Deployment and Integration Guide:

The Language-Agnostic "Reading Profile" Dyslexia Classifier

Prepared for: The Backend & Web Systems Team Date: November 17, 2025

ABSTRACT

This document provides the complete technical specifications for the deployment of the `dyslexia_profile_model`, a deep learning model for the high-accuracy, language-agnostic screening of dyslexia. The system is designed to analyze gaze-tracking data from a participant performing three distinct reading tasks (`Syllables`, `MeaningfulText`, `PseudoText`) and render a binary classification.

The model architecture is a **Multi-Input "Reading Profile" Classifier** that leverages a pre-trained, shared gaze encoder (`gaze_encoder_pretrained.h5`) to interpret eye-movement dynamics. This document details the model's architecture, the critical role of its core components, and the exact, non-negotiable data preprocessing pipeline required for successful real-world inference. This pipeline includes specific instructions for feature calculation, spatial normalization, data scaling via a provided `scaler.pkl` object, and input tensor padding.

I. INTRODUCTION

The goal of this project is to create a reliable, accessible screening tool for dyslexia. The final model, `dyslexia_profile_model.h5`, achieves this by analyzing the *kinematic profile* of a user's eye movements, which is a language-agnostic signal.

This system moves beyond simple classification of a single reading sample. It is designed to mimic the diagnostic process of a human expert by explicitly comparing a participant's reading performance across tasks of varying cognitive load.

This document will provide all necessary information for the backend team to build a web system that can successfully capture, process, and feed data to this model to get reliable predictions.

II. SYSTEM ARCHITECTURE & DESIGN RATIONALE

The final model is a **Multi-Input, Single-Task Classifier**. It was chosen over simpler architectures due to its superior performance and diagnostic power.

A. The "Reading Profile" Model

The model has three separate input branches, one for each diagnostic reading task:

1. `input_syllables`
2. `input_meaningful`
3. `input_pseudo`

Each input stream is processed by a shared, pre-trained encoder. The outputs of this encoder are then aggregated and concatenated into a single "profile vector." This vector, which represents the child's comparative performance across all three tasks, is fed to a final classifier head for a single prediction.

B. Rationale for this Architecture

During development, we explored several alternative plans. The "Reading Profile" model was chosen for the following critical reasons:

- **Failure of Simpler Models:**
  - **Plan 1 (Three Separate Models):** We rejected the idea of training three separate models (one for each task). This approach suffers from **data fragmentation**, increasing the risk of overfitting. More importantly, it fails to capture the most powerful diagnostic signal: the *comparative difference* in reading behavior *between* the tasks.
  - **Plan 2 (Task as a Feature):** We also rejected a simpler model that would take a single reading sequence and be told its task type (e.g., via a feature). This provides only a weak, implicit signal of the task's context.
- **Superiority of the Profile Model:** Our chosen architecture explicitly models the diagnostic process. The final classifier head is built to learn from the *relationship* between the three task profiles simultaneously. This allows it to learn complex patterns like, "This participant's gaze behavior on Pseudo-Text was *significantly worse* than on Meaningful-Text, a classic indicator of dyslexia." This approach yielded the highest accuracy (0.78 F1-score) and is the most clinically valid.

III. CORE COMPONENTS & DEPLOYMENT ASSETS

To run the model, the backend system requires **three** essential files.

1. `dyslexia_profile_model.h5` **(The "Brain")**
   - **What it is:** The final, trained Keras model. It contains the complete multi-input architecture and all the learned weights for both the encoder and the final classifier head.
   - **Requires:** TensorFlow/Keras.
2. `gaze_encoder_pretrained.h5` **(The "Foundation")**
   - **What it is:** A foundational LSTM-based encoder that was pre-trained on the massive OneStop dataset (150+ hours of reading).
   - **Its Role:** This model is **not called directly**. It is already integrated *inside* the `dyslexia_profile_model.h5` as a shared, frozen `TimeDistributed` layer. Its purpose was to learn the universal, language-agnostic "grammar" of human eye movements (what a fixation, saccade, and regression look like) *before* it ever saw dyslexic data. This transfer learning approach is the key to the model's high accuracy.
3. `scaler.pkl` **(The "Key")**
   - **What it is:** A `StandardScaler` object from Scikit-learn, saved as a pickle file.
   - **CRITICAL IMPORTANCE:** This is the most important file for the preprocessing pipeline. The `gaze_encoder` was pre-trained on data normalized with this *exact* scaler. All new, real-world data from a Tobii tracker **must** be transformed with this same scaler to match the numerical range and distribution the model expects. **Failure to use this scaler will result in 100% incorrect predictions.**

IV. DATA PREPROCESSING PIPELINE

This section details the exact, non-negotiable data pipeline required to convert raw gaze data from a Tobii tracker into the format the model expects.

The goal is to produce three tensors (one for each task) with the final shape `(100, 20, 5)`.

Step 1: Gaze Event Detection (Raw Stream -> Fixations)

The system must consume the raw gaze stream from the Tobii tracker (`(x, y, timestamp)`) and use a fixation detection algorithm (e.g., an I-VT or velocity-based algorithm) to convert it into a list of fixation events. The output of this step should be a list of fixations, each with:

- `start_ms`
- `end_ms`
- `fixation_x` (average x of points in the fixation)
- `fixation_y` (average y of points in the fixation)

Step 2: Core Feature Calculation

From the list of fixations for a single task, construct a DataFrame and calculate the following 5 core features.

1. `fixation_duration` : `end_ms - start_ms`
2. `fixation_x` : The average `x` pixel coordinate.
3. `fixation_y` : The average `y` pixel coordinate.
4. `saccade_amplitude_in` : The Euclidean distance from the *previous* fixation. `sqrt((x_i - x_i-1)² + (y_i - y_i-1)²)` . (This will be `0` for the first fixation of a trial).
5. `saccade_velocity_in` : The amplitude divided by the time *between* fixations. `saccade_amplitude_in / (start_ms_i - end_ms_i-1)` . (This will also be `0` for the first fixation).

Step 3: Data Cleaning

Apply the same cleaning rule used in training. Remove all fixations from the list where:

- `fixation_duration < 80` (ms)
- `fixation_duration > 1000` (ms)

Step 4: Spatial Harmonization (Normalization)

This step is critical to make the data resolution-independent. We must convert the pixel-based coordinates from the Tobii (which should be running at the same **1680x1050** resolution as the training data) into a `[0, 1]` scale.

- `x_norm = fixation_x / 1680.0`
- `y_norm = fixation_y / 1050.0`
- `amp_norm = saccade_amplitude_in / 1960.53` (where 1960.53 is the screen diagonal, `sqrt(1680² + 1050²)`)

Step 5: Feature Scaling (The "Key")

1. Load the `scaler.pkl` file: `scaler = pickle.load(open('scaler.pkl', 'rb'))` .

2. Create a DataFrame from your calculated features in this **exact order**: `['fixation_duration', 'x_norm', 'y_norm', 'amp_norm', 'saccade_velocity_in']`
3. Use the scaler to transform this data: `scaled_features = scaler.transform(feature_df)`

Step 6: Sequencing (The Sliding Window)

The `scaled_features` are now a `(num_fixations, 5)` array. We must convert this into a list of 20-fixation sequences.

- `SEQUENCE_LENGTH = 20`
- `STEP = 5`
- Iterate from `i = 0` to `(num_fixations - SEQUENCE_LENGTH)` with a step of `5`.
- For each `i`, slice the array: `sequence = scaled_features[i : i + SEQUENCE_LENGTH]`
- Append this `(20, 5)` sequence to a list.

Step 7: Padding (The Final Dimension)

Our model was trained on data with a fixed shape, which requires padding.

1. The `max_len` (maximum number of sequences for any participant) found during training was **100**.
2. Your final input tensor for each task **must** have the shape `(100, 20, 5)`.
3. Take the list of sequences from Step 6 (e.g., you have 35 sequences, shape `(35, 20, 5)`).
4. Create a "container" array of zeros with the shape `(100, 20, 5)`.
5. Copy your 35 sequences into the beginning of this container: `padded_tensor[:35] = your_sequences`.
6. This `padded_tensor` is the final input for one of the model's branches.

V. REAL-TIME INFERENCE & DEPLOYMENT GUIDE

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

1. **On Server Start:**
   - Load `dyslexia_profile_model.h5` into memory.
   - Load `scaler.pkl` into memory.
2. **During the Screening Session:**
   - The user must complete all three reading tasks sequentially.
   - For `Task 1 (Syllables)`:
     - The application streams gaze data from the Tobii.
     - The backend runs the **full preprocessing pipeline (Steps 1-6)** on this stream.
     - Store the resulting list of sequences (e.g., `list_sequences_syl`).
   - Repeat for `Task 2 (MeaningfulText)` and `Task 3 (PseudoText)`, storing their sequences in `list_sequences_mean` and `list_sequences_pse`.
3. **At the End of the Session (Making the Prediction):**

- **Padding:** Run **Step 7 (Padding)** on each of the three sequence lists to create the three final tensors: `tensor_syl`, `tensor_mean`, and `tensor_pse`. Each must have the shape `(100, 20, 5)`.
- **Prepare Input Dictionary:** The model expects a dictionary of inputs.

```
model_input = {
    'input_syllables': np.expand_dims(tensor_syl, 0),  # Add batch dim
    'input_meaningful': np.expand_dims(tensor_mean, 0), # Add batch dim
    'input_pseudo': np.expand_dims(tensor_pse, 0)   # Add batch dim
}
```

- **Run Inference:** `prediction_probability = model.predict(model_input)[0][0]`
- This will return a single float (e.g., `0.78`). A value > 0.5 indicates a prediction of dyslexia.

VI. CONCLUSION

The "Reading Profile" model is a powerful and highly accurate tool. Its performance is critically dependent on the exact replication of the data preprocessing pipeline detailed in this document. The two essential assets for deployment are the model file (`dyslexia_profile_model.h5`) and, most importantly, the data scaler (`scaler.pkl`).