Project Plan: FloatChat - AI-Powered ARGO Data Explorer

A step-by-step guide to building a functional MVP for the Smart India Hackathon.

Phase 1: The Foundation (Offline Data Preparation)

- **Goal:** To create a single, efficient data source for the application to use. This is a one-time setup process.
- **Action:** Consolidate data from all float folders into a single `master_dataset.parquet` file.
- **Process:**
    1. Write a Python script using the pandas library.
    2. The script will loop through each float's data directory, loading the `measurements.parquet` and `trajectory.parquet` files.
    3. It will merge these files to ensure every measurement has associated latitude, longitude, and date information.
    4. A `float_id` column will be added to each row to track the source.
    5. All processed data will be combined and saved into a single `master_dataset.parquet` file.
- **Outcome:** A unified, query-ready dataset that allows the application to perform fast searches across all floats simultaneously.

Phase 2: Building the MVP (The Live Application)

This phase focuses on creating the four core components of the working prototype.

1. **The Frontend (User Interface)**
    - **What:** A simple, clean web interface with a chat window.
    - **Tool:** Streamlit.
    - **Result:** Users have an intuitive way to interact with the system by typing questions in natural language.
2. **The Interpreter (AI Brain)**
    - **What:** A module that understands the user's question by extracting key details.
    - **Tool:** An open-source LLM accessed via an API (e.g., Mistral, Llama 3, or QWEN).
    - **Process:** The app will send the user's raw text question to the LLM API. The prompt will instruct the model to perform Entity Extraction and return a structured JSON.

- **Result:** An unstructured query like "show me salinity in the Arabian Sea" is converted into a machine-readable command: `{"parameter": "salinity", "location": "Arabian Sea", "date": null}`.

3. **The Query Engine (Data Logic)**
    - **What:** The backend logic that finds the data requested by the user.
    - **Tools:** Python, pandas, and a geocoding library (like `geopy`).
    - **Process:** This component takes the structured JSON from the Interpreter. It uses a pre-defined dictionary of bounding boxes for common locations but will use the `geopy` library to dynamically find coordinates for any location not in the dictionary. It then filters the `master_dataset.parquet` based on these coordinates and any other provided entities.
    - **Result:** A filtered pandas DataFrame containing only the data rows that precisely match the user's request.
4. **The Visualizer (Data Output)**
    - **What:** A module to display the results visually.
    - **Tool:** Plotly.
    - **Process:** The filtered data from the Query Engine is used to generate an interactive map (`scatter_mapbox`). Each data point is plotted, and its color will represent the value of the requested parameter.
    - **Result:** A clear, insightful map visualization is displayed directly in the chat window.

Phase 3: The Final Application Flow

1. **User Asks:** The user types a question into the Streamlit chat window.
2. **App Interprets:** The question is sent to the LLM API, which extracts entities and returns JSON.
3. **App Queries:** The Query Engine uses the JSON to filter the master dataset, using geocoding for location flexibility.
4. **App Visualizes:** The filtered data is used to generate a dynamic Plotly map.
5. **User Sees:** The application displays a text summary and the interactive map in the chat window.

Phase 4: The Vision (The Path to a Full RAG System)

- **Presentation Goal:** Explain that this MVP is the foundation for a more advanced system.
- **Future Plan:** The Query Engine's simple logic will be replaced by a FAISS vector database. The Interpreter's role will evolve to generate complex database queries from scratch. This

upgrade will transform the MVP into a full-scale Retrieval-Augmented Generation (RAG) system as required by the problem statement's full scope.