Software Deployment Guide: WebQAmGaze UDA Dyslexia Classifier

Version: 1.1

Date: November 20, 2025

Prepared For: Web Systems & Backend Engineering Team

Status: Released for Integration

I. INTRODUCTION

1.1 Purpose

This document specifies the technical requirements, architecture, and data pipeline necessary to deploy the **WebQAmGaze UDA (Unsupervised Domain Adaptation) Dyslexia Classifier**. This system enables high-accuracy dyslexia screening using standard consumer webcams, bridging the domain gap between laboratory-grade eye trackers and web-based inputs.

1.2 Scope

The system accepts raw, noisy gaze data from a web client (via `webgazer.js`), processes it through a specialized cleaning and normalization pipeline, and uses a deep learning model to output a binary risk assessment (High Risk / Low Risk) for dyslexia.

1.3 Definitions

- **UDA (Unsupervised Domain Adaptation):** A machine learning technique used to make the model invariant to the source of the data (Webcam vs. Eye Tracker).
- **Fixation:** A period where the eye is relatively still, processing information.
- **Saccade:** A rapid movement of the eye between fixations.
- **Single-Stream:** The model requires only one type of reading task ("Ordinary Reading") rather than multiple task types.

II. SYSTEM ARCHITECTURE

2.1 Overview

The inference system operates as a linear pipeline: `Raw Web Input` → `Preprocessing (Python)` → `Feature Scaling` → `Deep Learning Inference` → `Risk Score`.

2.2 Component Description

Component	Filename / Identifier	Role	Criticality
Classifier Model	`dyslexia_uda_classifier.h5`	The final trained Keras model. Accepts subject-level data and outputs a risk probability.	**Critical**

Target Scaler	`target_domain_scaler.pkl`	A Scikit-Learn `StandardScaler` fitted on the webcam domain. Used to normalize features.	**Critical**
Input Data	JSON / API Payload	Raw time-series data of (x, y, t) coordinates from the frontend.	**Input**

III. DATA INTERFACE SPECIFICATION

3.1 Input Format (Frontend $\rightarrow$ Backend)

The backend API endpoint should accept a JSON payload containing the raw gaze path for a single reading session.


```
{
  "participant_id": "user_123",
  "screen_width": 1920,
  "screen_height": 1080,
  "gaze_data": [
    {"x": 100, "y": 500, "t": 1630000001},
    {"x": 102, "y": 505, "t": 1630000034},
    ...
  ]
}
```


3.2 Output Format (Backend $\rightarrow$ Frontend)


```
{
  "risk_score": 0.78,
  "label": "High Risk",
  "status": "success"
}
```


IV. PREPROCESSING PIPELINE (MANDATORY IMPLEMENTATION)

The raw data **must** undergo the following transformation steps before being fed to the model. Failure to follow these exact steps will result in model failure.

4.1 Spatial Normalization (Resolution Independence)

Webcams have different resolutions. All coordinates must be mapped to a relative $[0, 1]$ space.

$$x_{norm} = \frac{x_{px}}{Screen\_Width}$$

$$y_{norm} = \frac{y_{px}}{Screen\_Height}$$

Constraint: Discard any points where $x < 0$, $x > 1$, $y < 0$, or $y > 1$.

4.2 Signal Smoothing

Webcam data contains high-frequency jitter. Apply an Exponential Moving Average (EMA) or similar smoothing filter (alpha ≈ 0.5) to (x, y) coordinates before velocity calculation.

4.3 I-VT Fixation Detection

Convert the continuous stream of points into discrete "Fixation Events."

- **Algorithm:** I-VT (Identification by Velocity Threshold).
- **Logic:** Calculate velocity between points. If velocity < 0.5 (normalized units/sec), group points into a fixation.
- **Minimum Duration:** 50ms. Fixations shorter than this are noise.

4.4 Feature Engineering

For each detected fixation, extract the following 5 features in this specific order:

1. **Duration:** (ms)
2. **Centroid X:** (Normalized 0-1)
3. **Centroid Y:** (Normalized 0-1)
4. **Amplitude:** Euclidean distance from the previous fixation (Normalized units).
5. **Regression Flag:** 1 if current $x <$ previous x, else 0.

4.5 Feature Scaling

CRITICAL: You generally cannot feed raw features to a neural network.

- **Action:** Load `target_domain_scaler.pkl`.
- **Operation:** Apply `scaler.transform()` to the feature matrix.
- **Note:** Do *not* fit a new scaler. Use the provided artifact.

4.6 Sequence Shaping (Subject-Level Batching)

The model expects a fixed input shape representing "One Person."

- **Windowing:** Slice the fixations into overlapping windows of length **20** (Step=5).
- **Aggregation:** Collect up to **82** of these windows.
- **Padding:** If the user has fewer than 82 windows, pad with zeros.
- **Final Input Tensor Shape:** `(1, 82, 20, 5)`

V. REAL-TIME INFERENCE LOGIC

The following is the recommended logic for the web system to perform a diagnosis.

5.1 On Server Start

The application must load the heavy ML assets into memory **once** during initialization, not per request.

- Load `dyslexia_uda_classifier.h5` (TensorFlow Model).
- Load `target_domain_scaler.pkl` (Scikit-Learn Scaler).

5.2 During the Screening Session

- The frontend streams or buffers raw gaze data (x, y, t) as the user reads the text.
- **Note:** Unlike previous versions, this model is **Single-Stream**. The user only needs to perform **one** task: "Ordinary Reading." There is no need to separate Syllables or Pseudo-text.

5.3 End-of-Session Execution Flow

Once the reading task is complete, the backend triggers the diagnosis:

1. **Receive Payload:** Accept the full list of raw gaze points.
2. **Run Pipeline:** Execute **Steps 4.1 through 4.6** (Preprocessing) defined in Section 4.
3. **Shape Validation:** Ensure the resulting tensor is exactly `(1, 82, 20, 5)`.
    - *Batch Dimension:* 1 (One user).
    - *Subject Sequence Length:* 82 (The model's fixed view of a "person").
    - *Time Steps:* 20 (Fixations per sequence).
    - *Features:* 5.
4. **Inference:** Pass the tensor to `model.predict()`.
5. **Interpretation:**
    - The model returns a float between 0.0 and 1.0.
    - **Score < 0.5:** Return "Low Risk" (Neurotypical Reading Pattern).
    - **Score >= 0.5:** Return "High Risk" (Dyslexic-Type Reading Pattern).

VI. DEPLOYMENT & ENVIRONMENT

6.1 Python Dependencies

The inference service requires the following libraries:

- `tensorflow >= 2.10`
- `numpy`
- `pandas`
- `scikit-learn` (must be compatible with the version used to save `scaler.pkl`)

6.2 Error Handling

- **Insufficient Data:** If a user produces fewer than 10 valid sequences (approx. 15 seconds of reading), the system should return an error: *"Insufficient Data. Please ensure good lighting and read for the full duration."*
- **High Noise:** If the `Regression Flag` mean > 0.4 (40% regressions), it likely indicates head movement noise rather than dyslexia. Consider flagging for "Quality Review."

VII. CONCLUSION

This pipeline successfully adapts a high-performance eye-tracking model for webcam usage. The key to reliability is the strict adherence to **Step 4.1 (Normalization)** and **Step 4.5 (Scaling)** using the