Lebanese University
Faculty of Engineering III
Electrical and Electronic Department

# MINI PROJECT

## Hybrid AI-Powered Adaptive Quiz System

**Instructor:** **Dr. Mohammad Aoude**

**Team:** **Rokaya Al Harakeh** **6441**
**Jana Jouni** **6348**

**2024- 2025**

# TABLE OF CONTENTS

# CHAPTER I: GENERAL INTRODUCTION

Traditional quiz systems typically fail to address individual learning differences. This static approach hinders engagement: advanced users find the content too easy, while beginners are discouraged by complexity. Moreover, instructors lack the time to personalize quizzes for every learner.

Our adaptive quiz system leverages both deterministic rule-based logic and AI-powered feedback to dynamically adjust question difficulty in real-time. Developed with Streamlit for the frontend and integrated with a GPT-based explanation engine, the system offers a personalized experience. A backend agent controls the adaptive logic, while question difficulty is pre-processed using machine learning (clustering based on semantic embeddings).

Additionally, this system features user authentication, performance tracking, and optional AI-generated feedback, all within a clean, Dockerized deployment.

# CHAPTER II: TECHNICAL STACK OVERVIEW

## 1- Frontend

We used Streamlit, a Python-based UI framework, to create an intuitive, clean, and responsive interface. This allowed us to rapidly prototype and deploy features like user login, quiz interaction, and performance dashboards without needing traditional HTML/CSS/JS stacks.

## 2- Backend Logic – Python 3.9

Python powers the core logic of the application. This includes the rule-based adaptive agent, quiz flow management, user data handling, and AI model integration. Its simplicity and rich ecosystem made it ideal for rapid development.

## 3- Databases – SQLite and CSV

User credentials and performance data are stored securely using SQLite, which is lightweight yet powerful enough for a multi-user educational tool. Quiz questions are stored in editable CSV files for flexibility. This hybrid model balances security with ease of content updates.

## 4- AI/NLP Stack – SentenceTransformers + scikit-learn

We utilized the pre-trained `all-MiniLM-L6-v2` model from SentenceTransformers to generate dense vector embeddings of question texts. These embeddings were then clustered using KMeans (from scikit-learn) to classify questions into three difficulty levels (Easy, Medium, Hard).

## 5- Deployment – Docker

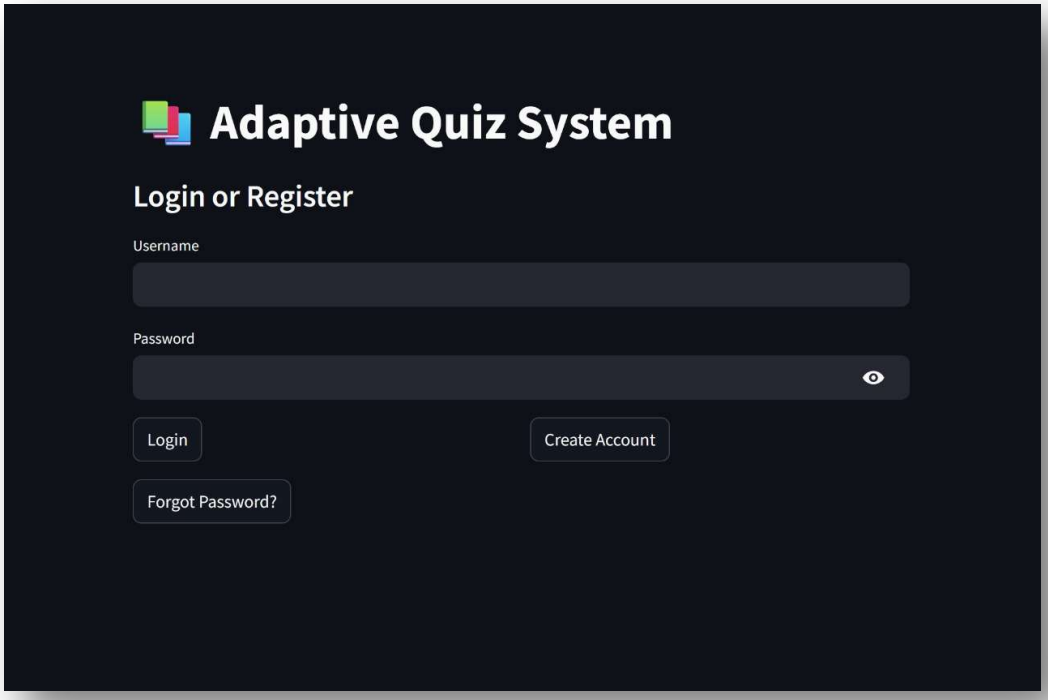
Docker was used to containerize the entire application. This ensures the app can run consistently on any machine without setup conflicts, and allows us to ship a “plug-and-play” educational tool.

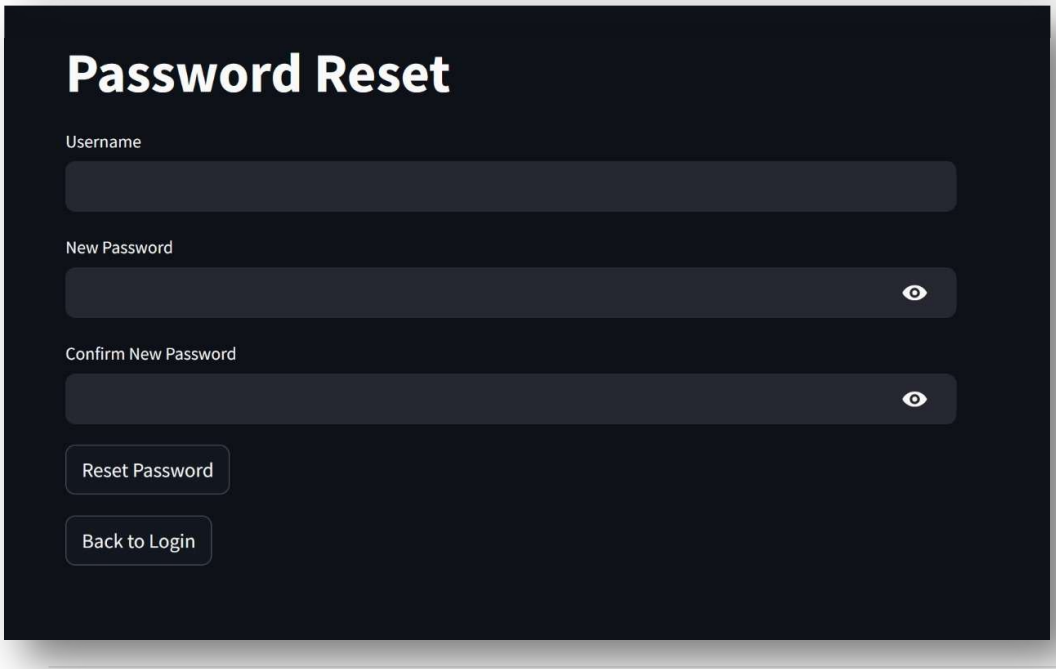
# CHAPTER III: SYSTEM DESIGN FLOW

## A. User Flow (Frontend Interaction)

1. User Login or Registration

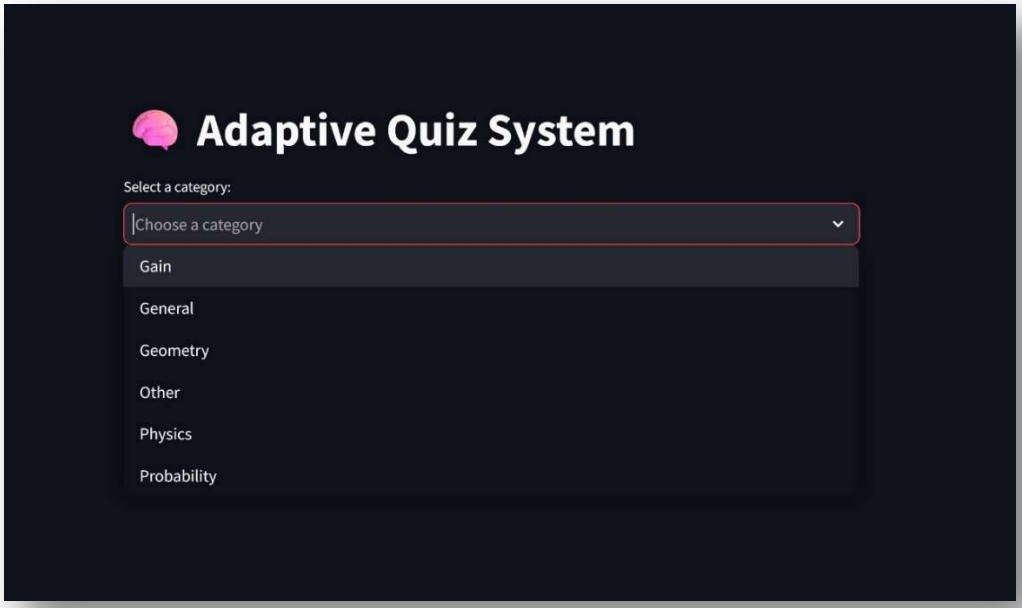
Credentials are verified through a local SQLite database.



2. Category Selection

User chooses a subject area from the available list.

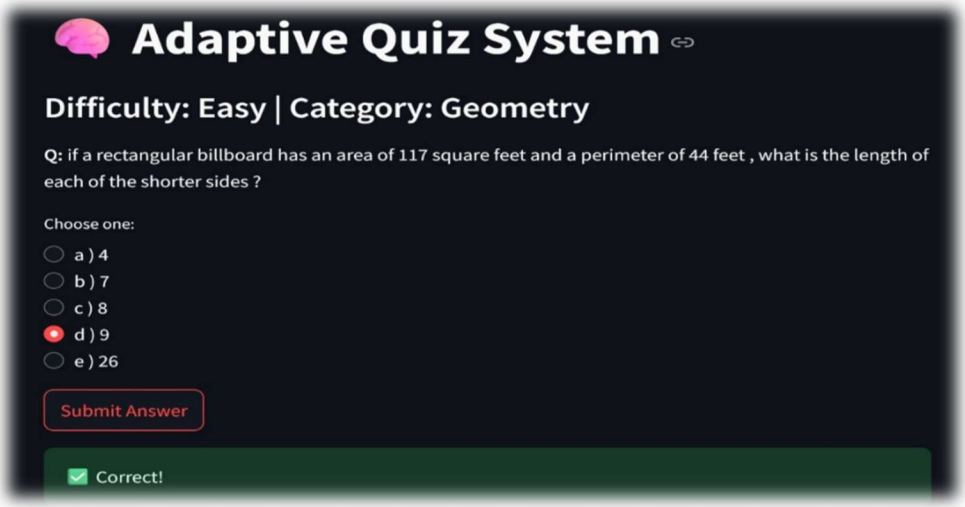


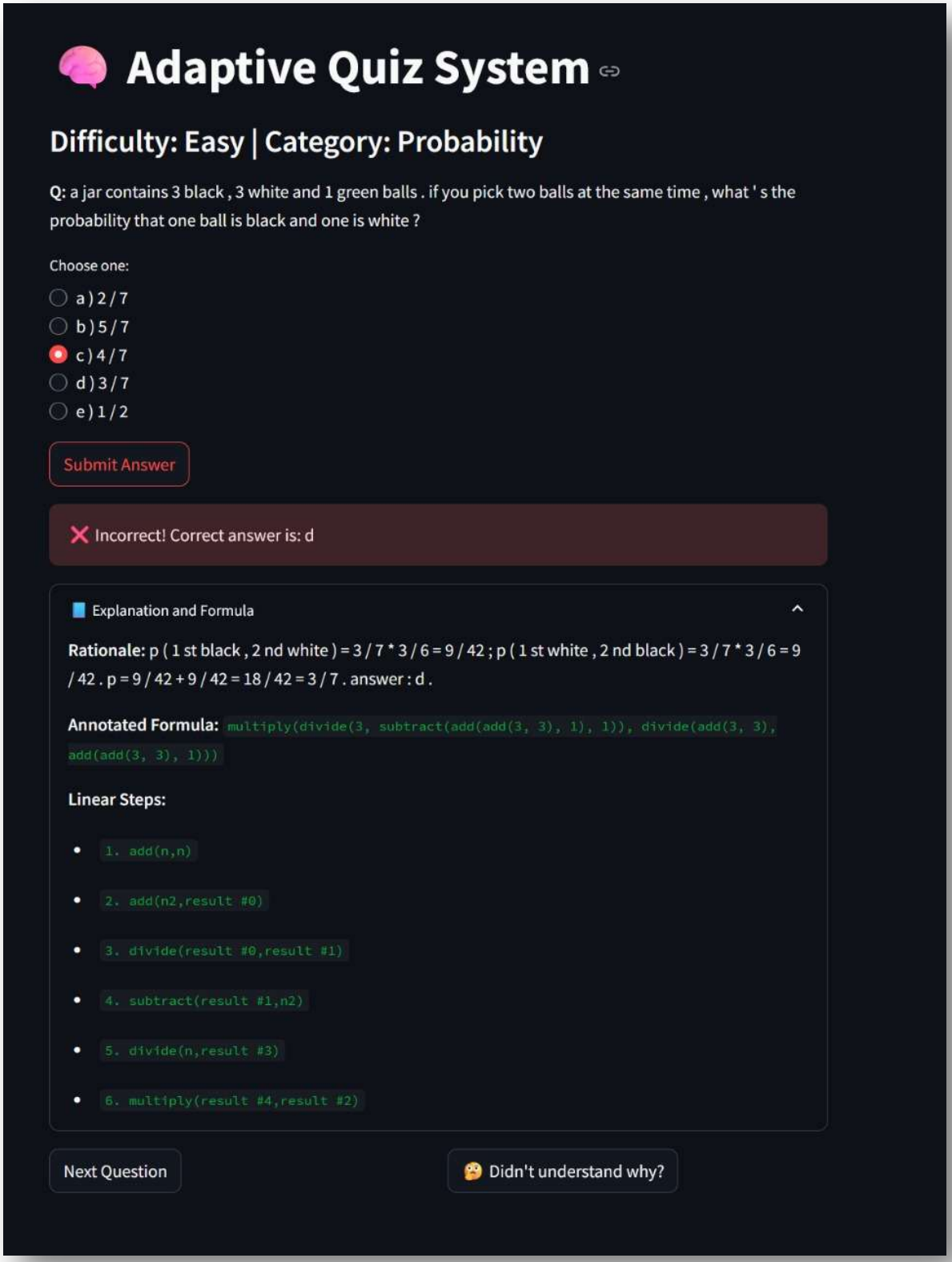

3. Quiz Interaction Begins

A question is presented based on current difficulty level.

4. Answer Submission & Evaluation

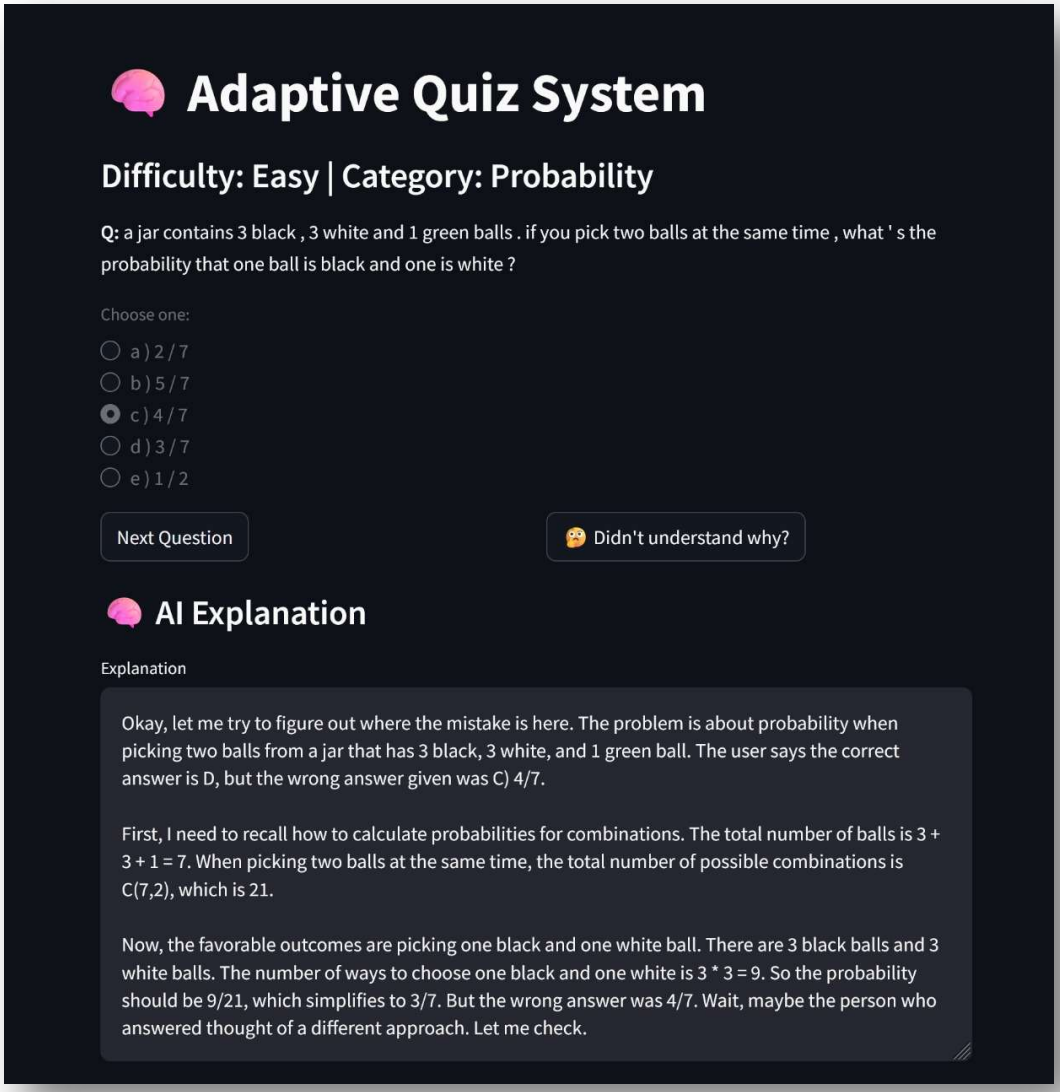
The system checks correctness and updates the user's streak.




Adaptive Quiz System
Difficulty: Easy | Category: Probability
Q: a jar contains 3 black , 3 white and 1 green balls . if you pick two balls at the same time , what ' s the probability that one ball is black and one is white ?
Choose one:
a ) 2 / 7
b ) 5 / 7
c ) 4 / 7
d ) 3 / 7
e ) 1 / 2
Submit Answer
Incorrect! Correct answer is: d
Explanation and Formula
Rationale: p ( 1 st black , 2 nd white ) = 3 / 7 * 3 / 6 = 9 / 42 ; p ( 1 st white , 2 nd black ) = 3 / 7 * 3 / 6 = 9 / 42 . p = 9 / 42 + 9 / 42 = 18 / 42 = 3 / 7 . answer : d .
Annotated Formula: multiply(divide(3, subtract(add(add(3, 3), 1), 1)), divide(add(3, 3), add(add(3, 3), 1)))
Linear Steps:
1. add(n,n)
2. add(n2,result #0)
3. divide(result #0,result #1)
4. subtract(result #1,n2)
5. divide(n,result #3)
6. multiply(result #4,result #2)
Next Question
Didn't understand why?

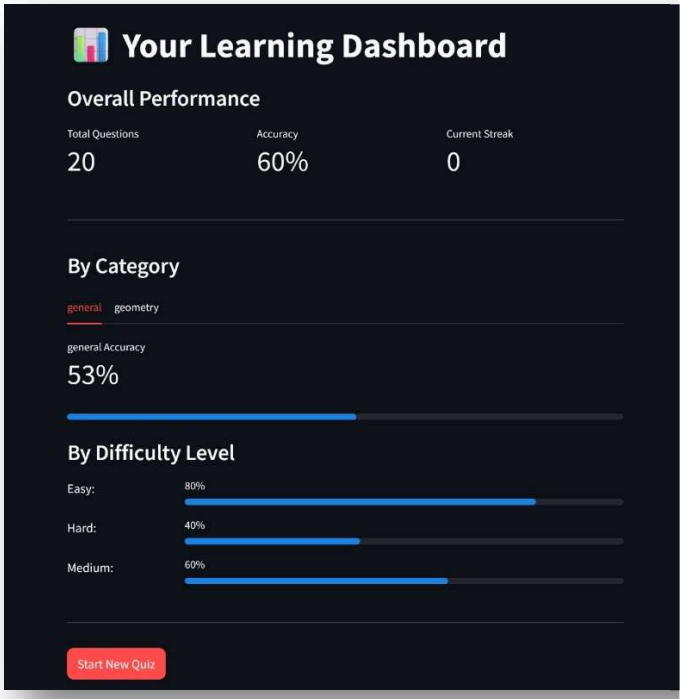
5. Explanation (if needed)

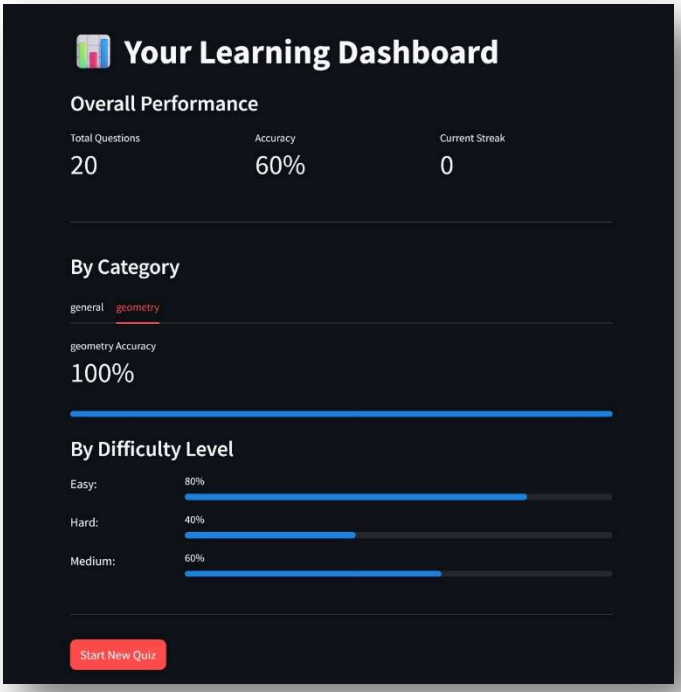
For incorrect answers, users can request an AI-generated explanation.



6. Next Question or Dashboard View

User continues the quiz or views performance stats.




## B. System Flow (Backend Logic)

**User Action → Quiz Engine → Rule-Based Agent → Question Selector**

↓ ↑

**Performance Log** **Difficulty-Tagged Dataset**

↓ ↓

**SQLite DB** **Embeddings + KMeans Clustering**

**STEPS:**

1) **Initialization**

- On startup, the system reads quiz questions from CSV files.
- SentenceTransformers generates question embeddings.
- KMeans clusters these into three difficulty groups.

2) **Quiz Engine**

   Manages session state, monitors streaks, and interacts with the adaptive agent.

3) **Adaptive Agent**

   Implements a transparent rule system to decide if the difficulty should increase, decrease, or stay the same.

4) **Question Selection**

   Based on agent's output, a matching question is drawn from the clustered dataset.

5) **Performance Logging**

   Each interaction (category, difficulty, correctness) is saved for analysis.

6) **AI Explanation (Optional)**

   If a user answers incorrectly and requests help, the system sends the question, wrong answer, and correct answer to the `AIExplainer`, which returns a short explanation using an external LLM API

# CHAPTER IV: ETHICAL CONSIDERATIONS

- **Data Privacy**

  User data (login info and performance) is stored locally in a protected SQLite database. Input fields are sanitized, and sensitive operations (like password updates) follow secure practices. This ensures that user identity and performance are not exposed or misused.

- **Fairness & Bias Avoidance**

  To avoid human bias in question difficulty, clustering was done using an unsupervised ML model. No manual tags or subjective human labeling was used to rank question difficulty. This promotes fairness and consistent standards for all users.

- **Transparency & Control**

  The system never makes irreversible decisions or hides logic from users. Every adaptive action (e.g., difficulty change) is based on clear rules that mimic how a human tutor might adjust a quiz. This makes the system explainable and user-friendly.

- **AI Disclosure**

  When explanations are generated by the language model, they are clearly marked as "AI Explanation." Users know which content is machine-generated versus rule-based.

- **Educational Integrity**

  Explanations are brief and tailored to understanding mistakes, not providing full solutions upfront. This supports learning rather than encouraging shortcut behavior

---

# CHAPTER V: DUAL AI IMPLEMENTATION APPROACH

## A. Rule-Based Adaptive Agent (Symbolic AI)

This agent simulates a logical decision-maker:

- It tracks how many questions a user gets right or wrong consecutively (the "streak").
- Based on predefined rules (e.g., **3 correct → Hard**, **2 wrong → Easy**), it adjusts the next question's difficulty.
- It ensures users are neither overwhelmed nor bored, keeping the quiz engaging.

**Benefits:**

- Fully explainable: students and instructors can understand how decisions are made.
- Lightweight and deterministic: no randomness or black-box behavior.

---

## B. Machine Learning – Clustering with KMeans (Subsymbolic AI)

Instead of manually tagging each question's difficulty, we let the machine learn it:

- Every question is embedded using SentenceTransformers into a 384-dimensional vector.
- KMeans clustering then groups these vectors into 3 clusters.
- Based on the centroid distances, we interpret them as Easy, Medium, and Hard.

**Benefits:**

- Automatic difficulty labeling of new questions.
- Scales to large datasets without manual work.
- Captures semantic and conceptual complexity beyond surface-level features.

### Why Combine Both?

- Symbolic AI (rules) gives real-time adaptiveness and control.
- Subsymbolic AI (ML) ensures intelligent question grouping without bias.
- Together, they replicate a human tutor's behavior more accurately and flexibly.

# CHAPTER VI: TESTING AND EVALUATION

## A. UNIT AND MODULAR TESTING

Each major module was independently tested:

- Authentication Module:
  Tested for case-insensitive usernames, secure password handling, and reset logic.
- AI Agent Logic:
  Simulated various streaks to ensure correct difficulty adjustments. Edge cases (e.g., empty dataset, repeated wrong answers) were handled gracefully.
- API Communication:
  The AI explanation system was tested for:
  - Invalid API key handling
  - Network failures
  - Unexpected API response formats

## B. End-to-End Testing

We conducted full simulations of the user flow:

1. Login/Registration
2. Category selection
3. Quiz attempt with answers
4. Feedback and difficulty scaling
5. Viewing performance dashboard

Different user types were simulated:

- **Beginner** → Starts at Easy, improves slowly
- **Advanced** → Moves quickly to Hard
- **Random** → Streak fluctuates, difficulty oscillates

Each scenario showed smooth transitions and accurate tracking.

### C. Manual QA and UX Testing

- **Interface Testing:**
  Streamlit components were tested across multiple browsers. Responsive design ensured readability and usability.
- **User Feedback:**
  Informal testers reported that the quiz was clear, the feedback helpful, and the AI-generated explanations short but useful.
- **Data Logging:**
  Performance logs were reviewed to ensure consistency across sessions and accuracy in user stats.

# CHAPTER VII: CONCLUSION & FUTURE WORK

## a) Conclusion

This project demonstrates the practical value of combining symbolic and subsymbolic AI for adaptive learning. By using a rule-based agent to adapt quiz difficulty and ML clustering to classify questions by complexity, we created a smart, fair, and scalable quiz system.
The use of Streamlit and Docker made development fast and deployment simple, ensuring accessibility. The system is also highly extendable, with future directions including:

- User-level personalization using ML
- Integration of analytics dashboards
- Support for audio/visual questions
- Real-time difficulty visualization

This hybrid AI-driven approach sets a foundation for future educational systems that respond intelligently and fairly to student needs.

## b) Future Work

This project lays a solid foundation, but several improvements and extensions can be explored:

- Timed quizzes + topic streaks – Adaptive difficulty per topic.
- Voice support – TTS for question reading or GPT explanations.
- Gamification layer – Visual badges, streak awards, category mastery.
- Multilingual support – Translate via AI APIs for inclusivity.

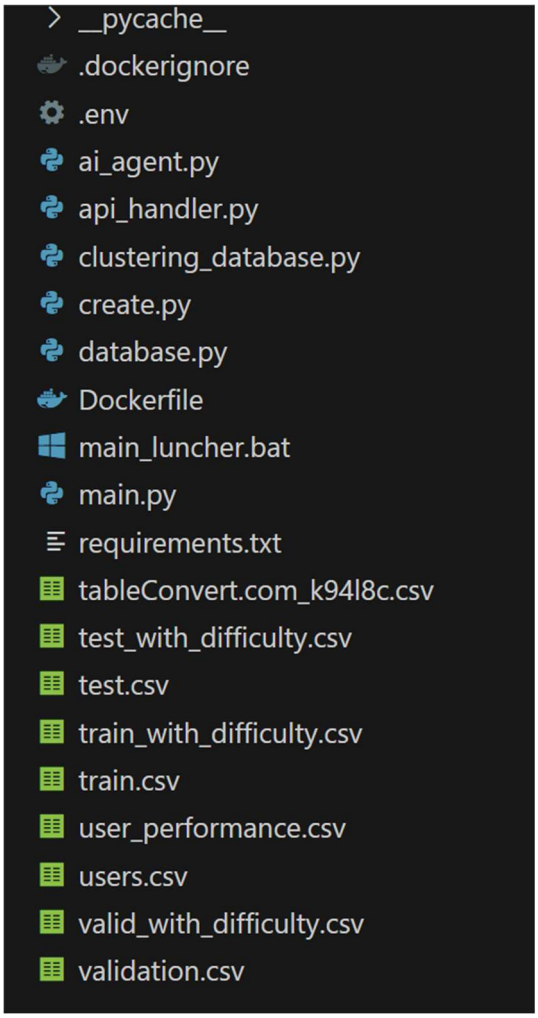
# APPENDIX

## ✍ GitHub Repository

All source code, data, plots, Dockerfile, and documentation are hosted publicly on GitHub to support collaboration, version control, and open access.

✅ **GitHub Link**: https://github.com/JanaJouni/mini-project.git

## ✍ Directories



✍ **The code of the project:**

**AI AGENT**

```
import random

class AdaptiveQuizAgent:
    def __init__(self, streak, difficulty):
        self.streak = streak
        self.current_difficulty = difficulty

    def sense(self, correct):
        """Receives whether the last answer was correct."""
        return {"correct": correct}

    def think(self, input_data):
        """Updates the internal state (streak and difficulty) based on the answer
correctness."""
        if input_data["correct"]:
            self.streak += 1
        else:
            self.streak -= 1

        # Adjust difficulty based on streak
        if self.streak >= 3:
            self.current_difficulty = "Hard"
        elif self.streak <= -2:
            self.current_difficulty = "Easy"
        else:
            self.current_difficulty = "Medium"

        return {"difficulty": self.current_difficulty, "streak": self.streak}

    def act(self, action, questions_df):
        """Selects a question of the appropriate difficulty."""
        available = questions_df[questions_df["difficulty"] == action["difficulty"]]
        if available.empty:
            return questions_df.sample(1).iloc[0]  # fallback
        return available.sample(1).iloc[0]

    def learn(self, experience):
        """Optional learning hook - currently logs experience."""
        print(f"Learning from experience: {experience}")

    def select_question_index(self, questions_df):
        """Returns a random index from the DataFrame (used to fetch the next question)."""
        if questions_df.empty:
            return None
        return random.choice(questions_df.index.tolist())
```

## API HANDLER

```
import os
from dotenv import load_dotenv
import requests

load_dotenv()

class AIExplainer:
    def __init__(self):
        self.api_key = os.getenv("OPENROUTER_API_KEY")
        self.base_url = "https://openrouter.ai/api/v1"

    def generate_explanation(self, problem: str, user_answer: str, correct_answer: str) ->
str:
        if not self.api_key:
            return "🔑 Error: API key not configured in .env file"

        headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json",
            "HTTP-Referer": "http://localhost",
            "X-Title": "adaptive-quiz-app"
        }

        payload = {
            "model": "deepseek/deepseek-r1:free",
            "messages": [
                {"role": "system", "content": "You are a math tutor. Explain mistakes in 2
sentences."},
                {
                    "role": "user",
                    "content": (
                        f"Problem: {problem}\n"
                        f"Wrong Answer: {user_answer}\n"
                        f"Correct Answer: {correct_answer}\n"
                        "Explain the mistake:"
                    )
                }
            ],
            "temperature": 0.3,
            "max_tokens": 2000,
        }

        try:
            response = requests.post(
                f"{self.base_url}/chat/completions",
                headers=headers,
                json=payload,
                timeout=10
            )
            response.raise_for_status()
            response_json = response.json()
            msg = response_json["choices"][0]["message"]
            explanation = msg.get("content") or msg.get("reasoning") or "⚠ No explanation
provided."
            return explanation.strip()
```

```
        except requests.exceptions.RequestException as e:
            return f"⚠ API Error: {str(e)}"
        except KeyError:
            return "⚠ Error: Unexpected API response format"
```

## CLUSTERING DATABASE

```
import pandas as pd
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans
import numpy as np

train_path = 'C:/codes/train.csv'
valid_path = 'C:/codes/validation.csv'
test_path = 'C:/codes/test.csv'

train_out = 'C:/codes/train_with_difficulty.csv'
valid_out = 'C:/codes/valid_with_difficulty.csv'
test_out = 'C:/codes/test_with_difficulty.csv'

# Load CSVs
train_df = pd.read_csv(train_path)
valid_df = pd.read_csv(valid_path)
test_df = pd.read_csv(test_path)

# Print columns to verify column names
print("Train CSV columns:", train_df.columns.tolist())
print("Valid CSV columns:", valid_df.columns.tolist())
print("Test CSV columns:", test_df.columns.tolist())

# Load sentence transformer model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Function to get embeddings from 'Problem' column
def get_embeddings(df):
    if 'Problem' in df.columns:
        texts = df['Problem'].tolist()
    else:
        raise ValueError("CSV does not contain 'Problem' column")
    return model.encode(texts, show_progress_bar=True)

# Get embeddings for train, valid, test sets
print("Embedding training problems...")
train_embeds = get_embeddings(train_df)

print("Embedding validation problems...")
valid_embeds = get_embeddings(valid_df)

print("Embedding test problems...")
test_embeds = get_embeddings(test_df)

# Cluster training set embeddings with KMeans
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(train_embeds)

# Map clusters to difficulty by centroid norms heuristic
centroids = kmeans.cluster_centers_
centroid_norms = np.linalg.norm(centroids, axis=1)
difficulty_order = np.argsort(centroid_norms)  # smallest norm = easiest?
```

```
difficulty_map = {
    difficulty_order[0]: 'Easy',
    difficulty_order[1]: 'Medium',
    difficulty_order[2]: 'Hard'
}

# Assign difficulty labels to train set
train_labels = kmeans.labels_
train_df['difficulty'] = [difficulty_map[label] for label in train_labels]

# Predict difficulty on valid and test sets
valid_labels = kmeans.predict(valid_embeds)
valid_df['difficulty'] = [difficulty_map[label] for label in valid_labels]

test_labels = kmeans.predict(test_embeds)
test_df['difficulty'] = [difficulty_map[label] for label in test_labels]

# Save updated CSVs with difficulty labels
train_df.to_csv(train_out, index=False)
valid_df.to_csv(valid_out, index=False)
test_df.to_csv(test_out, index=False)

print("Difficulty labeling complete and saved:")
print(f"- {train_out}")
print(f"- {valid_out}")
print(f"- {test_out}")
```

**CREATE**

```
import pandas as pd
import os
import streamlit as st
from datetime import datetime

USER_DB = "users.csv"
PERFORMANCE_DB = "user_performance.csv"

class data:
    @staticmethod
    def init_dbs():
        """Initialize databases with proper string types"""
        if not os.path.exists(USER_DB):
            pd.DataFrame(columns=["username", "password"]).astype(str).to_csv(USER_DB,
index=False)
        if not os.path.exists(PERFORMANCE_DB):
            pd.DataFrame(columns=[
                "username", "date", "category", "difficulty",
                "correct", "total", "streak"
            ]).astype(str).to_csv(PERFORMANCE_DB, index=False)

    @staticmethod
    def authenticate_user(username, password):
        """Verify credentials with proper string conversion"""
        try:
            data.init_dbs()
            df = pd.read_csv(USER_DB).astype(str)  # Ensure all data is string
```

```
        # Convert to lowercase and strip whitespace for comparison
        username = str(username).strip().lower()
        password = str(password).strip()

        # Find matching user
        match = df[
            (df["username"].str.strip().str.lower() == username) &
            (df["password"].str.strip() == password)
        ]

        if match.empty:
            st.error("Invalid username or password")
            return False
        return True

    except Exception as e:
        st.error(f"Login error: {str(e)}")
        return False

@staticmethod
def create_user(username, password):
    """Create new user with string validation"""
    try:
        data.init_dbs()
        df = pd.read_csv(USER_DB).astype(str)

        # Convert to strings and clean inputs
        username = str(username).strip()
        password = str(password).strip()

        # Check if username exists (case-insensitive)
        if df["username"].str.strip().str.lower().eq(username.lower()).any():
            st.warning("Username already exists")
            return False

        new_user = pd.DataFrame([{
            "username": username,
            "password": password
        }])

        pd.concat([df, new_user], ignore_index=True).to_csv(USER_DB, index=False)
        st.success("Account created successfully!")
        return True

    except Exception as e:
        st.error(f"Account creation failed: {str(e)}")
        return False

@staticmethod
def log_performance(username, category, difficulty, correct, streak):
    """Record quiz results"""
    df = pd.read_csv(PERFORMANCE_DB).astype(str)
```

```
        new_entry = pd.DataFrame([{
            "username": username,
            "date": datetime.now().strftime("%Y-%m-%d %H:%M"),
            "category": category,
            "difficulty": difficulty,
            "correct": int(correct),
            "total": 1,
            "streak": streak
        }])
        pd.concat([df, new_entry], ignore_index=True).to_csv(PERFORMANCE_DB, index=False)

    @staticmethod
    def get_user_stats(username):
        """Get performance metrics"""
        if not os.path.exists(PERFORMANCE_DB):
            return None

        # Read CSV without forcing all columns to strings
        df = pd.read_csv(PERFORMANCE_DB)

        # Convert specific columns to numeric types
        df["correct"] = pd.to_numeric(df["correct"], errors='coerce').fillna(0)
        df["total"] = pd.to_numeric(df["total"], errors='coerce').fillna(0)
        df["streak"] = pd.to_numeric(df["streak"], errors='coerce').fillna(0)

        user_data = df[df["username"] == str(username)]  # Ensure username comparison is
string

        if user_data.empty:
            return None

        total_correct = user_data["correct"].sum()
        total_questions = user_data["total"].sum()
        accuracy = total_correct / total_questions if total_questions > 0 else 0

        return {
            "total_questions": int(total_questions),
            "accuracy": accuracy,
            "current_streak": int(user_data.iloc[-1]["streak"]),
            "category_stats": user_data.groupby("category")["correct"].mean().to_dict(),
            "difficulty_stats": user_data.groupby("difficulty")["correct"].mean().to_dict()
        }



    @staticmethod
    def reset_password_page():
        """Simpler password reset without current password verification"""
        st.title("Password Reset")

        username = st.text_input("Username")
        new_password = st.text_input("New Password", type="password")
        confirm_password = st.text_input("Confirm New Password", type="password")

        if st.button("Reset Password"):
            if new_password != confirm_password:
                st.error("New passwords don't match!")
                return

            if data.update_password(username, new_password):
```

```
                st.success("Password updated successfully!")
            else:
                st.error("Failed to update password - username not found")

    @staticmethod
    def update_password(username, new_password):
        """Update user password in database"""
        try:
            df = pd.read_csv(USER_DB)

            # Convert both to lowercase and strip whitespace for comparison
            username_clean = str(username).strip().lower()
            df["username_clean"] = df["username"].str.strip().str.lower()

            # Find user and update password
            mask = df["username_clean"] == username_clean
            if not mask.any():
                return False

            # Update password and save
            df.loc[mask, "password"] = str(new_password).strip()
            df = df.drop(columns=["username_clean"])  # Remove temporary column
            df.to_csv(USER_DB, index=False)
            return True

        except Exception as e:
            st.error(f"Password update error: {str(e)}")
            return False
```

**DATABASE**

```
import pandas as pd

def load_questions():
    # Load from CSV or DB
    df = pd.read_csv("train_with_difficulty.csv")
    # Convert 'options' string to list if stored as stringified list
    df["options"] = df["options"].apply(eval)  # careful with eval, only if you trust the
data
    return df
```

**MAIN**

```
import streamlit as st
import pandas as pd
import ast
import re
from create import data
from ai_agent import AdaptiveQuizAgent
from api_handler import AIExplainer
from datetime import datetime

explainer = AIExplainer()

# --- Session Initialization ---
if "logged_in" not in st.session_state:
   st.session_state.logged_in = False
if "page" not in st.session_state:
   st.session_state.page = "login"
```

```
if "answered" not in st.session_state:
    st.session_state.answered = False
if "user_answer" not in st.session_state:
    st.session_state.user_answer = None
if "current_question" not in st.session_state:
    st.session_state.current_question = None
if "category" not in st.session_state:
    st.session_state.category = None
if "category_confirmed" not in st.session_state:
    st.session_state.category_confirmed = False
if "ai_explanation" not in st.session_state:
    st.session_state.ai_explanation = ""
if "correct" not in st.session_state:
    st.session_state.correct = False
if "show_dashboard" not in st.session_state:
    st.session_state.show_dashboard = True

# --- Utility Functions ---
def prettify_formula(expr):
    if not expr or expr == 'N/A':
        return "N/A"
    expr = re.sub(r'const_(\d+)', r'\1', expr)
    expr = expr.replace('*', '×').replace('/', '÷')
    expr = re.sub(r'([+\-×÷])', r' \1 ', expr)
    expr = re.sub(r'\s+', ' ', expr).strip()
    return expr

def prettify_rationale(text):
    if not text or text == 'N/A':
        return "N/A"
    return text.replace(',', ',\n').replace('.', '.\n')

def prettify_linear_steps(linear_raw):
    if not linear_raw or linear_raw == 'N/A':
        return ["N/A"]
    steps = linear_raw.strip('|').split('|')
    pretty_steps = []
    for i, step in enumerate(steps, 1):
        step = step.replace('n0', 'n')
        step = re.sub(r'const_(\d+)', r'\1', step)
        step = re.sub(r'#(\d+)', r'result #\1', step)
        pretty_steps.append(f"{i}. {step}")
    return pretty_steps

# --- Login Page ---
def login_page():
    st.title("🎴 Adaptive Quiz System")
    st.subheader("Login or Register")
```

```
    username = st.text_input("Username", key="login_username")
    password = st.text_input("Password", type="password", key="login_password")

    col1, col2 = st.columns(2)
    with col1:
        if st.button("Login"):
            if username and password:
                if data.authenticate_user(username.strip(), password.strip()):
                    st.session_state["username"] = username
                    st.session_state["logged_in"] = True
                    st.success("✅ Logged in!")
                    st.rerun()
                else:
                    st.error("❌ Invalid username or password.")
            else:
                st.warning("Please enter both username and password.")

    with col2:
        if st.button("Create Account"):
            if username and password:
                created = data.create_user(username.strip(), password.strip())
                if created:
                    st.session_state["username"] = username
                    st.session_state["logged_in"] = True
                    st.success("✅ Account created and logged in!")
                    st.rerun()
                else:
                    st.warning("⚠️ Username already exists.")
            else:
                st.warning("Please enter both username and password.")

    if st.button("Forgot Password?"):
        st.session_state.page = "reset"
        st.rerun()

# --- Performance Dashboard ---
def performance_dashboard():
    st.title("📊 Your Learning Dashboard")

    stats = data.get_user_stats(st.session_state["username"])
    if not stats:
        st.info("No performance data yet. Complete some quizzes first!")
        if st.button("Start Quiz"):
            st.session_state.show_dashboard = False
            st.rerun()
        return
```

```
    st.subheader("Overall Performance")
    col1, col2, col3 = st.columns(3)
    with col1:
        st.metric("Total Questions", stats["total_questions"])
    with col2:
        st.metric("Accuracy", f"{stats['accuracy']:.0%}")
    with col3:
        st.metric("Current Streak", stats["current_streak"])

    st.divider()

    st.subheader("By Category")
    tabs = st.tabs(list(stats["category_stats"].keys()))
    for tab, (category, accuracy) in zip(tabs, stats["category_stats"].items()):
        with tab:
            st.metric(f"{category} Accuracy", f"{accuracy:.0%}")
            st.progress(accuracy)

    st.subheader("By Difficulty Level")
    for difficulty, accuracy in stats["difficulty_stats"].items():
        col1, col2 = st.columns([1, 4])
        with col1:
            st.write(f"{difficulty}:")
        with col2:
            st.progress(accuracy, text=f"{accuracy:.0%}")

    st.divider()

    if st.button("Start New Quiz", type="primary"):
        st.session_state.show_dashboard = False
        st.rerun()

# --- Quiz Page ---
def quiz_page():
    st.title("🧠 Adaptive Quiz System")

    # Initialize agent
    if "agent" not in st.session_state:
        st.session_state.agent = AdaptiveQuizAgent(streak=0, difficulty="Easy")

    # Load questions
    try:
        questions_df = pd.read_csv("train_with_difficulty.csv").astype(str)
        questions_df = questions_df[
            (~questions_df['category'].isin(['', 'value'])) &
            (~questions_df['difficulty'].isin(['', 'Value']))
        ]
    except Exception as e:
```

```
        st.error(f"Data loading failed: {e}")
        return

    # Category selection
    if not st.session_state.category_confirmed:
        available_categories = sorted([c for c in questions_df['category'].unique() if c])
        category_display = st.selectbox(
            "Select a category:",
            [c.capitalize() for c in available_categories],
            index=None,
            placeholder="Choose a category"
        )
        if st.button("Proceed to Quiz"):
            if category_display:
                st.session_state.category = category_display.lower()
                st.session_state.category_confirmed = True
                st.rerun()
            else:
                st.warning("Please select a category to continue.")
        st.stop()

    # Load question with fallback
    if st.session_state.current_question is None:
        matched = questions_df[
            (questions_df['category'] == st.session_state.category) &
            (questions_df['difficulty'] == st.session_state.agent.current_difficulty)
        ]
        if len(matched) == 0:
            matched = questions_df  # Fallback to all questions

        try:
            st.session_state.current_question = matched.sample(1).iloc[0].to_dict()
        except:
            st.error("No questions available. Please check your data file.")
            st.stop()

    question = st.session_state.current_question

    # Display question
    st.markdown(f"### Difficulty: {question['difficulty']} | Category: {question['category'].capitalize()}")
    st.write(f"**Q:** {question['Problem']}")

    # Display options
    try:
        options = ast.literal_eval(question["options"]) if isinstance(question["options"], str) else
question["options"]
    except:
        options = [opt.strip() for opt in question["options"].split(",")]
```

```
# Answer submission
if not st.session_state.answered:
    st.session_state.user_answer = st.radio("Choose one:", options, key="answer_radio")
    if st.button("Submit Answer"):
        st.session_state.answered = True
        try:
            user_letter = re.match(r'^([a-zA-Z])[\)\.]?', str(st.session_state.user_answer)).group(1).upper()
            correct_letter = re.match(r'^([a-zA-Z])[\)\.]?', str(question["correct"])).group(1).upper()
            st.session_state.correct = user_letter == correct_letter
        except:
            st.session_state.correct = False

        # Update agent and log performance
        st.session_state.agent.think({"correct": st.session_state.correct})
        data.log_performance(
            username=st.session_state["username"],
            category=st.session_state.category,
            difficulty=st.session_state.agent.current_difficulty,
            correct=st.session_state.correct,
            streak=st.session_state.agent.streak
        )

        if st.session_state.correct:
            st.success("✅ Correct!")
        else:
            st.error(f"❌ Incorrect! Correct answer is: {question['correct']}")
# Show explanation
        with st.expander("📘 Explanation and Formula", expanded=True):
            rationale = prettify_rationale(question.get('Rationale', 'N/A'))
            annotated = prettify_formula(question.get('annotated_formula', 'N/A'))
            linear_steps = prettify_linear_steps(question.get('linear_formula', 'N/A'))

            st.markdown(f"**Rationale:**\n{rationale}")
            st.markdown(f"**Annotated Formula:** `{annotated}`")
            st.markdown("**Linear Steps:**")
            for step in linear_steps:
                st.markdown(f"- `{step}`")
else:
    try:
        index = options.index(st.session_state.user_answer)
    except ValueError:
        index = 0
    st.radio("Choose one:", options, index=index, disabled=True)
```

```
    # Bottom buttons
    col1, col2 = st.columns(2)
    with col1:
        if st.button("Next Question"):
            st.session_state.answered = False
            st.session_state.user_answer = None
            st.session_state.ai_explanation = ""
            st.session_state.current_question = None
            st.rerun()
    with col2:
        if st.session_state.answered and not st.session_state.correct:
            if st.button("🤔 Didn't understand why?"):
                with st.spinner("Generating AI explanation..."):
                    try:
                        response = explainer.generate_explanation(
                            problem=question["Problem"],
                            user_answer=st.session_state.user_answer,
                            correct_answer=question["correct"]
                        )
                        st.session_state.ai_explanation = response
                    except Exception as e:
                        st.session_state.ai_explanation = f"⚠️ Error: {str(e)}"

    # Display AI explanation
    if st.session_state.get("ai_explanation"):
        st.markdown("### 🧠 AI Explanation")
        st.text_area("Explanation", st.session_state.ai_explanation, height=300)

# --- Routing ---
if st.session_state.page == "reset":
    data.reset_password_page()
    if st.button("Back to Login"):
        st.session_state.page = "login"
        st.rerun()
elif st.session_state.logged_in:
    if st.session_state.get("show_dashboard", True):
        performance_dashboard()
    else:
        quiz_page()
else:
    login_page()
```