NeuroForge AI

Advanced Training Orchestration Platform

A Modern Implementation Guide for

YOLO Model Training with Genetic Algorithms

December 4, 2025

React + TypeScript + Docker + MLOps

Author

William Rodriguez

Software Engineer & System Architect

eCaptureDtech

Badajoz, Extremadura, España

Contact Information:

LinkedIn: es.linkedin.com/in/wisrovi-rodriguez

Email: william.rodriguez@ecapturedtech.com

GitHub: github.com/wisrovi

Expert in MLOps and AI Infrastructure

Contents

List of Figures

List of Tables

Listings

Chapter 1

Executive Summary

NeuroForge AI represents a paradigm shift in machine learning training orchestration, providing a sophisticated web-based platform designed to manage complex YOLO (You Only Look Once) model training workflows across distributed GPU clusters. Built with modern React 19 and TypeScript, the platform serves as a centralized command center that bridges the gap between raw computational power and intelligent model evolution through genetic algorithms and hyperparameter optimization.

The architecture leverages a microservices-based approach with seamless integration to industry-standard tools including MLflow for experiment tracking, MinIO for high-performance object storage, Redis for job queuing, and Ray Tune for scalable hyperparameter optimization. The platform's intuitive interface provides real-time monitoring of cluster performance, GPU utilization, and training progress while maintaining robust role-based access control and team collaboration features.

Key innovations include automated hyperparameter evolution using genetic algorithms, intelligent resource allocation across distributed GPU nodes, and comprehensive experiment lifecycle management. The system supports both development and production deployments through Docker containerization, offering scalable infrastructure that can grow from single-node setups to enterprise-grade GPU clusters. With its API-first design and comprehensive REST interface, NeuroForge AI enables seamless integration into existing MLOps pipelines and CI/CD workflows.

Chapter 2

Introduction

2.1 Background

The rapid evolution of machine learning operations (MLOps) has created a critical need for sophisticated orchestration platforms that can manage the complexity of modern AI workflows. Organizations deploying machine learning models at scale face numerous challenges including resource management, experiment tracking, team collaboration, and infrastructure monitoring. Traditional approaches often involve disparate tools and manual processes, leading to inefficiencies, reduced reproducibility, and increased operational overhead.

NeuroForge AI emerges as a response to these challenges, providing a unified platform that integrates essential MLOps components into a cohesive, user-friendly interface. The platform draws inspiration from industry best practices and addresses common pain points encountered in production ML environments, such as the need for real-time monitoring, streamlined experiment management, and efficient resource utilization.

2.2 Objectives

The primary objectives of NeuroForge AI include:

- **Centralized Management**: Provide a single interface for managing all aspects of ML workflows, from data ingestion to model deployment
- **Real-time Monitoring**: Offer comprehensive dashboard capabilities for tracking system performance, GPU utilization, and training progress
- **Role-based Access**: Implement granular access control to ensure appropriate permissions for different user types
- **Scalable Architecture**: Design a modular system that can grow with organizational needs and handle increasing workloads
- **Developer Experience**: Create an intuitive interface that reduces the learning curve for ML engineers and data scientists

2.3 Scope

This documentation covers the complete implementation of NeuroForge AI, including its architecture, deployment procedures, configuration requirements, and usage patterns. The platform is designed to work with containerized environments and integrates with popular MLOps tools such as MLflow for experiment tracking, Redis for job queuing, and MinIO for object storage. While the core functionality is self-contained, the architecture allows for extensibility and integration with existing enterprise systems.

Chapter 3

Technical Architecture

3.1 System Overview

NeuroForge AI follows a modern web application architecture with clear separation of concerns between frontend presentation, business logic, and external service integrations. The application is built using React with TypeScript, providing type safety and enhanced developer experience. The architecture emphasizes component reusability, state management efficiency, and responsive design principles.

Frontend Layer (React + TypeScript)
State Management (React Hooks)
Service Integration Layer
External Services (MLflow, Redis, MinIO)
Container Runtime (Docker)

Figure 3.1: High-level system architecture layers

3.2 Component Architecture

The application is structured around a modular component architecture where each major feature is encapsulated in its own React component. This approach promotes maintainability and allows for independent development and testing of individual features.

```
// Main application component with state management
const App: React.FC = () => {
  // Application modes
  const [showSplash, setShowSplash] = useState(true);
  const [showPresentation, setShowPresentation] = useState(false);

  // Theme and user management
  const [isDarkMode, setIsDarkMode] = useState<boolean>(true);
  const [userRole, setUserRole] = useState<UserRole>('guest');

  // Service and data management
  const [services, setServices] = useState<Microservice[]>([]);
  const [users, setUsers] = useState<UserProfile[]>([]);
  const [projects, setProjects] = useState<ProjectDefinition[]>([]);

  // Component rendering logic
  return (
    <>
      {showSplash && <SplashScreen />}
      <Sidebar />
      <Header />
      <main>
        {/* Dynamic content based on active service */}
      </main>
    </>
  );
};
```

Listing 3.1: Core Application Structure

3.3 State Management Strategy

NeuroForge AI employs React's built-in state management using hooks, combined with localStorage for persistence. This approach provides a lightweight yet effective solution for managing application state across user sessions.

```
// Persistent state management with localStorage
const [services, setServices] = useState<Microservice[]>(() => {
  const saved = localStorage.getItem('omni_services_config');
  return saved ? JSON.parse(saved) : DEFAULT_MICROSERVICES;
});

// Automatic persistence on state changes
useEffect(() => {
  localStorage.setItem('omni_services_config',
      JSON.stringify(services));
}, [services]);
```

Listing 3.2: State Persistence Pattern

3.4 Service Integration Architecture

The platform integrates with external services through a well-defined configuration system that allows for dynamic service registration and management. Each service is defined as a Microservice object with metadata including URLs, icons, access permissions, and descriptions.

```
interface Microservice {
  id: string;
  name: string;
  description: string;
  url: string;
  icon: ReactNode;
  minRole?: UserRole;
}
```

Listing 3.3: Service Configuration Interface

Chapter 4

Installation Guide

4.1 Prerequisites

Before installing NeuroForge AI, ensure your system meets the following requirements:

- **Node.js**: Version 18.0 or higher
- **Docker**: Latest stable version with Docker Compose
- **Git**: For version control operations
- **Memory**: Minimum 4GB RAM (8GB recommended)
- **Storage**: Minimum 10GB available disk space

4.2 Local Development Setup

4.2.1 Clone and Install Dependencies

```bash
# Clone the repository
git clone <repository-url>
cd NeuralForgeAI

# Install Node.js dependencies
npm install

# Create environment configuration
cp .env.example .env
# Edit .env with your configuration
```

Listing 4.1: Repository Setup

4.2.2 Environment Configuration

Create a `.env` file in the project root with the following configuration:

```
# Gemini AI API Configuration
GEMINI_API_KEY=your_gemini_api_key_here

# Application Configuration
NODE_ENV=development
PORT=3000

# External Service URLs
MLFLOW_URL=http://localhost:5000
REDIS_URL=http://localhost:6379
MINIO_URL=http://localhost:9000
```

Listing 4.2: Environment Variables

4.2.3 Running the Development Server

```
# Start the development server
npm run dev

# The application will be available at:
# http://localhost:3000
```

Listing 4.3: Development Server Startup

4.3 Production Deployment

4.3.1 Docker Deployment

The recommended production deployment method uses Docker containers:

```
# Build and start the container
docker-compose up --build -d

# View container logs
docker-compose logs -f

# Stop the application
docker-compose down
```

Listing 4.4: Docker Deployment Commands

4.3.2 Dockerfile Analysis

The Dockerfile follows a multi-stage build pattern for optimization:

```
# Build stage
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Production stage
FROM node:18-alpine
WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./package.json
EXPOSE 4173
CMD ["npm", "run", "preview", "--", "--host", "0.0.0.0"]
```

Listing 4.5: Dockerfile Structure

Chapter 5

Configuration and Customization

5.1 Service Configuration

NeuroForge AI uses a flexible service configuration system that allows administrators to customize available services based on organizational needs. Services are defined in the constants file and can be extended or modified.

```
export const DEFAULT_MICROSERVICES: Microservice[] = [
  {
    id: 'dashboard',
    name: 'Dashboard',
    description: 'Cluster overview and telemetry',
    url: 'internal:dashboard',
    icon: <LayoutDashboard size={20} />,
  },
  {
    id: 'mlflow',
    name: 'MLflow Tracking',
    description: 'Experiment logging and metrics',
    url: 'http://mlflow.example.com',
    icon: <GitBranch size={20} />,
    minRole: 'admin'
  }
];
```

Listing 5.1: Service Definition Example

5.2 User Role Management

The platform implements role-based access control with two primary roles:

Role	Permissions	Access Level
Admin	Full system access	All services
Guest	Limited access	Non-admin services only

Table 5.1: User role permissions matrix

5.3 Theme Customization

The application supports both light and dark themes with automatic persistence:

```
const handleThemeToggle = () => {
  setIsDarkMode(!isDarkMode);
  const root = window.document.documentElement;
  if (isDarkMode) {
    root.classList.add('dark');
  } else {
    root.classList.remove('dark');
  }
};
```

Listing 5.2: Theme Management Implementation

Chapter 6

Usage Examples

6.1 Dashboard Navigation

The main dashboard provides an overview of system status and quick access to all services. Users can navigate between services using the sidebar navigation or the command palette (Ctrl/Cmd + K).

```
const CommandPalette: React.FC = () => {
  const handleKeyDown = (e: KeyboardEvent) => {
    if ((e.metaKey || e.ctrlKey) && e.key === 'k') {
      e.preventDefault();
      setShowCommandPalette(true);
    }
  };

  return (
    <div className="command-palette">
      {/* Search and service selection interface */}
    </div>
  );
};
```

Listing 6.1: Command Palette Implementation

6.2 Project Management

The project registry allows administrators to manage ML experiments and track their lifecycle:

```
interface ProjectDefinition {
  id: string;
  name: string;
  description: string;
  createdAt: string;
}

const handleAddProject = (project: ProjectDefinition) => {
  setProjects(prev => [...prev, project]);
};
```

Listing 6.2: Project Management Interface

6.3 Training Launch

The training launch interface provides a streamlined way to start new ML experiments:

```
const LaunchTrainingView: React.FC = () => {
  const [selectedProject, setSelectedProject] = useState<string>('');
  const [selectedUser, setSelectedUser] = useState<string>('');

  const handleLaunchTraining = () => {
    // Training launch logic
    console.log(`Launching training for project: ${selectedProject}`);
  };

  return (
    <div className="training-launch">
      {/* Training configuration form */}
    </div>
  );
};
```

Listing 6.3: Training Launch Configuration

Chapter 7

Performance Metrics

7.1 System Monitoring

The dashboard provides real-time monitoring of key system metrics:

Metric	Description
Active Workers	Number of currently running worker nodes
GPU Utilization	Current GPU usage percentage
Queue Depth	Number of pending jobs in queue
Storage Used	Total storage consumption
Redis Memory	Redis memory usage
MinIO Bandwidth	Current I/O throughput

Table 7.1: System monitoring metrics

7.2 Performance Optimization

The application implements several optimization strategies:

- **Lazy Loading**: Components are loaded on-demand to reduce initial bundle size
- **State Persistence**: LocalStorage caching reduces API calls
- **Responsive Design**: Optimized for various screen sizes
- **Keyboard Shortcuts**: Power user features for efficiency

7.3 Expected Performance

Metric	Target	Acceptable Range
Initial Load Time	< 2 seconds	< 3 seconds
Navigation Response	< 500ms	< 1 second
Memory Usage	< 100MB	< 200MB
Bundle Size	< 500KB	< 1MB

Table 7.2: Performance targets and thresholds

Chapter 8

Best Practices and Troubleshooting

8.1 Development Best Practices

8.1.1 Code Organization

- Maintain clear separation between components and business logic
- Use TypeScript interfaces for all data structures
- Implement proper error handling and loading states
- Follow React hooks best practices
- Use consistent naming conventions

8.1.2 State Management

- Keep state as close to where it's used as possible
- Use localStorage for persistence of user preferences
- Implement proper cleanup in useEffect hooks
- Avoid unnecessary re-renders through memoization

8.2 Common Issues and Solutions

8.2.1 Build Failures

Issue: Build fails with TypeScript errors
Solution: Ensure all interfaces are properly typed and imports are correct
Issue: Docker build fails during npm install
Solution: Clear npm cache and rebuild: `npm cache clean -force`

8.2.2 Runtime Issues

Issue: Application doesn't render in browser
Solution: Check that JavaScript bundle is properly generated and referenced
Issue: External services not accessible
Solution: Verify service URLs in configuration and network connectivity

8.2.3 Performance Issues

Issue: Slow initial load time
Solution: Implement code splitting and lazy loading
Issue: Memory leaks in browser
Solution: Ensure proper cleanup of event listeners and timers

8.3 Debugging Techniques

```
// Enable debug mode
const DEBUG = process.env.NODE_ENV === 'development';

if (DEBUG) {
  console.log('Debug mode enabled');
  // Additional debug logging
}

// Error boundary implementation
class ErrorBoundary extends React.Component {
  componentDidCatch(error: Error, errorInfo: React.ErrorInfo) {
    console.error('Application error:', error, errorInfo);
  }
}
```

Listing 8.1: Debugging Configuration

Chapter 9

Conclusions and Future Work

9.1 Project Summary

NeuroForge AI successfully demonstrates a modern approach to MLOps orchestration, providing a comprehensive platform for managing machine learning workflows. The implementation showcases best practices in React development, TypeScript usage, and containerized deployment strategies. The platform's modular architecture and extensible design make it suitable for various organizational needs and scales.

9.2 Key Achievements

- Successfully integrated multiple MLOps tools into a unified interface
- Implemented responsive design that works across devices
- Created a scalable architecture that supports future enhancements
- Established proper development and deployment workflows
- Provided comprehensive documentation and examples

9.3 Future Enhancements

9.3.1 Short-term Goals

- Enhanced monitoring and alerting capabilities
- Integration with additional ML frameworks
- Improved mobile responsiveness
- Advanced user analytics and reporting

9.3.2 Long-term Vision

- Multi-cloud deployment support
- Advanced AI-powered automation features
- Integration with enterprise authentication systems
- Real-time collaboration features
- Advanced experiment comparison tools

9.4 Technical Debt and Improvements

- Migration to more robust state management solution
- Implementation of comprehensive testing suite
- Enhanced error handling and recovery mechanisms
- Performance optimization for large-scale deployments

Chapter 10

Bibliography

Bibliography

[1] React Documentation Team. *React 18 Documentation*. React, 2023. Available at: https://react.dev

[2] Microsoft Corporation. *TypeScript Handbook*. Microsoft, 2023. Available at: https://www.typescriptlang.org/docs

[3] Docker Inc. *Docker Documentation*. Docker, 2023. Available at: https://docs.docker.com

[4] Google Cloud. *MLOps: Continuous delivery and automation pipelines in machine learning*. Google Cloud, 2023.

[5] Vite Team. *Vite Documentation*. Vite, 2023. Available at: https://vitejs.dev

Appendix A

Advanced Configuration

A.1 Custom Service Integration

To add custom services to NeuroForge AI, modify the service configuration:

```
const customService: Microservice = {
  id: 'custom-service',
  name: 'Custom ML Service',
  description: 'Custom machine learning service',
  url: 'https://custom-service.example.com',
  icon: <CustomIcon size={20} />,
  minRole: 'admin'
};

// Add to services array
const services = [...DEFAULT_MICROSERVICES, customService];
```

Listing A.1: Custom Service Addition

A.2 Environment Variables Reference

Variable	**Description**
GEMINI_API_KEY	Google Gemini AI API key
NODE_ENV	Environment mode (development/production)
PORT	Application port number
MLFLOW_URL	MLflow server URL
REDIS_URL	Redis server URL
MINIO_URL	MinIO server URL

Table A.1: Complete environment variables reference

A.3 Docker Compose Configuration

```
version: '3.8'

services:
  app:
    build: .
    image: wisrovi/neuralforgeai:latest
    ports:
      - "5810:4173"
    environment:
      - GEMINI_API_KEY=${GEMINI_API_KEY}
    env_file:
      - .env
    restart: unless-stopped

  redis:
    image: redis:alpine
    ports:
      - "6379:6379"
    restart: unless-stopped

  mlflow:
    image: python:3.9-slim
    ports:
      - "5000:5000"
    command: mlflow server --host 0.0.0.0
    restart: unless-stopped
```

Listing A.2: Complete Docker Compose Setup

A.4 API Integration Examples

```
const fetchDashboardData = async () => {
  try {
    const response = await
       fetch(DASHBOARD_API_CONFIG.activeWorkers.url, {
      method: DASHBOARD_API_CONFIG.activeWorkers.method,
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(DASHBOARD_API_CONFIG.activeWorkers.payload)
    });

    const data = await response.json();
    return data;
  } catch (error) {
    console.error('Failed to fetch dashboard data:', error);
    return null;
  }
};
```

Listing A.3: External API Integration

Acknowledgments

Special thanks to the open-source community

for providing the tools and libraries that made this project possible

React, TypeScript, Vite, Docker, and countless contributors

Building the Future of MLOps Together