## Project Roadmap: AI-Based Cyber Security Threats Prediction

* **Project Title:** AI-Based Cyber Security Threats Prediction
* **Intern:** Killampalli Yaswanth Vardhan
* **Mentor:** Vivek
* **Duration:** 12 Weeks

### 1. Executive Summary

This project aims to develop an intelligent system that acts as an autonomous agent for network security. By leveraging a machine learning model trained on the CSE-CIC-IDS2018 dataset, the system will monitor network traffic, predict potential cyber threats in real-time, and present the findings on an intuitive web-based dashboard. The complete system will feature a React frontend, a Node.js backend for orchestration, a Flask server for model inference, and a Supabase database for data persistence. The ultimate goal is to create a proof-of-concept that enhances an organization's security posture by automating threat detection, allowing human experts to focus on strategic security challenges.

### 2. Project Goals & Objectives

* **Primary Goal:** To build, test, and document a functional prototype of an AI-powered cybersecurity threat prediction system.
* **Key Objectives:**
  1. Develop a high-accuracy threat prediction model using XGBoost.
  2. Build a scalable three-tier application architecture (Frontend, Backend, Model-Server).
  3. Create a real-time dashboard to visualize network status and predicted threats.
  4. Implement a mechanism for the "agent" to log threats and trigger alerts.
  5. Ensure seamless integration between all components of the system.

### 3. Technology Stack & Architecture

#### Core Technologies

* **Machine Learning Model:** XGBoost
* **Dataset:** CSE-CIC-IDS2018 on AWS
* **Frontend:** React (with Recharts for visualizations)
* **Backend Server:** Node.js (with Express.js)
* **Model Server:** Flask (Python)
* **Database:** Supabase (PostgreSQL)

#### System Architecture

The system follows a microservices-oriented approach where each component has a distinct responsibility.

1. **React Frontend:** The user-facing dashboard that visualizes data and alerts from the backend.
2. **Node.js Backend:** The central hub. It serves the frontend, communicates with Supabase to store and retrieve historical data/logs, and makes API calls to the Flask server to get predictions.
3. **Flask Model-Server:** A lightweight Python server whose sole purpose is to load the trained XGBoost model and expose an API endpoint (e.g., /predict) that accepts network packet data and returns a threat prediction.

### 4. Answering Your Question: What Else is Needed?

Your current stack is solid for a proof-of-concept. To elevate it to a more professional and "agentic" level, consider these additions:

* **Real-time Traffic Capture:** The project statement mentions real-time monitoring. The CSE-CIC-IDS2018 dataset is static and used for *training*. For the live application, you'll need a way to capture real network packets.
  + **Suggestion:** Use a Python library like **Scapy** or **Pyshark** within a separate script that captures traffic and sends it to your backend for prediction.
* **Containerization:** To manage your three separate servers (React, Node, Flask) easily, especially for deployment.
  + **Suggestion:** Use **Docker** and **Docker Compose**. This allows you to define and run your entire multi-container application with a single command. It's a critical skill in modern development.
* **Response Mechanism (The "Agentic" Part):** How does the agent *respond*? Prediction is just the first step.
  + **Suggestion:** Implement an alerting module in your Node.js backend. When a threat is detected, it could use a service like **Twilio SendGrid** to send an email alert or a webhook to post a message in a **Slack** channel.
* **Environment Variables:** To securely manage credentials like your Supabase API keys.
  + **Suggestion:** Use .env files to store sensitive keys and access them in your Node.js and Flask applications, ensuring they are not hard-coded in the source code.

### 5. Detailed Project Roadmap & Timeline

#### Phase 1: Foundation & Model Development (Weeks 1-3)

* **Week 1: Setup & Data Exploration**
  + Set up development environments for Python, Node.js, and React.
  + Download and explore the CSE-CIC-IDS2018 dataset.
  + Perform Exploratory Data Analysis (EDA) to understand features, distributions, and class imbalances.
* **Week 2: Data Preprocessing & Feature Engineering**
  + Clean the dataset (handle missing values, duplicates).
  + Perform feature scaling and selection.
  + Prepare the final, cleaned dataset for model training.
* **Week 3: Model Training & Evaluation**
  + Train the XGBoost classifier on the preprocessed data.
  + Evaluate model performance using metrics like Accuracy, Precision, Recall, and F1-Score.
  + Fine-tune hyperparameters to optimize performance.
  + Save the final trained model (.joblib or .pkl) and any necessary processors (e.g., LabelEncoder).

#### Phase 2: Backend & API Development (Weeks 4-6)

* **Week 4: Model-Server (Flask)**
  + Build the Flask API server.
  + Create a /predict endpoint that loads the saved XGBoost model.
  + Define the JSON structure for input (features) and output (prediction).
* **Week 5: Backend Server (Node.js)**
  + Set up the Node.js project with Express.js.
  + Establish a connection to your Supabase project.
  + Design the database schema for storing threat logs.
* **Week 6: API Integration & Logic**
  + Create backend API routes (e.g., /api/predict, /api/logs).
  + The /api/predict route will call the Flask server, receive the prediction, and then save the result to Supabase.
  + Implement the alerting mechanism (e.g., email or Slack notifications).

#### Phase 3: Frontend Development & Integration (Weeks 7-9)

* **Week 7: UI/UX & Component Scaffolding**
  + Set up the React project using Vite.
  + Design a simple and effective dashboard layout.
  + Create reusable components for charts, tables, and alerts.
* **Week 8: Dashboard & Visualization**
  + Integrate **Recharts** to display data.
  + Create a live-updating chart for network traffic status and a table to display the threat log from Supabase.
* **Week 9: Full-Stack Integration**
  + Connect the React frontend to the Node.js backend APIs.
  + Ensure data flows correctly from the frontend to the backend and is displayed accurately.
  + Test the entire prediction-to-visualization pipeline.

#### Phase 4: Finalization & Deployment (Weeks 10-12)

* **Week 10: Real-time Component & Testing**
  + Develop the Python script using **Scapy** to capture live network data.
  + Integrate this script with the backend.
  + Perform thorough end-to-end testing of the entire system.
* **Week 11: Containerization & Documentation**
  + Write Dockerfiles for the React, Node.js, and Flask applications.
  + Create a docker-compose.yml file to orchestrate all services.
  + Begin writing the final project report and documenting the codebase.
* **Week 12: Final Presentation & Handover**
  + Prepare a presentation summarizing the project's goals, architecture, and outcomes.
  + Demonstrate the live working application.
  + Finalize all documentation and clean up the code repository for submission.

### 6. Key Deliverables

1. **Source Code:** A well-documented code repository (e.g., on GitHub) with separate folders for the frontend, backend, and model server.
2. **Trained Model:** The final, serialized XGBoost model file.
3. **Technical Documentation:** A report detailing the project architecture, setup instructions, API endpoints, and design choices.
4. **Final Presentation:** A slide deck and live demonstration of the completed project.