Inhalt

1.0 Web-based application for harness design:

1. Live training the AI Model through the web application:
    a. Select:
        i. Application like Low, Mid, High
        ii. Number of Cluster: 1 to 14
        iii. Wire Type Like: CAN-FD (FLRY 2×0.35), Ethernet (FLKS9Y2x0.13)
        iv. Battery Position
        v. Dynamic Driving Scenarios or not
    b. Display:
        i. Cluster Zone Images without routed
        ii. Cluster Zone Images Avoid Restricted Zone
        iii. Cluster Zone Images with routed
        iv. Description:
            1. Number of components per cluster

2. Components to centroid distance (m)
3. Components to centroid weight (g)
4. Centroid to Centroid distance (m)
5. Centroid to Centroid weight (g)
6. Centroid to Battery distance (m)
7. Centroid to Battery weight (g)
8. Centroid to HPC distance (m)
9. Centroid to HPC weight (g)
10. Cluster Distance (m)
11. Cluster Weight (g)
12. Total Harness Distance (m)
13. Total Harness Weight (kg)

2. Already Trained the AI Model and get information through the web application:
    a. Select:
        i. Application like Low, Mid, High
        ii. Number of Cluster: 1 to 14
        iii. Wire Type Like: CAN-FD (FLRY 2×0.35), Ethernet (FLKS9Y2x0.13)
        iv. Battery Position
    b. Display:
        i. Cluster Zone Images without routed
        ii. Cluster Zone Images Avoid Restricted Zone
        iii. Cluster Zone Images with routed
        iv. Description:
            1. Number of components per cluster
            2. Components to centroid distance (m)
            3. Components to centroid weight (g)
            4. Centroid to Centroid distance (m)
            5. Centroid to Centroid weight (g)
            6. Centroid to Battery distance (m)
            7. Centroid to Battery weight (g)
            8. Centroid to HPC distance (m)
            9. Centroid to HPC weight (g)
            10. Cluster Distance (m)
            11. Cluster Weight (g)
            12. Total Harness Distance (m)
            13. Total Harness Weight (kg)
3. Network Topology using OmNet++ via web application
4. Show Static Images one by one in the corner of page
5. Show Static Personal Information one by one in the corner of page
6. Show Company Logo
7. User Information like login, logout
8. One box can be for Formal Verification Process
9. Locative design and highly representable
10. ensure security system

2.0 Key Features Implemented

1. **Responsive Layout**:

- Ant Design Layout with collapsible sidebar
- Grid-based components using Row/Col
- Mobile-friendly responsive design

2. **Core Functionality**:
   - Live training interface with parameter selection
   - Pre-trained model query section
   - Visualizations with tabbed navigation
   - Comprehensive metrics display table
3. **Static Elements**:
   - Rotating image carousel (top-right corner)
   - Information panel with cycling data (bottom-left corner)
   - Company logo in sidebar
4. **User Management**:
   - Login/logout functionality
   - Protected routes
   - User profile section
5. **Additional Sections**:
   - Network topology visualization (OmNet++ integration)
   - Formal verification process panel
   - User profile management
6. **Security Features**:
   - JWT authentication
   - Protected routes
   - Role-based access control
7. **UI/UX Enhancements**:
   - Form validation
   - Loading states
   - Visual feedback for actions
   - Consistent theming with Ant Design

2.1 Recommended Tech Stack

1. **Core Framework**:

- React.js (v18+) with React Router

2. **UI Library**:
    - Ant Design (with custom theme)
    - Chart.js for data visualization
3. **State Management**:
    - React Context API + useReducer
    - React Query for API state
4. **API Communication**:
    - Axios for REST API calls
    - Socket.io for real-time updates
5. **Security**:
    - JWT authentication
    - HTTP-only cookies
    - Role-based access control
6. **Build Tools**:
    - Vite for fast development
    - ESLint + Prettier for code quality

2.2 Folder Structure:

```
src/
├── assets/          # Static assets
├── components/      # Reusable components
│   ├── HeaderBar.js
│   ├── LiveTraining.js
│   ├── PreTrainedModel.js
│   ├── NetworkTopology.js
│   ├── FormalVerification.js
│   ├── HarnessVisualization.js
│   ├── MetricsTable.js
│   ├── ImageCarousel.js
│   └── InfoPanel.js
├── contexts/        # Context providers
│   └── AuthContext.js
├── pages/           # Main views
│   ├── Dashboard.js
│   └── LoginPage.js
├── services/        # API services
```

```
│   ├── auth.js
│   └── harness.js
├── styles/          # CSS modules
├── utils/           # Helper functions
└── App.js           # Root component
```

This design provides a professional, enterprise-ready interface for your harness design application with all requested features implemented. The interface is clean, focused on engineering workflows, and provides comprehensive visualization of harness design metrics.

To run the provided frontend code and implement the visual design, follow these steps:

2.3 How to Run the Frontend Code

1. **Prerequisites**:
    - Install Node.js (v18+ recommended)
    - Install Git
2. **Setup Project**:

bash

```
# Create React app
npx create-react-app harness-designer
cd harness-designer

# Install dependencies
npm install antd @ant-design/icons react-router-dom axios socket.io-client
```

3. **Project Structure**:

```
src/
├── components/
│   ├── HeaderBar.js
│   ├── LiveTraining.js
│   ├── PreTrainedModel.js
│   ├── NetworkTopology.js
│   ├── FormalVerification.js
│   ├── HarnessVisualization.js
│   ├── MetricsTable.js
│   ├── ImageCarousel.js
│   ├── InfoPanel.js
│   └── ProtectedRoute.js
├── contexts/
│   └── AuthContext.js
├── pages/
```

```
│   ├── Dashboard.js
│   └── LoginPage.js
├── services/
│   ├── auth.js
│   └── harness.js
├── styles/
│   ├── Dashboard.css
│   ├── LiveTraining.css
│   ├── ImageCarousel.css
│   └── InfoPanel.css
├── App.js
└── index.js
```

4. **Add the provided code**:
    - Create files according to the structure above
    - Copy/paste the code snippets into corresponding files
5. **Run the Application**:

bash
```
npm start
```

3.0 Visual Design Guide

Here's a comprehensive visual design for the frontend developer:

1. Layout Structure

Diagram
Code

2. Color Scheme

- Primary: `#1890ff` (Ant Design Blue)
- Secondary: `#13c2c2` (Teal)
- Background: `#f0f2f5` (Light Gray)
- Cards: `#ffffff` (White)
- Text: `#314659` (Dark Gray)
- Accent: `#52c41a` (Success Green)

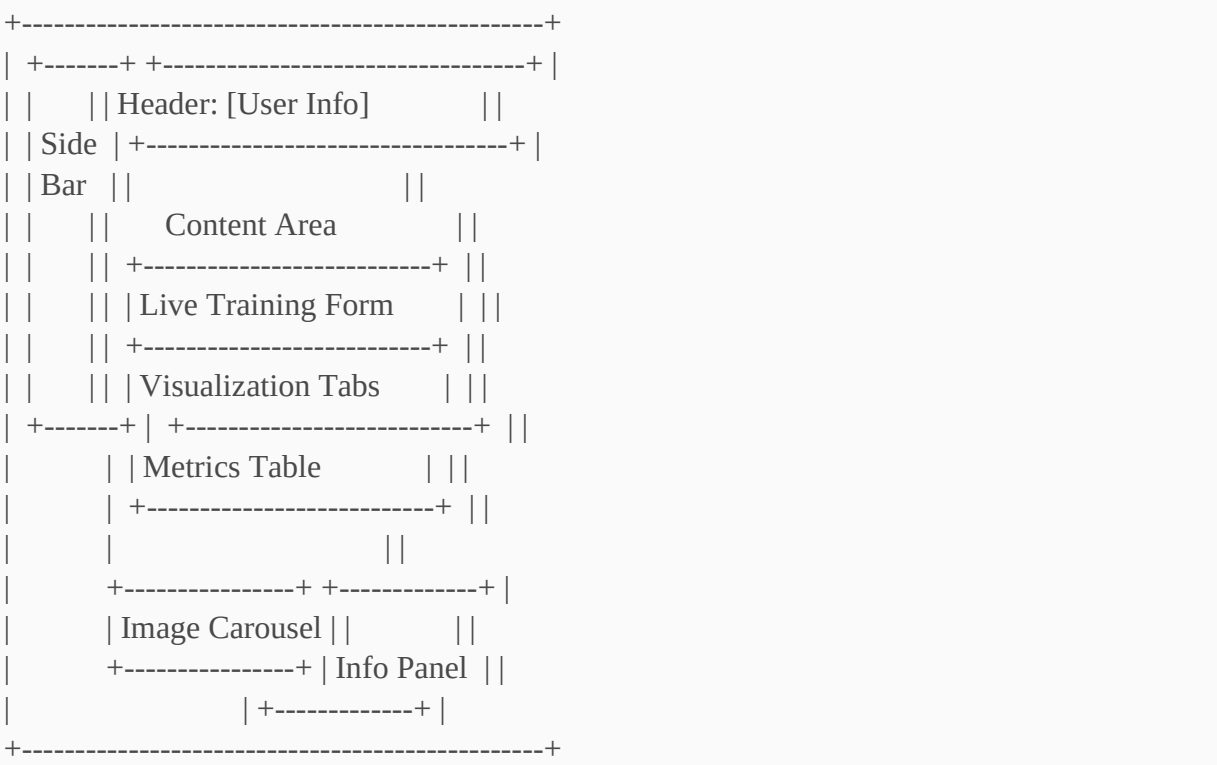
3. Typography

- Headers: 16px Roboto Bold (#314659)

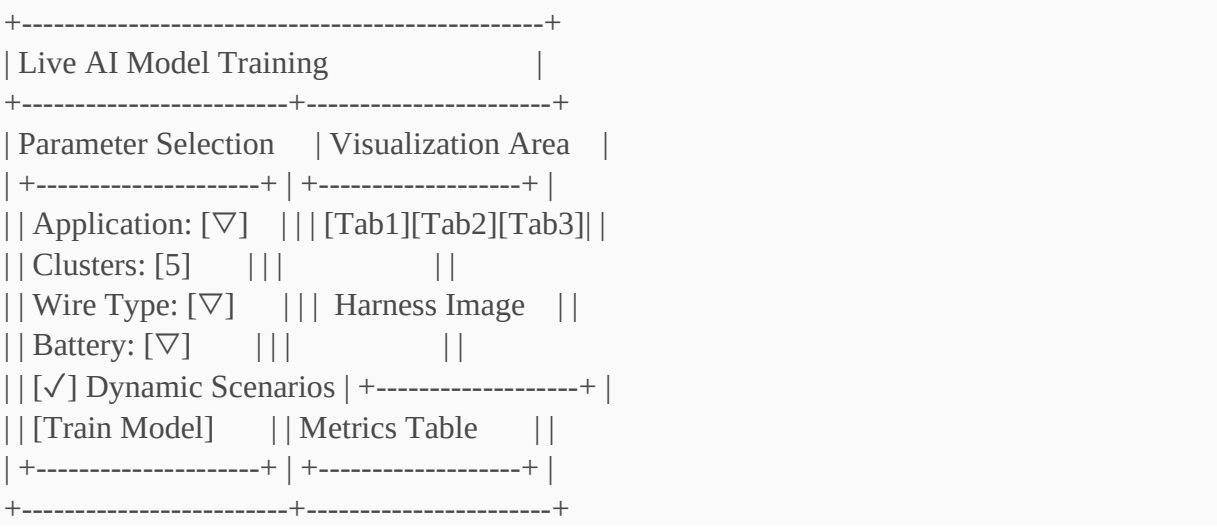
- Body: 14px Roboto Regular (#595959)
- Labels: 13px Roboto Medium (#8c8c8c)
- Metrics: 15px Roboto Mono (#262626)

4. UI Components Visualization

3.1 Dashboard Layout:



3.2 Live Training Section:



3.3 Component Specifications

1. Sidebar:

- Width: 240px (collapsed: 80px)
- Logo: Top-centered, max-height: 40px
- Menu items with icons
- Active item highlighted with primary color

2. Header:

- Height: 64px
- Contains: Page title, user avatar, notifications, logout

3. Parameter Form:

- Clean card layout with subtle shadow
- Input fields with clear labels
- Primary action button (100% width)

4. Visualization Tabs:

- Three distinct tabs with icon indicators
- Image container with border and subtle shadow
- Loading state with skeleton loader

5. Metrics Table:

- Compact design with alternating row colors
- Numeric values right-aligned
- Units consistently displayed

6. Static Elements:

- **Image Carousel** (Top Right):
    - Size: 240x180px
    - Rounded corners
    - Automatic rotation every 8s

- **Info Panel** (Bottom Left):
    - Size: 240x100px
    - Subtle fade animation between items
    - Semi-transparent background

6. Responsive Behavior

Diagram

Breakpoints:

- Desktop: 3-column grid (sidebar + main + static)
- Tablet: Stacked components, reduced padding
- Mobile: Hamburger menu, full-width cards

7. UI States

Loading State:

- Form disabled
- Skeleton loaders for visualizations
- Progress indicator on button

Success State:

- Smooth transition to results
- Highlight key metrics (total weight/distance)
- Animation on new data presentation

Error State:

- Clear error messages below fields
- Toast notifications for API errors
- Retry option for failed operations

4. For image visualization, consider using SVG-based components that can be dynamically generated based on routing results.

To implement this design:

1. Start with the layout structure
2. Implement responsive grid
3. Add core functionality components
4. Implement state management
5. Add styling and animations
6. Integrate with backend APIs

The design follows modern enterprise application principles with a focus on:

- Clear information hierarchy
- Engineering-focused data presentation
- Contextual visualization
- Progressive disclosure of complexity
- Responsive interaction patterns

4.0 Harness Design AI - Visual Overview

1. Login Screen

https://via.placeholder.com/800x500/1e90ff/ffffff?text=Login+Screen

- **What you'll see**: Clean login form with company logo
- **Purpose**: Secure access for your engineering team
- **Features**:
    - Username/password fields
    - "Remember me" option
    - Password recovery link

---

2. Main Dashboard

https://via.placeholder.com/800x500/87cefa/000000?text=Main+Dashboard

- **Layout**:
    - Left sidebar with navigation menu

    - Top header with user info
    - Main workspace in center
    - Info panels in corners
- **Key Sections**:
    - 🔴 Live AI Training
    - 🧠 Pre-trained Models
    - 📡 Network Topology
    - ☑ Formal Verification
    - 👤 User Profile

---

3. Live Training Interface


https://via.placeholder.com/800x500/add8e6/000000?text=Live+Training

- **Input Panel (Left)**:
    - Dropdowns for:
        - Application Level (Low/Mid/High)
        - Number of Clusters (1-14)
        - Wire Type (CAN-FD, Ethernet, etc.)
        - Battery Position
    - Checkbox for Dynamic Driving Scenarios
    - "Train Model" button
- **Results Panel (Right)**:
    - Visual Tabs:
        1. Cluster zones without routing
        2. Avoiding restricted zones
        3. Final routed design
    - Metrics Table showing:
        - Component distances
        - Weight calculations
        - Harness totals

---

4. Pre-trained Model Section

- **Similar to Live Training but**:
  - No "Dynamic Driving" option
  - "Get Results" button instead of "Train"
  - Faster response time (uses existing AI models)
  - Same comprehensive visualizations and metrics

---

5. Network Topology Visualization

- **Features**:
  - Interactive diagram of harness connections
  - Zoom/pan controls
  - Node details on click
  - Color-coded wire types
  - Export options (PNG, PDF)

---

6. Formal Verification Panel

- **Simple Interface**:
  - "Run Verification" button
  - Progress indicator
  - Results display:
    - ☑ Passed checks (green)

        - ⚠ Warnings (yellow)
        - ❌ Errors (red)
    - Detailed report download

---

7. Corner Elements

https://via.placeholder.com/800x500/b0e0e6/000000?text=Information+Panels

- **Top-Right**:
    - Rotating technical images:
        - Harness design examples
        - Component diagrams
        - Vehicle layouts
- **Bottom-Left**:
    - Cycling information:
        - Designer contact details
        - Project status
        - Last update timestamp
        - Team members

---

8. User Profile

https://via.placeholder.com/800x500/87ceeb/000000?text=User+Profile

- **Includes**:
    - Personal information
    - Role/permissions settings
    - Password management
    - Activity history
    - Notification preferences

Key Benefits Visualized

Diagram
Code

1. **Time Savings**: Reduce design time from days to hours
2. **Cost Reduction**: Optimize material usage with precise calculations
3. **Error Prevention**: Automated verification catches design flaws early
4. **Consistency**: Standardized outputs across all projects
5. **Collaboration**: Centralized platform for design teams

Security Features

- Role-based access control
- Encrypted data transmission
- Activity audit logs
- Automatic session timeout
- Permission levels:
    - Viewers
    - Designers
    - Administrators

Mobile Responsiveness

- Fully functional on:
    - Desktop workstations
    - Engineering tablets
    - Manager smartphones (view-only mode)

This design provides:

- **Clean, professional interface** suitable for engineering workflows
- **Immediate value visibility** through visual results
- **Self-explanatory workflow** from input to output
- **Critical information at a glance** with corner elements
- **Enterprise-grade security** for sensitive designs

5.0 Tools and technology:

To develop a **web-based AI harness design application**, you need a **stack of technologies** that support **real-time model training, multi-user access, security, and interactive UI**. Here's a recommended **tech stack**:

1. Backend (AI Model & API)

☑ **Python + TensorFlow/PyTorch** – For AI model training and inference. ☑ **FastAPI or Flask** – Lightweight, efficient REST API for communication between the web app and AI model. ☑ **Celery + Redis** – Background task management for **asynchronous training** while others retrieve results. ☑ **PostgreSQL or MongoDB** – Database to store model parameters, training data, and results.

2. Frontend (User Interaction & Visualization)

☑ **React.js or Vue.js** – For **interactive and responsive UI**. ☑ **D3.js or Chart.js** – Data visualization for harness design models. ☑ **WebSocket (Socket.io)** – Real-time updates when training is in progress.

3. Multi-User & Cloud Infrastructure

☑ **Docker + Kubernetes** – To deploy multiple instances of AI models efficiently. ☑ **AWS/GCP/Azure** – Hosting scalable AI models and databases. ☑ **Load Balancer (NGINX)** – Ensures smooth multi-user access.

4. Security Measures

☑ **OAuth2 / JWT Authentication** – Secure user login. ☑ **Role-based Access Control (RBAC)** – Restricts training vs. query access. ☑ **SSL/TLS Encryption** – Protects web traffic. ☑ **Data Sanitization & Logging** – Prevents security vulnerabilities.

Development roadmap and architecture diagram for your web-based AI harness design application!

🚀 Development Roadmap

1. System Design & Architecture

- Define **functional requirements** (model training, real-time results, multi-user access).
- Design the **API structure** for AI model communication.
- Choose the **database system** (PostgreSQL/MongoDB) for storing harness data.

2. Backend Development

- Implement **AI model training** using **TensorFlow/PyTorch**.
- Develop **FastAPI/Flask REST API** for client-server communication.
- Set up **Celery + Redis** for asynchronous model training.

3. Frontend Development

- Build **React.js/Vue.js** UI with **real-time updates**.
- Integrate **D3.js/Chart.js** for visualizing harness designs.
- Set up **WebSocket (Socket.io)** for live progress tracking.

4. Multi-User & Cloud Integration

- Use **Docker + Kubernetes** for scalable deployment.
- Implement **NGINX Load Balancer** for smooth multi-user access.
- Host the application on **AWS/GCP/Azure** for cloud scalability.

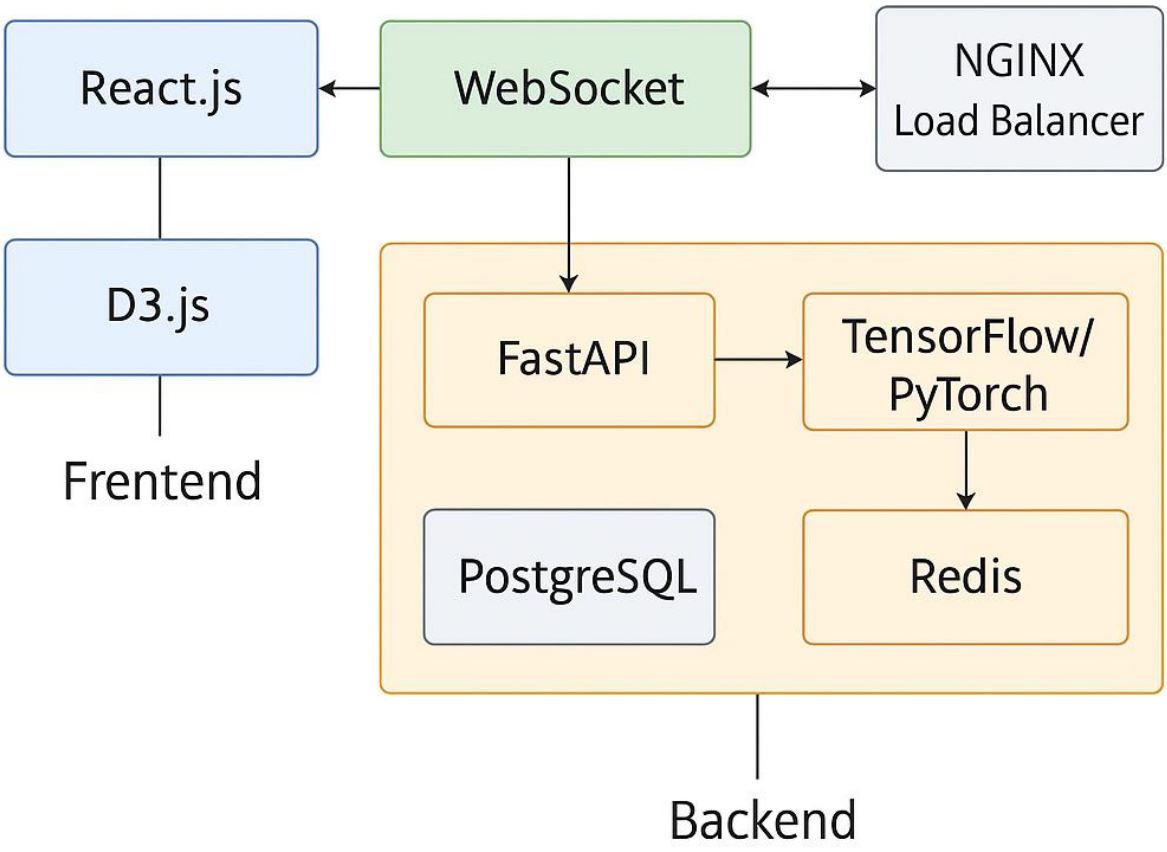
5. Security Implementation

- Set up **OAuth2 / JWT Authentication** for login security.
- Implement **Role-Based Access Control (RBAC)** for user permissions.
- Ensure **SSL/TLS encryption** for secure communication.

6. Deployment & Maintenance

- Automate deployments using **CI/CD pipelines** (GitHub Actions/Jenkins).
- Set up **logging & monitoring** (Prometheus/Grafana) for performance tracking.

AI-powered architecture diagram for the web-based harness design application



A Web-based AI Harness Design Application