Problem Statement 1:

“AI powered voice enabled universal code debugger and explainer”

Build an intelligent, interactive platform that acts as a Universal Code Debugger and Explainer, capable of live step-by-step debugging, code explanation, and real-time tutoring using voice input/output across multiple programming languages.

3 CORE TASKS:

Task 1: Multi-language Code Debugger Engine (Backend)

Goal: Build the backend engine that can parse, run, and debug code in real time across multiple languages (e.g., Python, JavaScript, C++).

- Secure sandbox environment to execute user-submitted code
- Step-by-step debugger with state inspection (variables, stack, etc.)
- Support for at least 2 programming languages
- API for sending code and receiving step-by-step execution state

Voice-based error explainer:

- Integrate a **voice assistant** that detects and **explains errors or exceptions aloud** in natural language.
- Example: “There’s a Type Error on line 8 — you’re trying to add a string and an integer.”
- Offer possible **fix suggestions** (e.g., “You can convert the string to an integer using `int()`.”).

Task 2: Code Explainer & AI Tutor (AI + Logic Layer)

Goal: Implement an AI-powered module that explains code logic and helps users understand each step like a human tutor.

- Line-by-line code explanation engine
- Natural language responses for “What does this code do?”, “Why is there an error?”
- Integration with debugger state (from Task 1) for contextual explanation
- Optionally: AI-generated suggestions or hints

Task 3: Interactive Frontend with Voice Support (UI/UX)

Goal: Design a user-friendly interface with voice input/output to create a real-time tutoring experience.

- Web or desktop frontend interface

- Code editor with live execution view and step debugger UI
- Voice input (speech-to-text) and output (text-to-speech) integration
- Live session view showing code, variable state, and AI tutor responses
- **Skills Involved:** Frontend development (React, Vue, etc.), Web APIs, TTS/STT integration (e.g., Web Speech API, Whisper, Google TTS)

FINAL DELIVERABLES

1. Functional Prototype (Core Platform)

A working application (web or desktop) that:

- Accepts code in at least two programming languages (e.g., Python, JavaScript)
- Allows users to **run and debug code step-by-step.**
- Shows **real-time variable tracking and execution flow.**
- Supports **voice commands and voice responses** (STT + TTS).
- Provides **AI-generated code explanations** and tutoring-like feedback.

2. Technical Components

Backend (Task 1 Deliverables)

- Secure sandboxed code execution engine
- Step debugger with state tracking
- REST or WebSocket API for code submission and execution feedback

AI/Logic Layer (Task 2 Deliverables)

- Integration with an AI model (e.g., OpenAI, Claude) for:
    - Line-by-line code explanation
    - Answering “why” and “how” questions about the code
- Contextual analysis of runtime state for better explanations

Bonus (for Extra Points)

- Multi-language support beyond 2 languages
- Real-time collaboration (multiple users)
- Custom AI tutor personas or voice choices
- Adaptive learning features (track user's progress)

Problem Statement 2:

“AI-Powered Contextual PDF Query Assistant”

Domain:

Artificial Intelligence | Natural Language Processing | Information Retrieval

Problem Overview:

Students and professionals often deal with long documents or notes 20+ pages of dense information and when they need a specific answer, traditional AI tools **summarize**, **paraphrase**, or **hallucinate**, instead of giving the **exact referenced answer** from the notes.

This leads to loss of accuracy and trust in AI systems used for academic or technical purposes.

Your challenge is to develop an **AI-driven PDF Query System** that can **extract exact answers** from uploaded notes or documents not rewrite or summarize them and present the **precise content as it appears in the source file.**

Core Challenge:

Build a system that allows users to:

1. Upload large PDFs (lecture notes, research papers, manuals).
2. Ask questions in natural language.
3. Get **accurate, context-based answers** directly from the content — without rewording.
4. Optionally, display the **page number and snippet location** where the answer was found.

Constraints:

- The system must **not translate or summarize** the text.
- It should return **verbatim answers** as found in the PDF.
- Must handle **multi-page** PDFs efficiently (20+ pages minimum).
- Should respond within a **reasonable time (<10 seconds)** for standard documents.

Tech Scope:

You may use:

- **Python** for backend (PyMuPDF, pdfplumber, langchain, FAISS, or embeddings).
- **Frontend:** Streamlit / Flask / React for interaction.
- **AI Layer:** OpenAI / Hugging Face / Local Embeddings for semantic search.

Milestones (6-Hour Hackathon Plan):

Task 1: Setup and PDF Processing

- Extract text from multi-page PDFs.
- Clean and store text efficiently (e.g., using vector embeddings or chunking).

Task 2: Question Answering Logic

- Implement question input + search mechanism.
- Return matching content **exactly as found** in the PDF.
- Include basic text highlighting or snippet source reference.

Task 3: Frontend + Visualization Layer

- Create a simple UI for file upload + question input.
- Display retrieved text snippet + matched image/diagram side-by-side.
- Optional: Add voice input or response playback.

Bonus Enhancements:

- OCR for text embedded inside diagrams.
- Page number and bounding box highlighting.
- Voice-based questioning and spoken answers.
- Automatic labeling of extracted figures and captions.

⚡ Impact:

This project transforms static PDFs into interactive, intelligent learning assistants, bridging the gap between search engines and document understanding AI.

Problem Statement 3:

Parkinson's Disease (PD) is a progressive neurological disorder affecting movement, speech, and motor functions. Early diagnosis is critical, as it allows for timely medical intervention to slow disease progression and improve quality of life.

However, early symptoms are subtle and often missed in traditional clinical diagnosis. This project aims to develop a machine learning-based model to predict Parkinson’s Disease from voice measurements and other biomedical features. To ensure **trust and transparency**, the model will incorporate **Explainable AI (XAI)** techniques such as SHAP or LIME to provide insights into why a prediction was made.

- **Why This Matters?**

Area	**Impact**
Healthcare	Enables early intervention and symptom management
Accessibility	Can work with voice data, making remote and rural diagnosis feasible
Trust	Explainable AI makes model decisions transparent
Cost-effective	Potential to develop into a non-invasive, scalable screening tool

Parkinson’s Disease – Symptoms

Symptom	**Description**
Tremors	Involuntary shaking, especially in hands or fingers
Bradykinesia	Slowness of movement
Muscle Rigidity	Stiffness in limbs or joints

Symptom	Description
Postural Instability	Loss of balance and coordination
Voice Changes	Soft, hoarse, or monotonous voice
Facial Masking	Reduced facial expressions
Micrographia	Small or cramped handwriting

Voice changes, which are early symptoms, can be quantitatively analyzed using signal features like:

- **Fundamental frequency (Fo)**
- **Jitter, Shimmer**
- **Harmonics-to-Noise Ratio (HNR)**

These are captured in datasets such as the **UCI Parkinson's Dataset**.

Project Breakdown – 3 Core Tasks

Task 1: Data Exploration and Preprocessing

- Load dataset (e.g., UCI Parkinson's Dataset)
- Analyze feature distribution, remove irrelevant features
- Normalize/scale data
- Visualize correlations and class imbalance

Task 2: Model Development

- Use classifiers like Random Forest, XGBoost
- Perform train-test split and cross-validation
- Evaluate models using:
    - Accuracy

    - Confusion Matrix
    - ROC Curve
    - F1 Score

Task 3: Explainable AI Integration

- Apply **SHAP** or **LIME** to interpret predictions
- Generate:
    - Global feature importance plots
    - Local (individual prediction) explanations
- Optional: build a **Streamlit app** for demo

Realistic Outcomes:

Time Frame	**Task**	**Outcome**
Hour 1	Load & explore data	Dataset loaded, EDA visuals, understand key features
Hour 2	Preprocessing	Feature scaling, train-test split, remove unnecessary columns
Hour 3	Train model	Train & test 1–2 ML models, get evaluation metrics
Hour 4	Explainable AI (SHAP or LIME)	Visualize important features and explain sample predictions
Hour 5	Generate visuals + insights	Save SHAP plots, confusion matrix, ROC curve
Hour 6	Final presentation	Create slide deck with problem, model, explainability, results

Final Deliverables

1. Trained Prediction Model

- XGBoost or Random Forest classifier
- Achieves ~85–90% accuracy

2. Explainable AI Integration

- SHAP summary plot: shows top contributing features
- SHAP force/waterfall plot: explains individual predictions

3. Visual Outputs

- SHAP plots
- ROC curve
- Confusion matrix

5. Optional: Streamlit App

- User input sliders for voice features
- Displays prediction and SHAP explanation