles.acl16.pdf>

[[23]](https://arxiv.org/html/2502.12633v1#:~:text=significance%20of%20recognizing%20and%20adapting,develop%20individualized%20teaching%20strategies%20that) [[24]](https://arxiv.org/html/2502.12633v1#:~:text=restrict%20their%20ability%20to%20address,In) One Size doesn’t Fit All: A Personalized Conversational Tutoring Agent for Mathematics Instruction

<https://arxiv.org/html/2502.12633v1>

[[25]](https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full#:~:text=Soland%20,a%20test%20item%2C%20while%20non) [[26]](https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full#:~:text=Soland%20,responding%20would%20suggest%20that%20a) [[27]](https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full#:~:text=Kuhfeld%2C%202020%3B%20Wise%20et%20al,negative%20impacts%20on%20estimated%20performance) Frontiers | Test engagement and rapid guessing: Evidence from a large-scale state assessment

<https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2023.1127644/full>

[[35]](https://arxiv.org/html/2501.09092v1#:~:text=the%20specific%20task%20of%20grading%2C,are%20generated%20to%20the%20user) [[36]](https://arxiv.org/html/2501.09092v1#:~:text=%28NLP%29,an%20overall%20grade%2C%20the%20breakdown) [[37]](https://arxiv.org/html/2501.09092v1#:~:text=Our%20system%20is%20the%20first,points%20examined%20in%20the%20problem) [[41]](https://arxiv.org/html/2501.09092v1#:~:text=%28NLP%29,are%20generated%20to%20the%20user) SteLLA: A Structured Grading System Using LLMs with RAG

<https://arxiv.org/html/2501.09092v1>

[[38]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=,output%20in%20a%20deterministic%20way) [[39]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=,output%20in%20a%20deterministic%20way) [[40]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=cut,also%20not%20be%20present) [[44]](https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5#:~:text=ASAG%20are%20manyfold%2C%20for%20both,1) LLM-based automatic short answer grading in undergraduate medical education | BMC Medical Education | Full Text

<https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-024-06026-5>

[[42]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=learners,of%20extraneous%20processing%20relative%20to) [[43]](https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F#:~:text=principle%20states%20that%20novice%20students,some%20of%20the%20boundary%20conditions) The Feedback Principle in Multimedia Learning (Chapter 19) - The Cambridge Handbook of Multimedia Learning

<https://www.cambridge.org/core/books/abs/cambridge-handbook-of-multimedia-learning/feedback-principle-in-multimedia-learning/F52480F8ED64AE42AA8D999E9460C08F>

[[48]](https://nssa.stanford.edu/tutoring/data-use/measures-data-collection/student-data-privacy-guidance#:~:text=Accelerator%20nssa,volunteers%2C%20companies%2C%20or%20other) Student Data Privacy Guidance | National Student Support Accelerator

<https://nssa.stanford.edu/tutoring/data-use/measures-data-collection/student-data-privacy-guidance>

[[49]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=The%20Children%E2%80%99s%20Online%20Privacy%20Protection,providers%2C%20including%20online%20tutoring%20platforms) [[50]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=The%20Children%E2%80%99s%20Online%20Privacy%20Protection,providers%2C%20including%20online%20tutoring%20platforms) [[51]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=,necessary%20for%20the%20stated%20purposes) [[52]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=Another%20major%20concern%20is%20unauthorized,help%20detect%20and%20address%20vulnerabilities) [[53]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=including%20names%2C%20birthdates%2C%20and%20academic,help%20detect%20and%20address%20vulnerabilities) [[54]](https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/#:~:text=,necessary%20for%20the%20stated%20purposes) Ensuring Student Safety and Data Privacy in Online Tutoring: Essential Guidelines and Best Practices

<https://caddellprep.com/live-online-tutoring/student-safety-and-data-privacy/>