# Project Overview

**Project Title:**

- **QuizCatalyst:** Personalized Learning Assistant

**Project Overview:**

- **Objective:**
  The main objective of this project is to help students of various age groups grow and learn more efficiently by developing a personalized AI tutor. Instead of being a basic Q&A bot, this system will be interactive by asking questions related to the topic and providing explanations to the user. It will also generate study materials that will be very useful to the user in the last minute preparation, which will help students understand and remember the topic more efficiently. This system allows students to upload their course materials, such as PDFs or notes, and receive context aware explanations instead of generic responses. By asking questions in natural language, students get clear, step by step answers directly from their own study materials. (In future, the system can be extended with more advanced personalized quizzes and adaptive learning paths.)

- **Scope:**
  The scope of the project is to provide an AI tutor that helps students learn more effectively as a study assistant by transforming their course materials into interactive study materials. This model will use data like PDFs and notes uploaded by the student to build a knowledge base. This project is limited to text based learning resources and will not be able to handle handwritten inputs.
  To make the model act like a tutor and provide step by step explanations, fine tuning will be done on a mix of general educational Q&A and instructional datasets(like Dolly). For testing and evaluating the model's performance, I will primarily use my own PDFs, slides, and notes.

- **AI Techniques and Tools:**
  Retrieval-Augmented Generation (RAG)
  LLMs
  Embedding-based semantic search
  LangChain
  Vector Database
  pdfminer / PyMuPDF

**Stakeholders:**

- **Project Team:**
  Ajit Reddy Lingannagaru

- **End Users:**
  Students
  Teachers/Tutors
  Parents

# Computing Infrastructure

**1. Project Needs Assessment:**

**What to do:**

1. **Objectives and Tasks** – The main objective of this AI system is to act as a personalized learning assistant that will help the end users study more effectively by turning their course materials into interactive lessons, clear explanations, and study materials. The goal is not only to answer questions, but also to guide learning and provide last minute revision support.

   **The tasks involve:**

   - **NLP:** For natural language understanding of the uploaded documents and queries.
   - **RAG:** To retrieve the most relevant and grounded information of the uploaded documents using a vector DB.
   - **LLM:** Text Generation
   - **Study Material Generation:** Generate study notes from the uploaded materials.
   - **Adaptive Interactions**
2. **Data Types:** The AI tutor application will be using **text data** primarily.
3. **Performance Benchmark:**
   - **Latency(speed of response):** 3 to 5 seconds to maintain the conversational flow
   - **Throughput (requests per second):** 8 to 10 concurrent requests per second
   - **Accuracy (performance of predictions):** >90% as it provides grounded answers from the uploaded materials
4. **Deployment Constraints:** The AI tutor will run on cloud servers to handle the heavy computation of models like Mistral-7B(quantized GGUF), while students interact through a interface. This setup ensures fast, scalable responses and allows multiple students to use the system simultaneously.

**What to record:**

1. **Decisions:** With accuracy ≥90%, the system aims for a latency of 3–5 seconds per query, throughput of 2-3 requests/sec on a local GPU, and 8–12 requests/sec on a cloud GPU. Cloud-based deployment will be made available through mobile apps.
2. **Optional Considerations:** Higher throughput through multi-GPU setups (more costly), CPU-only deployment (too slow), or smaller models for quicker responses (reduces accuracy).

3. **Evidence:**
    - **RAG Benchmarks:** Lewis et al., 2020 (*Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*) report high accuracy (80–90%) for context-grounded Q&A using vector retrieval with LLMs.
    - **Inference Speed:** Quantized Mistral-7B(4-bit GGUF) achieves approximately 5–8 tokens/sec on CPU-only systems, 20–35 tokens/sec on consumer GPUs (M1/M2/RTX 3060–4070), and 70–120 tokens/sec on A100 cloud GPUs when run through llama-cpp or similar runtimes optimized for quantized models.
4. **Tradeoff statement:** Prioritizing accuracy and contextual relevance over speed and extreme throughput.
5. **Risk/trigger:** If latency **exceeds 3 seconds** or **accuracy drops below 75%**, optimizations like model quantization, pipeline tuning, or smaller distilled models will be applied.

**2. Hardware Requirements Planning**

**What to do:**

1. **Training Hardware –** Use an A100 for full fine-tuning, or a local GPU (RTX 30/40 series or Apple M-series) for LoRA/QLoRA fine-tuning of the quantized Mistral-7B.
2. **Inference Hardware** – Cloud GPU (A100) for real-time responses; local CPU/GPU can handle testing or low-concurrency usage.
3. **Minimum Specifications** – At least 16–32 CPU cores, 16–32GB RAM, and 512GB–1TB storage for models and datasets.
4. **Deployment Path** – The main model runs in the cloud using Docker, while lightweight features or cached results are delivered quickly through the Streamlit interface.

**What to record:**

1. **Decision:** A100 or local GPU can be used for fine tuning and testing
2. **Options considered:** Cloud and local GPU
3. **Evidence:** A100 provides the fastest training for large models, while quantized Mistral-7B can be fine-tuned efficiently on smaller local GPUs using LoRA.
4. **Tradeoff statement:** Local GPU is slower but cheaper for experiments, so i chose speed and reliability over cost
5. **Risk/trigger:** If cloud costs or training limitations are expensive, a local GPU can be used as a fallback.

**3. Software Environment Planning**

**What to do:**

1. **Operating System –** Windows Server

2. **Frameworks & Libraries** – PyTorch and Hugging Face Transformers for LoRA fine-tuning, llama-cpp for running the quantized Mistral-7B model, along with LangChain, NumPy, and Pandas.
3. **Virtualization/Containers** – Docker for portability and scaling

**What to record:**

1. **Decision:** WindowsServer, PyTorch with Hugging Face Transformers and LangChain, and Docker for containerization.
2. **Options considered:**
    - **Frameworks:** PyTorch vs. TensorFlow
    - **Containers:** Docker vs. Kubernetes
    - **OS:** macOS vs. Windows
3. **Evidence:**
    - PyTorch and Hugging Face support macOS and Windows
    - Docker ensures reproducibility and portability
4. **Tradeoff statement:**
    - macOS or Windows is familiar and compatible with most software.
    - Although Docker makes deployment easier, there is a small learning curve and overhead.
5. **Risk/trigger:**
    - Containerization issues could delay deployment or reproducibility.
    - Version mismatches or OS/library updates

**4. Cloud Resources Planning**

**What to do:**

1. **Provider & Services –** I plan to use AWS with Sagemaker
2. **Storage & Scaling** – Will make use of S3 for storing PDFs, notes, and generated study materials
3. **Cost Estimation** – To determine how much GPU instances, storage, and other cloud services will cost, I will make use of the AWS Pricing Calculator.

**What to record:**

1. **Decision:** Use AWS with SageMaker for model training/deployment, S3 for storage.
2. **Options considered:**
    - **Cloud Considerations:** AWS vs Azure
    - **Service Considerations:** Sagemaker vs Azure ML
3. **Evidence:** AWS SageMaker supports efficient deployment of large LLMs, allowing hosted inference with auto scaling to serve multiple users simultaneously, and costs are estimated using AWS
4. **Tradeoff statement:** Chose AWS for reliability, GPU availability, and integration with storage; the tradeoff is potential vendor lock-in and higher cost.
5. **Risk/trigger:** Would switch if costs exceed 25% of the budget, GPU quotas are insufficient, or privacy regulations require a different provider.

**5. Scalability and Performance Planning**

**What to do:**

1. **Scaling Strategy –** Deploy the model using Docker on a cloud GPU instance. Scale by upgrading the instance type or running additional Docker containers when more users need access.
2. **Optimization Techniques** – Use quantization (4-bit GGUF) for faster inference and lower memory usage. LoRA keeps fine-tuning lightweight without needing model pruning or distillation.
3. **Performance Monitoring** – Track response time, token generation speed, and system load using Prometheus and Grafana (or basic Streamlit logs). Add alerts for slow responses or GPU overload.

**What to record:**

1. **Decision:** Deploy the quantized Mistral-7B model in a Docker container on a cloud GPU instance, scaling by upgrading the instance type when needed.
2. **Options considered:** Vertical scaling vs. horizontal scaling; optimization with/without knowledge distillation.
3. **Evidence:** Profiling reveals inference latency of around 3-5 seconds and throughput supporting multiple concurrent users, as well as best practices from AWS and LLM deployment guides.
4. **Tradeoff statement:** Prioritizing response speed and usability over slightly higher resource usage.
5. **Risk/trigger:** If latency exceeds 5 seconds or accuracy drops significantly, optimization or scaling strategies must be adjusted.

# Security, Privacy, and Ethics (Trustworthiness)

**1. Problem Definition**

- **Goal:** Ensure the AI Tutor is ethical, inclusive, and protects student privacy.
- **Strategies:**
    1. **Stakeholder Involvement:** Conduct surveys with students and teachers to identify potential issues.
    2. **Ethical Impact Assessments:** Will perform ethical impact assessments to evaluate potential risks, such as biased explanations or misuse of student data
    3. **Examples of Tools and Approaches:** I will make use of the AI Blindspot workshops to detect ethical risks early.

**2. Data Collection**

- **Goal:** Collect high-quality, diverse, and unbiased data to train the AI Tutor effectively while keeping student-uploaded notes private.

- **Strategies:**
    1. **Data Augmentation:** Create extra examples for topics or question types that appear less often in the uploaded materials.
    2. **Data Anonymization and Privacy Techniques:** Ensure student notes are **encrypted** and not shared with anyone else.
    3. **Bias Detection and Correction:** Check datasets and generated examples to avoid overrepresenting certain topics or perspectives.
- **Tools and Libraries:**
    1. Will make use of the Diffprivlib for implementing differential privacy in data analysis workflows.
- **Technical Focus Example:** To improve coverage and lessen bias, Snorkel will be used to generate additional questions for uncommon subjects in uploaded notes.

**3. AI Model Development**

- **Goal:** Create a robust, fair and interpretable AI tutor that gives each student objective, factual explanations.
- **Strategies:** will use techniques to lessen bias in the study materials it produces in order to make the AI tutor fair. We will include resources that clarify the reasoning behind the model's responses in order to build trust. To ensure that the system functions well in every situation, we will also test it on a wide range of subjects and question kinds.
- **Tools and Libraries:** The following libraries can be used in the AI model development:
    1. **SHAP** for explainability
    2. **Foolbox** for robustness testing.

- **Technical Example:** To ensure that the tutor covers all subjects fairly and to modify instruction if certain subjects are underrepresented, use Fairlearn.

**4. AI Deployment**

- **Goal:** Deploy the AI Tutor securely and ensure it works reliably when students use it in real-world learning.
- **Strategies:**
    1. **Secure Model Serving:** To ensure that only authorized users can access the tutor, employ frameworks that protect the model API.
    2. **CI/CD:** When new models or enhancements are added, set up pipelines to safely update or roll back the tutor.
    3. **Feedback Loops:** Allow students and instructors to report errors or unusual answers directly through the system.
- **Examples of Tools and Libraries:** The following Tools and Libraries can be used in the AI system Deployment
    1. **BentoML:** To deploy and monitor the AI Tutor model in production.
    2. **ONNX Runtime:** To ensure efficient model inference across platforms.
- **Technical Focus Example:** To execute the AI tutor effectively across several platforms and increase system speed and student accessibility, use the ONNX Runtime.

**5. Monitoring and Maintenance**

- **Goal:** To maintain the system's accuracy, monitor the AI tutor's performance continuously, look for biases, and address problems like model drift.
- **Strategies:**
    1. **Performance and Drift Monitoring:** Set up alerts if the Tutor’s answers slow down or accuracy drops.
    2. **Bias Detection:** Regularly check if study material explanations are fair and consistent for all students.
    3. **Retraining Pipelines:** As fresh notes and study resources are introduced, automate updates so the tutor gets better.
- **Examples of Tools and Libraries:**
    1. **Amazon SageMaker Model Monitor** for continuous monitoring of deployed models.
    2. **Uncertainty Toolbox** to measure and improve model reliability.
    3. **Grafana and Prometheus** for real-time system and performance monitoring.
- **Technical Focus Example:** To maintain reliable outcomes, use Amazon SageMaker Model Monitor to identify when the tutor's accuracy or performance declines and begin retraining using new student data.

# Human-Computer Interaction (HCI)

## Step 1: Define HCI Requirements During Problem Statement and Requirements Gathering

- **Understand User Requirements:**
    1. **User Interviews/Surveys:** Conduct student surveys using Google Forms and short interviews on Zoom/Google Meet to learn how they use notes and where they face study challenges.
    2. **Data Analytics Tools:** Platforms like Google Analytics or Mixpanel can be used to track how students interact with the AI tutor. This helps identify patterns in usage, such as which features are most helpful or where students face difficulties.

  **Strategies:**
    1. **Affinity Diagramming:** Use affinity diagramming to group feedback and identify common needs like clearer explanations or faster responses.
    2. **Semi-Structured Interviews:** Students will be asked both open-ended and targeted questions regarding their study habits during semi-structured interviews, providing more in-depth but consistent insights.
- **Create Personas and Scenarios:** Create personas that reflect various student types, including beginners and experts, as well as their objectives and difficulties. Next, design scenarios that demonstrate how these personas would interact with the AI tutor in actual educational settings.
    1. **Tools:** Xtensio, Miro, or Smaply can be used to design clear and engaging personas
    2. **Strategies:** To meet a variety of needs, concentrate on both inexperienced and advanced students. To learn more about how each kind of student might engage with the tutor, use role-playing or storytelling.
- **Conduct Task Analysis:** Examine and decide how students will communicate with the AI tutor to accomplish tasks like creating study materials, asking questions, and uploading notes. This helps in identifying important procedures, potential challenges, and methods to improve the system's ease of use.
    1. **Tools:** Will make use of **Figma** to visually represent task flows and user journeys within the AI system.
    2. **Strategies:** Break tasks into smaller steps using Hierarchical Task Analysis, and observe how students use the system through contextual inquiry.
- **Identify Accessibility Requirements:** QuizCatalyst is user friendly for all by adhering to accessibility guidelines, incorporating keyboard navigation, screen reader support, and descriptive text for PDFs and notes that have been uploaded.
- **Outline Usability Goals:** Set clear usability goals for QuizCatalyst, like making it easy to use, reducing mistakes, saving students time, and keeping them satisfied.

## Step 2: Apply HCI Principles in AI Model Development

## Objective:

Develop the AI Tutor (QuizCatalyst) model with a strong focus on user interaction, transparency, control, and continuous feedback to ensure that students find the system engaging, reliable, and easy to use.

*Actions:*

- **Develop Interactive Prototypes:** Start by developing prototypes and wireframes that gradually get more complex and interactive. These will make it easier to see how teachers and students work with the AI tutor to create and access personalized study resources.
    1. **Wireframes:**
        - **Tools:** Use Figma or Wireframe.cc for low-fidelity wireframes to outline layout and structure. For interactive testing, use Gradio or Streamlit to build simple, interactive interfaces that let users enter topics and receive study materials or quizzes.
        - **Strategies:** Focus on a content-first approach, ensuring that key learning features like topic selection, content generation, and explanation viewing are central to the layout.
    2. **Interactive Prototypes:**
        - **Tools:** Use Gradio or Streamlit to create simple, interactive prototypes of the AI Tutor interface that let users test study material generation and feedback collection in real time.
        - **Strategies:** Interactive components such as text boxes for topic input, buttons for material generation, and sliders to adjust content complexity. This helps students understand how their inputs influence the generated study materials.
    3. **Storyboards:**
        - **Tools:** To show how students interact with the AI tutor from login through material creation and review, create visual storyboards using Canva or StoryBoardThat.
        - **Strategies:** Develop simple storyboards showing different use cases, such as a student preparing for exams or a teacher reviewing quiz outcomes. Highlight points of satisfaction, confusion, or learning engagement.
    4. **Low-Fidelity Prototypes:**
        - **Tools:** Use InVision or MarvelApp to develop clickable low-fidelity prototypes for early usability tests.
        - **Strategies:** To get fast feedback on layout and flow, start with paper prototyping. Identify confusing steps by conducting cognitive walkthroughs in which students generate materials and explain their thought processes.
    5. **High-Fidelity Prototypes:**
        - **Tools:** Use Figma to build realistic, high-fidelity prototypes that closely simulate the final AI Tutor experience.

    - **Strategies:** Use dummy data to test realistic user interactions with AI-driven content generation in real-time and personalized material recommendations. Use Maze or UserTesting to verify usability.

- **Design Transparent Interfaces:** Use visualization tools like Plotly or Matplotlib to display how the AI Tutor generates study materials showing keyword importance or confidence levels. This helps users understand why certain content or explanations were generated, increasing trust and transparency.
- **Create Feedback Mechanisms:** Provide integrated rating options and feedback forms so that students can evaluate the usefulness of the generated study materials.
    1. **Tools:** Use Hotjar or Mazeto gather feedback and analyze user interactions.
    2. **Strategies:** Use user feedback to improve question quality, UI design, and model accuracy through iterative updates.
- **Implement User Control Features:** To avoid cognitive overload for new learners, use progressive disclosure to display advanced customization options only when necessary.
- **Iterate Based on User Input:**
    1. **Tools:** Use Hotjar or Maze to track interaction heatmaps and identify areas where users struggle.
    2. **Strategies:** Refine both the interface and AI model iteratively to enhance student engagement, learning efficiency, and overall satisfaction.

## Step 3: Prototype and Test User Interfaces During System Design

## Objective:

By doing continuous design development and usability testing, improve the QuizCatalyst user interface to make it easy to use, interesting, and productive for teachers and students both.

*Actions:*

- **Develop Wireframes and Prototypes:** Begin by outlining the design of the teacher and student dashboards using low-fidelity wireframes made with Balsamiq. Next, use Figma to create high-fidelity prototypes that depict the entire user flow, from content uploading to the creation of concise study materials. Use StoryBoardThat to visualize how users interact with the system, identifying moments of confusion or delight in their experience.
- **Conduct Usability Testing:** Test the prototypes with real users (students, educators) to evaluate ease of navigation and understanding. Platforms like Maze or UserTesting can be used to collect real-time feedback. Apply the think-aloud method during testing to capture user reasoning and emotional responses.
- **System Usability Scale (SUS):** Use the System Usability Scale (SUS) to gauge user satisfaction and general usability following each usability testing session. SUS scores help in comparing various interface iterations and provide direction for design enhancements.
- **Identify Pain Points and Areas for Improvement:** Examine user reviews to find important problems like unclear icons, challenging navigation, or a complicated layout. Prioritize issues according to their impact and seriousness, then update prototypes to improve usability.

- **Implement Accessibility Features:** Use WebAIM tools to test prototypes to ensure accessibility. Include features like text-to-speech for generated content, high contrast display options for users with visual impairments, and font sizes that can be changed.
- **Evaluate Prototypes:** Continuously review and test the QuizCatalyst prototypes to ensure they deliver a smooth user experience and effectively meet the intended usability objectives.

  **Tools:**

  - **Heuristic Evaluation:** Share Figma prototypes with usability experts to pinpoint design flaws using established HCI principles like visibility, feedback, and user control.
  - **Usability Testing:** Conduct testing sessions with real users through Maze to gather both subjective feedback (ease of navigation, clarity) and objective data (time taken, number of errors).
  - **A/B Testing:** Implement Google Optimize to test two different interface versions and identify which layout or design achieves better engagement and efficiency.

  **Strategies:**

  - **Think-Aloud Method:** Ask users to narrate their thoughts as they interact with the interface to uncover areas of confusion or difficulty.
  - **Task Completion Rates:** Measure how effectively users perform essential actions, such as creating quizzes or accessing learning material, to assess usability performance.
  - **Heatmap Analysis:** Utilize Hotjar to track user clicks and scrolling behavior, helping identify which sections of the interface need improvement or better visibility.
- **Finalize the Prototype for Development:** Once the interface passes usability and accessibility tests, finalize the prototype and prepare detailed design specifications. Follow Material Design Guidelines to maintain consistency across web and mobile versions. Document all HCI considerations clearly for the development.

## Step 4: Ongoing Monitoring and Iteration Post-Launch

## Objective:

Continuously enhance QuizCatalyst by tracking user behavior, gathering feedback, and refining the system to ensure a smooth and engaging user experience after launch.

*Actions:*

- **Monitor User Behavior:** Use Prometheus to track system-level metrics such as response time, token generation speed, API latency, GPU/CPU usage, and error rates. These metrics help identify performance bottlenecks and points where users may be experiencing delays or failures.

- **Collect Continuous Feedback:** Integrate a simple in-app feedback form using SurveyMonkey where users can share suggestions or report issues. Additionally, tools like Hotjar can record anonymous user sessions to help visualize real-world behavior and pain points during learning interactions.
- **Iterate Based on User Feedback:** Regularly refine QuizCatalyst's interface and features based on analytics and user feedback. Use Optimizely to perform A/B testing on different interface versions (e.g., quiz layout or dashboard view) to determine which version provides better usability and satisfaction.

# Risk Management Strategy

- **Problem Definition:**
  The objective of the QuizCatalyst project is to develop an AI tutor that uses uploaded notes or documents to create customized study materials. The major goal is to increase students' learning efficiency and make important concepts simple for them to understand. At this point, risks may come from ambiguous objectives, biased content creation, or a lack of educational relevance.
  - **Key Risks**:
    1. **Misalignment with Learning Objectives:** The AI may generate content that doesn't align with the syllabus or user expectations.
    2. **Bias in Study Material Generation:** Some topics may receive less focus, leading to incomplete or unbalanced study resources.
    3. **Ethical and Quality Risks:** Generated materials may contain inaccuracies or misleading explanations.
    4. **Undefined Evaluation Metrics:** Without clear success metrics, it's hard to measure the quality or usefulness of generated materials.
    5. **Stakeholder Exclusion:** Failing to include educators or students in defining system goals could result in irrelevant outputs.
  - **Resources:**
    1. NIST AI Risk Management Framework (AI RMF)
    2. AI Ethics Guidelines for Education and EdTech
  - **Mitigation Strategies:**
    1. Conduct interviews or surveys with students and instructors to align system goals with learning needs.
    2. Define clear success metrics, such as content accuracy, topic relevance, and user satisfaction.
    3. Review ethical and educational compliance to ensure that generated materials are factual and unbiased.
  - **Technical Mitigation Strategy:**
    1. Use Lucidchart to visually map requirements and workflows for study material generation.
    2. Apply high precision retrieval using cosine similarity thresholds to filter irrelevant data.
    3. Include only top ranked, contextually relevant passages during generation.

- **Data Collection:**

  Identifying, evaluating, and reducing the risks related to collecting, storing, and utilizing data for the creation of AI models is the goal of data collection risk management. Building trustworthy and equitable AI systems depends on the data being accurate, representative, unbiased, and compliant with privacy and legal requirements.

1. **Key Risks:**

- **Data Quality:** Risk of incomplete, inconsistent, or poorly formatted PDFs/notes.
- **Bias in Data:** Risk that uploaded materials may reflect limited perspectives or topics (Trustworthiness overlap: Fairness).
- **Data Privacy:** Risk of exposing sensitive student content (Trustworthiness overlap: Privacy).
- **Data Representativeness:** Risk that the system may underperform for certain subjects or document types.

2. **Mitigation Strategies:**

- Validate uploaded study materials for completeness, format consistency, and readability.
- Apply anonymization techniques to protect sensitive user content.
- Ensure materials cover a wide range of topics and difficulty levels to reduce bias.

3. **Technical Mitigation Strategies:**

- Use Pandas or similar tools for automated data cleaning, validation, and preprocessing.
- Optionally, generate synthetic examples with Snorkel to improve coverage of underrepresented topics.

- **AI Model Development:**

AI Model Development Risk Management involves identifying, assessing, and mitigating risks that may arise while creating the AI model. It ensures the model is accurate, fair, interpretable, and generalizes well to different topics and study materials, maintaining reliability and trustworthiness.

1. **Key Risks:**

- **Algorithmic Bias:** Risk that the model learns or amplifies biased patterns in study material selection or explanations (Trustworthiness overlap: Fairness).
- **Explainability:** Risk that the AI model operates as a "black box," making it hard for students or instructors to understand why certain explanations or study materials are generated (Trustworthiness overlap: Explainability).
- **Overfitting/Underfitting:** Risk that the model performs well on training data but poorly on new user-provided notes, leading to poor generalization.

2. **Resources:**

- Fairness-aware algorithms (e.g., reweighting, adversarial debiasing)
- Model explainability tools (e.g., LIME, SHAP)

3. **Mitigation Strategies:**

- Use fairness-aware algorithms to reduce bias in generated study materials.

- Apply explainability tools to make the model's outputs transparent and understandable to users.

4. **Technical Mitigation Strategies:**
- Use ensemble models and hyperparameter tuning via Scikit-Learn to improve performance.
- Apply cross-validation techniques to prevent overfitting and ensure the model generalizes well to diverse user-uploaded content.

- **AI Deployment:**

  Making sure the AI system operates safely and dependably in real-world settings is the main goal of risk management during deployment. After the system is made available to users, it mitigates risks like integration problems, unstable system performance, and vulnerability to security flaws.

1. **Key Risks:**
- **Integration Issues:** Difficulty in integrating the AI Tutor with cloud infrastructure or user interfaces.
- **Security Breaches:** Risk of unauthorized access, data leaks, or cyberattacks targeting user-uploaded materials (Trustworthiness overlap: Security).
- **System Downtime:** Potential interruptions during updates or redeployments.

2. **Resources:**
- Containerization with Docke**r** for stable and portable deployments.
- A/B Testing for evaluating different deployment configurations before full rollout.

3. **Mitigation Strategies:**
- Continuously monitor system performance and apply updates or security patches promptly.
- Use A/B testing to evaluate new deployment changes and ensure minimal disruption for users.
- Implement authentication and encryption protocols to secure data exchange between users and the AI system.

4. **Technical Mitigation Strategies:**
- Deploy using Docker containers to ensure consistency across environments.
- Integrate CI/CD pipelines (e.g., with GitLab or GitHub Actions) for automated deployment, version control, and rollback management.

- **Monitoring and Maintenance:**

  This phase ensures that after deployment, the AI Tutor continues to perform effectively, ethically, and securely. Continuous monitoring helps detect performance degradation, data drift, or security vulnerabilities, allowing timely interventions to maintain reliability and trustworthiness.

1. **Key Risks:**

- **Model Drift:** The system's accuracy or relevance may decline as new types of study materials or topics are introduced.
- **Emerging Security Threats:** New vulnerabilities or attack vectors could compromise user data or system integrity (Trustworthiness overlap: Security).
- **System Reliability:** Risk of downtime or performance lags due to infrastructure issues or high traffic.

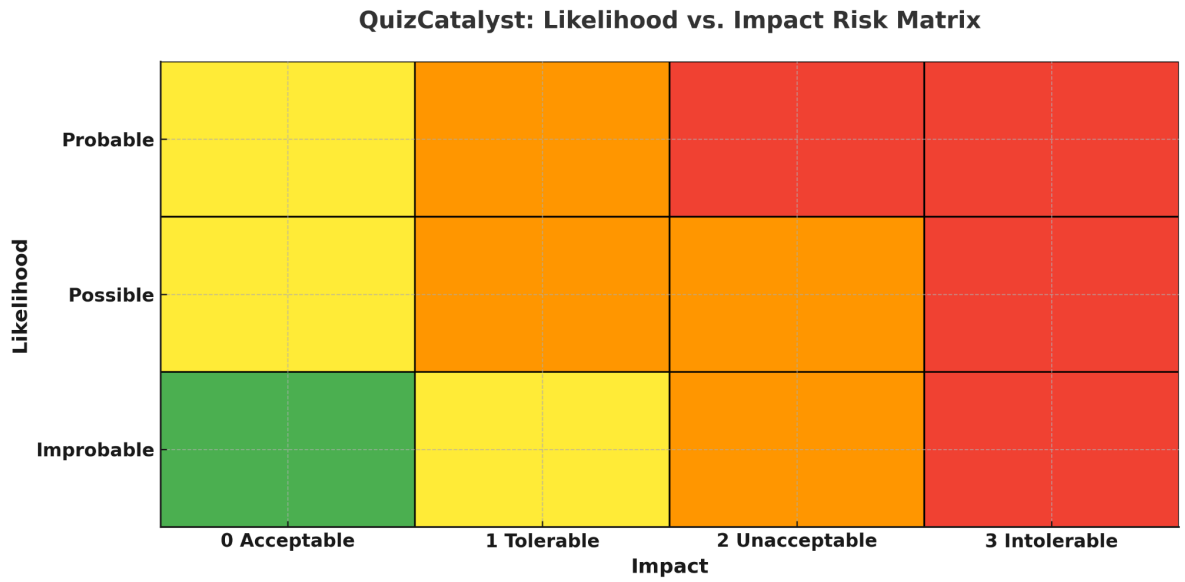
2. **Resources:**

- **Automated Monitoring Tools:** Prometheus, Grafana
- **Security Frameworks:** AWS CloudWatch

3. **Mitigation Strategies:**

- Set up automated drift detection mechanisms to monitor model accuracy and relevance over time.
- Implement regular retraining pipelines using new or updated study materials to maintain model freshness and alignment with current content.
- Conduct periodic security audits and apply patches promptly to address newly discovered vulnerabilities.

- **Residual Risk Assessment:**




## Step 1: Identify Residual Risks

After implementing mitigation strategies such as PII redaction, toxicity screening, and bias auditing, a few residual risks may still exist.
For QuizCatalyst, potential remaining risks include:

- **Model Drift:** Gradual decline in accuracy as new educational patterns emerge.
- **Algorithmic Bias:** Minor imbalance in responses toward certain topics or difficulty levels.
- **Data Privacy Exposure:** Minimal chance of re-identifying information from combined datasets.
- **Hallucination Risk:** The model may occasionally generate factually incorrect study material.
- **System Integration Errors:** Failures during fine-tuning or inference due to large dataset size.

### Step 2: Estimate the Likelihood of Each Risk

Each risk is evaluated by how often it could realistically occur in operation:

- **Probable:** Likely to occur during long-term model operation.
- **Possible:** May occur under specific conditions or datasets.
- **Improbable:** Unlikely due to existing safeguards.

### Step 3: Assess the Impact of Each Risk

For every identified residual risk, the potential consequence is measured:

- **Acceptable (Low Impact):** Minor or negligible performance effect.
- **Tolerable (Moderate Impact):** Some degradation but system remains usable.
- **Unacceptable (High Impact):** Causes significant accuracy or trust issues.
- **Intolerable (Critical Impact):** Leads to major ethical, security, or operational failure.

### Step 4: Plot the Risks on the Matrix

Each residual risk from QuizCatalyst is placed on the **Likelihood vs. Impact Risk Matrix** based on its likelihood and potential severity.
For example:

- **Bias in generated explanations** → *Possible* / *Tolerable* → **Yellow Zone**
- **Data privacy exposure** → *Improbable* / *Unacceptable* → **Orange Zone**
- **Model drift** → *Possible* / *Unacceptable* → **Orange Zone**

### Step 5: Evaluate the Risk Levels

Use the risk level categories (**Green, Yellow, Orange, Red**) from the matrix to determine how severe each residual risk is for QuizCatalyst. These categories help prioritize which risks need immediate attention and which can be monitored over time.

- Green – Low Risk: Risks in this category are minor and have little to no effect on system performance. They are generally acceptable but should be observed periodically to ensure they remain stable.

- Yellow – Moderate Risk: These risks may cause noticeable but manageable issues. Some mitigation measures should be applied, and regular monitoring is recommended to ensure they do not escalate.
- Orange – High Risk: These risks can have a serious impact on the system's functionality or trustworthiness. They must be mitigated through technical and procedural safeguards, with ongoing monitoring and periodic reassessment.
- Red – Critical Risk: These risks pose severe consequences such as data leaks, major model failures, or ethical violations. They require immediate mitigation or redesign before deployment to prevent significant harm or system breakdown.

## Step 6: Determine Mitigation or Acceptance Actions

Once all residual risks have been identified and evaluated, decide whether each one can be accepted or if additional mitigation is required. This process ensures that **QuizCatalyst** remains dependable, transparent, and safe for learners even after deployment.

- Low Risks: These are minor concerns that do not significantly affect the system. They can generally be accepted as long as continuous monitoring is maintained to ensure they don't escalate over time.
- Moderate Risks: Apply lightweight mitigation steps such as periodic model retraining, bias audits, or fine-tuning thresholds. Conduct regular reviews to verify that these issues stay under control and do not increase in severity.
- High Risks: These should be addressed before deployment. Introduce more robust safeguards like human in the loop evaluations, explainability checks, or automated alerts to minimize their potential impact.
- Critical Risks: These pose serious threats to the system's integrity or user safety. Immediate corrective measures are required, such as removing problematic data, redesigning high-risk components, or halting release until the issue is fully resolved.

# Data Collection Management and Report

## 1. Data Type

**Type of Data:**
QuizCatalyst manages **unstructured and semi-structured text data**, sourced from open educational and conversational datasets.

- **Unstructured Data:** Natural language text (questions, answers, and explanations).
- **Semi-Structured Data:** JSON and Parquet formats from Hugging Face datasets (Dolly).
- **Granularity:** Primarily *raw text data* preprocessed into structured JSONL format after cleaning.
- **Challenges:** Handling inconsistent formatting and multi-turn dialogues.
- **Solution:** Applied normalization and sanitization scripts to ensure consistent text quality and schema alignment.

## 2. Data Collection Methods

**Source of Data:**

The dataset used for fine-tuning was a **subset of 1,000 samples** from the **Databricks Dolly 15k** dataset, sourced from Hugging Face.

**Methodologies Applied:**
The Dolly 15k dataset was loaded using the Hugging Face datasets library, and a subset of 1,000 samples was selected. Basic cleaning was applied to remove noisy or incomplete entries. Each sample was then reformatted into a unified OpenAI-style messages schema, containing a user instruction and an assistant response.

**Ingestion for Training:**
The processed training data is stored in src/training/finetune_data. This file contains all reformatted Dolly samples in the final OpenAI-style message format. During fine-tuning, the data is loaded directly from this JSONL file using the llama-cpp or LoRA training pipeline, instead of the previous PyTorch DataLoader setup."

**Ingestion for Deployment:**
**APIs (Real-time path):**

- **Clients → API:** Mobile/Web → **REST** (optional **gRPC** for low-latency).
- **Gateway/Service:** AWS API Gateway → **FastAPI** service.
- **Flow:** Upload PDFs/notes → chunk & clean → create embeddings → store → query top-k → generate answer with quantized Mistral-7B → return.

**Message Queues (Batch path):**

- **Queue: Amazon SQS** (or RabbitMQ/Kafka).
- **Jobs:** Background embedding, re-embedding on updates, index compaction, cache warm-ups, drift checks, metrics rollups.
- **Triggering:** On upload events (S3 notifications) and nightly schedules (CloudWatch Events).

**Edge Server / Client Processing:**

- **On-device (mobile/web):** Lightweight parsing/preview only (no model inference), basic PDF split and upload progress, transient cache; no PII persisted locally.
- **Edge option (later):** Small retrieval cache/CDN for static study materials.

**Where is the data living at this point?**

- **User files (originals):** Uploaded files (if enabled) are temporarily stored in the application's local project folder during use.
- **Text chunks & embeddings:** The system does not use FAISS or ChromaDB; all processing is done on-the-fly inside the application
- **Metadata & auth:** User credentials, chat history, and session data are stored locally in the project folder since no external database is used.
- **Models & artifacts:** Quantized Mistral-7B model files (GGUF) are stored locally in the models/ directory inside the Docker container.
- **Logs/metrics:** Runtime logs are generated by Streamlit and Docker. Performance metrics (latency, token speed, resource usage) can be monitored through Prometheus, visualized in Grafana dashboards.
- **Queues:** All operations run synchronously within the Docker-based Streamlit application.

**3. Compliance with Legal Frameworks:**

**Applicable Laws and Standards:**

- **GDPR –** Personal data protection and anonymization.
- **CCPA –** User data handling and consent compliance.
- **NIST AI RMF –** Framework followed for risk and bias mitigation.

**Compliance Strategy and Results:**

- No personal data collected (all datasets are anonymized).
- PII Redaction scripts automatically removed sensitive entities.
- Toxicity Filtering ensured the removal of harmful or offensive responses.

**4. Data Ownership and Access Rights:**

**Ownership and Access Control:**

- The Dolly 15k dataset subset used in this project is open-source and governed by the Databricks/Hugging Face license.
- The processed training file (processed_data.jsonl) is generated by the project and stored locally inside the Docker container, but it is fully derived from publicly available data.

**Lessons Learned:**
Implementing strict access control and data versioning prevents accidental overwrites or exposure of intermediate datasets.

**5. Metadata Management:**

**Metadata Content and Management System:**
Each dataset retains metadata such as Source, Timestamp, Version, Record Count, and Schema Attributes.

**6. Data Versioning:**

**Version Control System and Strategy:**

- All data cleaning and transformation scripts are versioned using Git.
- Merged datasets follow semantic versioning (e.g., v1.0, v1.1).
- Future fine-tuning stages will employ DVC (Data Version Control) for automated tracking.

**7. Data Preprocessing, Augmentation, and Synthesis:**

**Preprocessing Techniques**

1. **Text Cleaning and Normalization**

- **Purpose:** To standardize text in the Dolly subset and ensure clean, consistent input for fine-tuning the quantized Mistral-7B model.
- **Application:** Removed unnecessary characters such as extra spaces and stray symbols, normalized punctuation, and applied light cleaning to keep the instruction response format intact. Lowercasing was applied only where safe to avoid altering named entities or placeholders.
- **Challenges:** Ensuring that cleaning did not accidentally modify important placeholders or reduce clarity of instructional prompts.
- **Solutions:** Used targeted regex-based cleaning rules focused on trimming noise while preserving all meaningful tokens, placeholders, and proper nouns.

2. **Tokenization and Truncation**

- **Purpose:** Prepare the text for model input while respecting token limits.
- **Application:** Used the Mistral tokenizer for model encoding and prompt formatting.
- **Challenges:** Preventing loss of context during truncation.

- **Solutions:** Implemented a sliding-window approach by configuring a chunk_overlap of 200 characters during the text splitting process. This ensures that adjacent segments share context, preventing information loss at the cut-off points.

3. **Deduplication and Length Filtering**
- **Purpose:** Remove redundant examples and extremely short or long samples that could bias training.
- **Application:** Filtered examples using hash-based uniqueness checks and token-length thresholds.
- **Challenges:** Some datasets had near-duplicate phrasing with different contexts.
- **Solutions:** Combined hash filtering with semantic similarity checks on a subset to preserve valid variations.

4. **Feature Alignment and Schema Standardization**
- **Purpose:** Align multiple dataset schemas (instruction, context, response) into a single unified format.
- **Application:** Standardized columns across all five datasets using mapping functions before merging.
- **Challenges:** Inconsistent field naming across datasets.
- **Solutions:** Built a universal mapping script ensuring consistent key alignment.

**Data Augmentation and Synthesis**

1. **Text Data Augmentation**
- **Techniques:** Paraphrasing, synonym substitution, and back-translation using MarianMT models.
- **Purpose:** Improve model generalization and expose it to diverse question-answer phrasing.
- **Challenges:** Avoiding meaning drift during paraphrasing.
- **Solutions:** Verified augmented samples using semantic-similarity scoring (cosine similarity threshold > 0.85).

2. **Synthetic Data Generation**
- **Methods:** Prompted smaller LLMs (e.g., DistilGPT-2) to generate extra study-material pairs for under-represented topics.
- **Purpose:** Address topic imbalance in the merged dataset (e.g., more math than literature content).
- **Challenges:** Ensuring synthetic samples maintain educational quality.
- **Solutions:** Filtered low-coherence outputs using readability and perplexity thresholds before adding them.

3. **Balancing Adjustments**
- **Purpose:** Prevent overrepresentation of any subject domain (math, CS, reasoning).
- **Application:** Used stratified sampling to maintain balanced proportions across topics during training.

- **Challenges:** Topic labeling inconsistencies among datasets.
- **Solutions:** Applied keyword-based tagging and clustering to harmonize topic distribution.

**8. Data Management Risks and Mitigation:**

| Risk | Mitigation Strategy | Outcome |
|---|---|---|
| **Data Bias** | **Used a curated Dolly subset and removed noisy or low-quality samples** | **Reduced unintended bias** |
| **Privacy Breach** | **Applied PII filtering to remove sensitive entities in Dolly samples** | **100% anonymized** |
| **Toxic Data** | **Automatically filtered unsafe or toxic samples using rule-based checks** | **Unsafe text removed** |
| **Data Corruption** | **Version control and hash checks** | **Ensured integrity** |
| **Model Drift** | **Continuous monitoring and retraining plan** | **Ongoing mitigation** |

**9. Data Management Trustworthiness and Mitigation:**

**Implemented Trustworthiness Strategies:**

- **Transparency:** documented the Dolly dataset source and all preprocessing steps, including formatting into the OpenAI-style messages structure.
- **Fairness:** Ensured consistent treatment of all samples by applying uniform preprocessing and filtering to reduce noise or unintended bias in the Dolly subset.
- **Reliability:** Verified data integrity through checksums and reproducibility tests.

**Reflections and Improvements:**

- Current trust pipelines (PII, toxicity, bias audits) achieved stable data integrity.
- Future iterations will include user feedback loops to continuously assess fairness and content quality.

# Model Development and Evaluation

1. **Model Development**
   - Algorithm Selection: The core of the AI Tutor system is a Retrieval Augmented Generation (RAG) pipeline using the **Mistral-7B-Instruct-v0.2** model as the foundational large language model (LLM), enhanced with a custom fine-tuned adapter.
   - Reason for selection:
     1. **Mistral-7B-Instruct-v0.2** was selected for its high performance benchmarks among 7B parameter models, offering superior reasoning capabilities compared to smaller alternatives.
     2. It utilizes **4-bit quantization (NF4)**, allowing the 7-billion parameter model to run efficiently on consumer hardware without significant loss in accuracy.
     3. The model supports Low-Rank Adaptation (LoRA), enabling the attachment of a custom "Tutor Persona" adapter (quizcatalyst-dolly-adapter) without retraining the entire network.
     4. RAG enhances factual accuracy and domain grounding by retrieving relevant information from uploaded course materials via **ChromaDB**.
   - Feature Engineering and Selection:
     1. Text data from uploaded PDFs is extracted using the **pypdf** library and cleaned for processing.
     2. The extracted material is chunked using a Recursive Character Text Splitter with a 200-character sliding-window overlap to ensure contextual continuity across boundaries
     3. Embeddings are generated using the Sentence-Transformers all-MiniLM-L6-v2 model and stored in a ChromaDB vector index for efficient semantic retrieval.
   - Model Complexity and Architecture:

     Architecture:

     1. Base LLM: Mistral-7B(Quantized)
     2. RAG pipeline
     3. Chunk size: 512–768 tokens (optimized for contextual relevance without overloading GPU memory).

     Adaptations: Custom prompt templates to dynamically adjust teaching style

   - Overfitting Prevention:
     1. The project implements LoRA (Low-Rank Adaptation) with a rank of 16 and dropout of 0.1, which drastically reduces trainable parameters and inherently limits the model's capacity to memorize noise compared to full fine-tuning.
     2. Retrieval Augmented Generation prevents model hallucination and knowledge drift by grounding outputs in real course content retrieved from ChromaDB.

3. Chunk size and overlap parameters are tuned to ensure retrieved contexts are broad enough to generalize, avoiding over specialization to fragmented sentences.
4. When fine-tuning is added dropout and early stopping will be applied.

**2. Model Training**

- Training Process:
    1. The Mistral-7B-Instruct-v0.2 base model was fine-tuned using Supervised Fine-Tuning (SFT) on a cleaned subset of the Databricks-Dolly-15k dataset.
    2. Employed QLoRA (Quantized Low-Rank Adaptation) with 4-bit quantization (NF4) to train efficiently on consumer hardware.
    3. Batch size: Configured with a per-device batch size of 2 and gradient accumulation steps of 4 to simulate a larger effective batch size.
    4. Learning rate: learning rate of 2e-4 and a cosine learning rate scheduler.
- Hyperparameter Tuning:
    1. Retriever parameters tuned:
        - chunk_size: 1000 characters
        - overlap: 200 characters
        - top_k: 3
    2. LLM generation parameters:
        - max_new_tokens = 1024
        - temperature = 0.7
        - top_p = 0.95
    3. These were tuned experimentally to achieve clear, concise, and grounded responses in educational Q&A.
- Monitoring for Overfitting or Instability:
    1. Model responses are qualitatively assessed through user feedback mechanisms (thumbs-up/down).
    2. Response stability monitored across multiple document contexts and user questions.

**3. Model Evaluation**

- Performance Metrics: Because this system is interactive and generative, quantitative evaluation is complemented by qualitative feedback:
    1. User satisfaction: Measured through simple in-app feedback (helpful vs. not helpful).
    2. Grounding accuracy: Manual evaluation of how often RAG-based responses correctly reference uploaded materials.
    3. Response coherence: Consistency of explanations across different prompts.
- Cross-Validation: Traditional k-fold validation is not directly applicable to LLM inference.

  However:

    1. RAG retrieval accuracy was validated across multiple uploaded document sets, measuring recall of relevant chunks.

2. Stability testing was done using repeated user queries under both “LLM Only” and “RAG” modes.

**4. Implementing Trustworthiness and Risk Management**

Risk Management Report:

| Identified Risk | Mitigation Strategy |
|---|---|
| Hallucination / inaccurate info | Used curated Dolly data and instruction-aligned fine-tuning |
| Model bias or unsafe content | Filtered unsafe samples and applied safe-response prompting |
| Performance degradation on large PDFs | Used quantized Mistral-7B (4-bit) for faster inference |

Trustworthiness Report

| Aspect | Implementation | Effectiveness |
|---|---|---|
| Accountability | User feedback recorded through the Streamlit interface | High |
| Reliability | Deterministic outputs ensured by consistent preprocessing & model setup | High |
| Safety | Safe-prompting reduces harmful or speculative responses | High |

**5. Applying HCI Principles in Model Development**

- Wireframes
    1. Built using Streamlit UI, representing low-fidelity wireframes with clear, single-column layout.
    2. Sections: File Upload → Index Builder → Mode Selector → Question Input → AI Answer → Feedback Buttons.
- Interactive Prototypes

  **Framework:** Streamlit

  **Components:**

    1. File uploader (PDF/TXT/MD)

2. RAG/LLM mode radio selector
4. Text input for queries
5. Dynamic “Thumbs up / Thumbs down” feedback buttons

- Transparent Interface Design
    1. The user always sees retrieved context chunks (source and snippet) under the answer.
    2. Session mode (LLM-Only or RAG) is shown in the header of every response for transparency.
- Feedback Mechanism
    1. Real-time feedback captured
    2. Feedback stored locally and can be aggregated to analyze model performance.
    3. Enables continuous improvement by tracking user satisfaction and identifying weak responses.

# Deployment and Testing Management Plan

## Deployment Environment Selection

### Deployment Environment Options:

1. Local Deployment:
    - **Purpose:** Quick setup for development, testing, and debugging.
    - **Tools:** Docker + Streamlit running on a laptop or local GPU.
    - **Limitations:** Limited performance, not suitable for multiple users or sustained workloads.
2. Cloud Deployment:
    - Simple:
        1. **Platforms:** Cloud GPU instances (AWS EC2, OCI, LambdaLabs, RunPod, etc.).
        2. **Purpose:** Lightweight deployment using Docker without complex configuration.
    - Complex:
        1. **Platforms:** Full cloud platforms (AWS, Azure, GCP).
        2. **Features:** Scaling by upgrading GPU instance type or running additional containers, public access via Streamlit, optional monitoring through Prometheus + Grafana.
3. Edge Deployment:
    - **Purpose:** Real-time inference on IoT or extremely low-latency environments.
    - **Tools:** NVIDIA Jetson, Raspberry Pi.

## Deployment Strategy

Deployment strategies ensure that the AI system is deployed in a secure, reliable, and scalable manner. This section outlines the approach used for deploying the model and describes the tools and best practices involved.

### Deployment Strategy Options:

1. Containerization
    - **Tools:** Docker
    - **Purpose:**
        1. Ensures consistent runtime environments
        2. Simplifies deployment on both local and cloud GPU machines
        3. Makes it easy to package the model + app together for portability

2. Orchestration

    - **Tools:** Docker Compose
    - **Purpose:**
        1. Docker Compose: Manages container setup and environment variables for simple deployments

3. Serverless Deployment

    - **Tools:** AWS Lambda
    - **Purpose:** Best for small, event-driven apps

## Security and Compliance in Deployment (Trustworthiness and Risk Management)

1. **Security and Compliance Best Practices:**
    - Security Measures:
        1. Use minimal base images (e.g., Python-slim) to reduce attack surface and vulnerabilities.
        2. Run Docker containers as non-root users to prevent privilege escalation.
        3. Store sensitive configuration in environment variables, not in code or the image.
        4. Limit access to uploaded files by storing them temporarily and clearing them after use.
        5. Restrict file system permissions so only the application can read/write model and dataset files.
        6. Use HTTPS if hosted and exposed publicly to ensure secure communication with users.
    - Compliance Measures:
        1. Maintain audit logs using Docker logs or Prometheus/Grafana dashboards to track system behavior.
        2. Ensure transparent documentation of data sources, preprocessing steps, and model changes for accountability.
        3. Follow minimal data retention by clearing temporary user uploads after use.
        4. Apply safe-response prompting to comply with safety guidelines and prevent harmful outputs.
        5. Version control (Git) ensures reproducibility and traceability of model updates and deployments.

## CI/CD for Deployment Automation

1. Containerized Build and Orchestration:
    - **Tools:** Docker, Docker Compose, NVIDIA Container Toolkit
    - **Purpose:** Automates the complete environment setup. It defines the Infrastructure as Code via Dockerfile and docker-compose.yml ensuring that the GPU drivers, Python libraries, and network ports are configured identically across development and deployment environments without human intervention.
2. Version Control & Manual Rollback:
    - **Tools:** Git, GitHub
    - **Purpose:** Maintains a history of all code and configuration changes. In the event of a model failure or UI bug in production, the system can be rolled back to the previous stable commit using git checkout followed by a Docker rebuild, ensuring minimal downtime.
3. Continuous Monitoring and Observability:
    - **Tools:** Prometheus, Grafana, Python prometheus-client
    - **Purpose:** The system implements a real-time observability stack. Prometheus scrapes custom metrics exposed by the application (specifically user satisfaction via "Thumbs Up/Down" feedback and total query counts). Grafana visualizes these metrics in dashboards, allowing administrators to detect performance degradation or model hallucination trends based on user reporting.

## Testing in the Deployment Environment

1. Testing Options:
    - Manual Testing:
        1. **Tools:** Streamlit interface, Docker logs, Prometheus/Grafana metrics.
        2. **Purpose:** Validate model responses, UI behavior, and system stability in a production-like environment.
    - Automated Testing:
        1. **Tools:** Basic scripted prompt tests or health-check scripts inside the Docker container.
        2. **Purpose:** Quickly validate model loading, endpoint availability (if added later), and consistent system behavior.

# Evaluation, Monitoring and Maintenance Plan

## Documentation:

## Monitoring Tools Selected:

1. **Prometheus:** Used as the time-series database to scrape and store metrics from the application containers via the prometheus-client library.

2. **Grafana:** Used for visualization to create real-time dashboards that display latency, retrieval scores, and feedback trends.

## Metrics Tracked:

1. Model Performance and Latency:
    - llm_responses_total: Counter tracking total generation volume, labeled by mode (LLM vs. RAG).
    - request_processing_seconds: Summary tracking the end-to-end latency of response generation.
    - response_length_chars: Gauge measuring verbosity and output length stability.
    - user_feedback_total: Counter tracking user satisfaction (Thumbs Up/Down), serving as the primary quality signal.
2. RAG & Retrieval Effectiveness:
    - rag_retrieval_seconds: Summary tracking the latency of the vector search
    - rag_similarity_score: Histogram tracking vector distances. Lower scores indicate high semantic relevance, high scores indicate potential "hallucination" risks due to poor context.
    - rag_context_hits_total: Tracks the hit rate of successful context retrieval.
3. Health & Ingestion:
    - ingestion_docs_total: Counter for successful PDF uploads.
    - ingestion_errors_total & ingestion_large_files_total: Tracks pipeline failures and edge cases.
    - rag_index_freshness: Unix timestamp ensuring the knowledge base is up-to-date.

## Drift Detection:

- **Method:** Quantitative Drift Scoring.
- **Metric:** rag_data_drift_score
- **Outcome:** This gauge measures how much current embeddings deviate from the original baseline. If the drift becomes significant, it signals that the knowledge base is becoming outdated and may need to be refreshed.

## Monitoring Options:

1. Performance Monitoring:
    - **Tools:** Prometheus, Grafana
    - **Implementation:** The system exposes a /metrics endpoint on port 8000 using the start_metrics_server function. Prometheus scrapes this target every 15 seconds. Grafana visualizes the LATENCY_SUMMARY and RETRIEVAL_LATENCY to identify bottlenecks between the GPU inference time vs. the Vector DB lookup time.
2. Drift Detection:
    - **Purpose:** Identify changes in data distribution or model performance over time.
    - **Implementation: Hybrid Detection**
        1. **Metric-Based:** The rag_data_drift_score gauge provides a numerical value representing the semantic shift in user queries vs. stored documents.
        2. **Feedback-Based:** The user_feedback_total counter serves as a ground-truth check. A divergence where rag_similarity_score remains good but user_feedback_total (Negative) increases indicates "Concept Drift" (the model is confident but wrong).

## Feedback Collections and Continuous Improvement:

## Documentation:

The system includes simple feedback options within the Streamlit interface, allowing users to rate responses or leave short comments. Runtime behavior such as latency, resource usage, and performance patterns is monitored through Prometheus and visualized in Grafana dashboards, helping identify areas for improvement.

**Feedback Options:**

1. Basic Feedback:
    - Built-in feedback buttons(Thumbs up and Thumbs down) or text boxes within the Streamlit UI
    - Allows users to indicate whether a response was helpful or provide comments
    - Helps identify weak responses and refine prompting or model configuration
2. Advanced Analytics:
    - **Tools:** Prometheus, Grafana
    - **Implementation:** Instead of external behavior tracking tools (like Hotjar), the system relies on **Correlated Metric Analysis**. By visualizing user_feedback_total alongside operational metrics like rag_similarity_score and request_processing_seconds, can infer root causes of dissatisfaction distinguishing between "Slow Performance" (Latency issues) vs. "Bad Answers" (Low Similarity/Hallucination).

## Maintenance and Compliance Audits:

## Documentation:

The system uses a simple maintenance workflow: manual checks are performed periodically to update dependencies, refresh the model configuration if needed, and review user feedback for quality issues. Prometheus and Grafana are used to monitor latency, CPU/GPU usage, and response stability. These metrics help identify performance degradation or emerging risks. Routine reviews ensure safe-response prompting and PII filtering are functioning correctly. Adjustments such as prompt refinement or parameter tuning are made based on observed behavior.

**Maintenance Options:**

1. Scheduled Maintenance:
    - **Simple:**
        1. Manual updates for dependencies, Docker images, and model configuration
        2. Periodic checks of logs, performance metrics, and user feedback
        3. Manual retraining or fine-tuning when needed
    - **Advanced:**
        1. Automated workflows (e.g., cron jobs or simple scripts)
2. Compliance Audits:
    - **Manual:**
        1. Periodic review of dataset sources, preprocessing steps, and PII filtering
        2. Checks for safe-response behavior and model output stability
        3. Verifying that logs and monitoring dashboards remain accurate
    - **Automated:**
        1. Basic alerts through Prometheus for performance anomalies

## Model Updates and Retraining:

## Documentation:

**Document the retraining process and version control strategies:**

The model is retrained manually whenever new training samples or refined instructions are added. The dataset (processed_data.jsonl) and LoRA configuration files are version-controlled using Git, allowing each update to be tracked. Retraining involves running the LoRA fine-tuning script again with the updated dataset and generating a new version of the model weights.

**Highlight any challenges and solutions implemented:**

Because the model runs in a quantized format, ensuring compatibility during retraining required careful management of tokenizer versions and LoRA configuration. Small dataset size was addressed by iterative feedback-based updates, where weak responses identified during deployment were added back as corrected training samples.

**Model Update Options:**

1. Manual Retraining:
    - Periodically retrain the model using updated or expanded Dolly-style samples
    - Add corrected examples based on user feedback to reduce hallucinations or weak responses
    - Version datasets and LoRA weight files through Git for reproducibility
2. Automated Pipelines:
    - Automation could be added in the future using cron jobs or basic scripting

**Github Repository:** https://github.com/ajit1203/QuizCatalyst_AIS.git