**ML Model Performance Dashboard**

**Project Overview**

This is a React-based web application that provides a visual interface for data scientists and ML engineers to track their model performance metrics, training progress, and predictions. While it currently uses mock data, it's structured to be easily connected to real backend APIs that would fetch live ML model data.

**Technology Stack**

* **Framework**: React with TypeScript for type safety
* **Styling**: Tailwind CSS for responsive design
* **Charts**: Chart.js with React-Chartjs-2 wrapper for visualizations
* **Icons**: Heroicons for the UI elements
* **Build Tool**: Vite with Bun as the package manager

**Project Structure**

The project follows a component-based architecture:

* **Layouts**: Contains the DashboardLayout which provides the overall structure with sidebar navigation
* **Components**:
  + **Dashboard**: Components specific to the dashboard like SummaryCardand PredictionsTable
  + **Charts**: Various chart components like LineChart, BarChart, HeatMap, and HorizontalBarChart
  + **UI**: General UI components like ThemeToggle for switching between light and dark modes
* **Data**: Mock data in mockData.ts that simulates ML metrics and performance data

**Main Features**

1. **Dashboard Overview**:
   * Key performance metrics displayed in summary cards
   * Model performance trends over time
   * Comparison between different model versions
2. **Visualizations**:
   * Line charts for temporal performance data
   * Bar charts for comparing models across metrics
   * Confusion matrix heatmap for classification results
   * Feature importance visualization
3. **Recent Predictions**:
   * Table showing recent model predictions
   * Status indicators (correct/incorrect)
   * Confidence level visualization
4. **Resource Usage**:
   * Monitoring GPU, CPU, and memory usage
   * Training cost tracking
5. **Dark Mode**:
   * System preference detection
   * Manual toggle option
   * Smooth transitions between themes

**User Experience**

The dashboard is designed to be:

* **Responsive**: Works across mobile, tablet, and desktop devices
* **Intuitive**: Clear organization of information with visual hierarchy
* **Interactive**: Visual feedback on hover and click events
* **Accessible**: Proper contrast ratios and semantic HTML

**Application Flow**

1. The app initializes by checking for user theme preferences
2. Mock data is loaded to populate the dashboard components
3. Users can navigate between different sections using the sidebar
4. The theme toggle in the header allows switching between light and dark modes

**Possible Extensions**

This project is set up to be extended in several ways:

1. Adding real data fetching from backend APIs
2. Implementing filters and date range selectors
3. Adding drill-down capabilities for detailed analysis
4. Creating additional dashboard views (training, datasets, infrastructure)
5. Implementing authentication and authorization

The codebase is well-structured and follows TypeScript best practices, making it maintainable and extensible for future development.

**Language and Technology Stack of the AI/ML Dashboard**

**Programming Languages**

1. **TypeScript (v5.6.3)**
   * Strongly-typed superset of JavaScript
   * Provides type safety, reducing runtime errors
   * Used throughout the project for all component and data definitions
   * Enables better IDE support with autocompletion and type checking
   * Example usage: Type definitions for chart data, component props, and prediction objects
2. **JSX/TSX**
   * XML-like syntax extension used with React
   * Allows mixing HTML-like markup with JavaScript/TypeScript
   * All UI components are written using TSX
   * Used for declaring component structure and composition

**Frontend Framework**

1. **React (v18.3.1)**
   * Component-based UI library
   * Uses a virtual DOM for efficient rendering
   * Provides hooks like useState and useEffect for state management and side effects
   * Used for creating reusable UI components like charts, cards, and tables
   * Example usage: The ThemeToggle component uses React hooks to manage light/dark mode state

**Styling and UI**

1. **Tailwind CSS (v3.4.17)**
   * Utility-first CSS framework
   * Enables rapid UI development with predefined utility classes
   * Provides responsive design utilities
   * Used for all styling in the project
   * Example: className="bg-white dark:bg-gray-800 rounded-lg shadow p-4"applies multiple styles at once
2. **Heroicons (v24)**
   * SVG icon set designed for use with Tailwind CSS
   * Provides consistent styling for UI elements
   * Used for navigation, status indicators, and action buttons
   * Example: <HomeIcon className="h-6 w-6" /> renders a home icon

**Data Visualization**

1. **Chart.js (v4.4.9)**
   * JavaScript charting library
   * Provides highly customizable, responsive charts
   * Supports multiple chart types including line, bar, and pie
   * Used for all data visualizations in the dashboard
2. **React-Chartjs-2 (v5.3.0)**
   * React wrapper for Chart.js
   * Simplifies using Chart.js in React applications
   * Provides React components for different chart types
   * Example usage: <Line options={options} data={data} /> creates a line chart

**Build Tools and Development Environment**

1. **Vite (v6.2.0)**
   * Modern frontend build tool
   * Provides fast development server with hot module replacement
   * Efficiently bundles the application for production
   * Used as the main development and build tool
2. **Bun (v1.2.8)**
   * JavaScript runtime and package manager
   * Alternative to Node.js and npm/yarn
   * Provides faster package installation and script execution
   * Used for managing dependencies and running scripts
3. **Biome (v1.9.4)**
   * JavaScript/TypeScript linter and formatter
   * Helps maintain code quality and consistency
   * Used for static code analysis and automated formatting

**Project Structure Technologies**

1. **ESLint/TypeScript-ESLint (v8.29.0)**
   * Static code analysis tools
   * Enforces coding standards and best practices
   * Helps catch common errors
   * Configured with rules specific to React and TypeScript
2. **Prettier (v3.5.3)**
   * Code formatter
   * Ensures consistent code style
   * Integrates with Biome for formatting TypeScript and React code

**Deployment Technologies**

1. **Netlify Configuration**
   * Included for easy deployment to Netlify hosting
   * Configured in netlify.toml file
   * Supports both static and serverless deployments

**Browser Technologies Utilized**

1. **LocalStorage API**
   * Used for persisting user theme preferences
   * Example: localStorage.getItem('darkMode')
2. **Media Queries API**
   * Used to detect system color scheme preference
   * Example: window.matchMedia('(prefers-color-scheme: dark)')
3. **CSS Custom Properties (Variables)**
   * Used for theme-based styling
   * Facilitates dark/light mode switching
4. **CSS Transitions**
   * Used for smooth theme transitions
   * Example: transition-colors duration-200