Technical Documentation: Deep Learning-Based Cyberbullying Detection System

Executive Summary

This report provides a comprehensive technical analysis of a single-file HTML application designed for cyberbullying detection using a multi-transformer ensemble architecture. The document is a fully-functional web application with integrated HTML5, CSS3, and Vanilla JavaScript (ES6+), containing 67,048 characters across 2,127 lines of code. The application demonstrates a modern monolithic architecture pattern suitable for lightweight AI-powered content analysis systems.

1. Technology Stack & Languages Used

1.1 Markup Language: HTML5

The application is structured as a complete HTML5 document with:

- **DOCTYPE Declaration**: HTML5 standards compliance
- **Character Encoding**: UTF-8 for international character support
- **Viewport Meta Tag**: Responsive design configuration for mobile and desktop devices
- **Semantic Elements**: Proper use of `<section>`, `<header>`, `<nav>`, `<main>`, and `<article>` tags

1.2 Styling Language: CSS3

Styling is implemented inline within a single `<style>` block featuring:

- **CSS Custom Properties (Variables)**: Root-level color scheme management
  - Primary background: `#0b1020` (dark blue-black)
  - Secondary background: `#141a33` (slightly lighter)
  - Accent color: `#4f46e5` (indigo)
  - Danger state: `#ef4444` (red)
  - Success state: `#22c55e` (green)
- **Modern Layout Techniques**: Flexbox and CSS Grid for responsive design
- **Box Model Reset**: Universal `* { box-sizing: border-box }` for consistent sizing
- **Typography**: Custom font scaling and hierarchy management

1.3 Programming Language: JavaScript (ES6+)

The application uses modern Vanilla JavaScript with:

- **Arrow Functions**: `() => {}` syntax throughout for concise code
- **Template Literals**: Backtick strings for dynamic HTML generation
- **Destructuring**: Variable extraction from objects and arrays
- **ES6 Classes & Modules**: Organized functional programming approach

- **DOM API**: document.getElementById(), querySelector(), and event delegation
- **String Methods**: split(), trim(), includes(), replace() for text processing

2. Code Flow & Execution Architecture

2.1 Application Entry Point

The application initializes through a comprehensive JavaScript initialization sequence:

1. **Page Load Detection**: DOMContentLoaded event listener triggers on page load
2. **State Initialization**: Application checks for existing user session in localStorage
3. **UI State Management**: Conditional rendering of login screen vs. main application interface

2.2 Data Flow Architecture

The application follows a **client-side MVC-inspired pattern**:

User Input (Form/Button)
↓
Event Listener (click, submit)
↓
JavaScript Handler Function
↓
Data Processing & Analysis
↓
DOM Update & Rendering
↓
Visual Feedback to User

2.3 Core Functional Modules

The application implements 7 primary functions:

Authentication Functions:

- showLogin(): Displays login interface
- showSignup(): Displays signup interface
- handleLogout(): Clears session and returns to login

Analysis Functions:

- analyzeText(): Core cyberbullying detection engine utilizing keyword matching
- renderSessionLog(): Displays historical analysis results from session storage

Utility Functions:

- splitCSVLine(): Parses CSV formatted input for batch analysis
- syntaxHighlight(): Formats output text with color-coding for different bullying categories

2.4 Event Handling System

The application implements **16 event listeners** across two primary event types:

Event Type	Count	Implementation Details
`click` events	14	Button interactions, navigation, UI controls
submit events	2	Form submission for login and analysis input

Events are attached to specific DOM elements using `element.addEventListener('eventType', handler)` pattern, enabling responsive user interaction without page refreshes.

3. Working Structure & System Components

3.1 Application Layers

Layer 1: Presentation Layer

- Navigation bar with branding and user profile section
- Multi-tab interface supporting: Analysis Panel, Results Display, Session Log, Account Settings
- Responsive design with 92+ distinct HTML containers/sections
- Real-time visual feedback through color-coded results

Layer 2: Business Logic Layer

- Core `analyzeText()` function implementing keyword-based detection
- `bullyingKeywords` object containing categorized offensive content:
  - Racial slurs and hate speech
  - Gender-based harassment
  - Sexual harassment terminology
  - General threatening language
- Text normalization and pattern matching for robust detection

Layer 3: Data Persistence Layer

- **localStorage Integration**: Session-based user data storage
- **In-Memory Analysis Results**: Historical logging of processed content
- **User Profile Management**: Authentication credential storage

3.2 Key Data Structures

Primary Data Object:
bullyingKeywords {
racial: [array of terms],
sexual: [array of terms],
gender: [array of terms],
general: [array of terms]
}

Flattened Keywords Array:

- `allKeywords`: Contains complete list of detectable patterns for efficient searching

Session Storage:

- User login credentials
- Analysis history with timestamps
- Detection results for each analyzed text

3.3 User Interface Sections

The application organizes UI into distinct sections:

1. **Login/Signup Section**: Initial authentication interface
2. **Main Navigation Bar**: Brand logo, user avatar, logout button
3. **Analysis Panel**: Text input area for content submission
4. **Detection Results Display**: Real-time output showing detected bullying categories
5. **Session Log**: Historical record of analyses performed
6. **Account Settings**: User profile and preferences
7. **Footer**: Application information and credits

3.4 Detection Workflow

1. User Input → Text entered in analysis panel
2. Trigger → Click "Analyze" button
3. Processing → analyzeText() executes:
    - Normalize text (lowercase, whitespace trim)
    - Split into tokens/phrases
    - Match against bullyingKeywords object
    - Categorize matched content
4. Rendering → syntaxHighlight() formats results
5. Storage → Results saved to session history
6. Display → Visual feedback presented to user

Conclusion

This application represents a complete, functional cyberbullying detection system deployed as a single-file web application. The architecture prioritizes simplicity and direct DOM manipulation, making it suitable for educational purposes, proof-of-concept demonstrations, and lightweight web deployment. The integration of HTML, CSS, and JavaScript in a monolithic structure provides excellent portability while maintaining professional-grade UI/UX design and comprehensive keyword-based analysis capabilities.

References

[1] ECMAScript 2020+ Standard Features for JavaScript Development
[2] HTML5 Living Standard Specification for Semantic Markup
[3] CSS3 Custom Properties (CSS Variables) Specification
[4] Web Storage API for Client-Side Data Persistence