Project Workflow: A Dual-Module Bot Detection System

This workflow outlines the development, implementation, and deployment of a sophisticated bot detection system. It is based on the provided Python scripts: web_log_detection_bot.py, mouse_movements_detection_bot.py, and fusion.py. The system employs a dual-module architecture that analyzes user behavior through web logs and mouse biometrics, combining their outputs with a decision-level fusion strategy for a final, robust classification.

---

1. Problem Statement & Proposed Solution

- **Problem:** Traditional CAPTCHA systems are often intrusive and susceptible to advanced bots. A more intelligent, passive system is needed.
- **Solution Architecture:** Develop a multi-layered machine learning framework composed of three core components, as implemented in the provided code:
    1. A **Web Log Detection Module** (web_log_detection_bot.py) that analyzes server-side request patterns.
    2. A **Mouse Movement Detection Module** (mouse_movements_detection_bot.py) that analyzes client-side behavioral biometrics.
    3. A **Fusion Module** (fusion.py) that intelligently combines the scores from the two detector modules to make a final, high-confidence decision.

---

2. Data Collection & Preprocessing

The system relies on two distinct data streams, each with its own specific preprocessing pipeline.

- **Web Logs:**
    - **Source:** The system processes raw **Apache2 log files**.
    - **Sessionization:** The extract_sessions function parses these logs to group user activity into sessions based on **PHP session IDs**. A session is considered complete after a **30-minute inactivity timeout**, and the code is capable of splitting a single log into multiple sessions if significant time gaps are detected.
- **Mouse Movements (Behavioral Biometrics):**

- **Source:** The system ingests user interaction data from **JSON files** that contain mouse coordinates and timestamps.
- **Grouping:** The _preprocess_mouse_movements function groups raw mouse movements into distinct pages. In early-phase data, it estimates page changes by detecting time gaps of over 5 seconds between movements; in later phases, it uses explicit URL data for more accurate grouping.
- **Matrix Conversion:** For each page, the mouse trajectory is converted into a **2D matrix representation** of shape (480, 1320, 1). The value at each (y, x) coordinate in the matrix is the time delta (dt)—the time the user's mouse spent at that point—providing a rich visual pattern for the model.

---

3. Feature Engineering & Behavior Analysis

Each module transforms its raw data into a feature set optimized for its specific machine learning model.

- **Web Log Module:**
  - **Feature Extraction:** The extract_features function calculates **19 specific, pre-selected features** for each session. These features, defined in the self.selected_features list, include metrics like total_requests, session_time, browse_speed, percent_http_4xx_requests, and sd_inter_request_times.
  - **Normalization:** The features are normalized using sklearn.preprocessing.StandardScaler to ensure consistent scale before being fed into the model.
- **Mouse Movement Module:**
  - **Direct Feature Representation:** The feature for this module *is* the preprocessed **480x1320x1 matrix** itself. The Convolutional Neural Network is designed to automatically learn the relevant spatial and temporal features directly from these matrix representations of mouse paths.

---

4. Machine Learning Model Development & Training

The project uses two distinct, specialized models trained sequentially to handle increasingly complex bot behaviors.

- **Module 1: Web Log Classifier (web_log_detection_bot.py)**

- **Model Architecture:** An **Ensemble VotingClassifier** is used, combining the strengths of four underlying models: SVC, MLPClassifier, RandomForestClassifier, and AdaBoostClassifier.
- **Training Strategy:** The train_sequentially method demonstrates a continuous improvement pipeline. The model is first initialized with hyperparameters optimized for "Phase 1" data, and then re-initialized with different, fine-tuned parameters for "Phase 2" data, allowing it to adapt to more advanced threats over time.
- **Output:** The trained model, along with its scaler and feature list, is saved as a single **.pkl file** using pickle.

- **Module 2: Mouse Movement Classifier (mouse_movements_detection_bot.py)**
  - **Model Architecture:** A **Convolutional Neural Network (CNN)** is built using tensorflow.keras. The architecture consists of stacked Conv2D and MaxPooling2D layers, followed by a Flatten and Dense layer with a softmax activation function to classify movements as human or bot.
  - **Training Strategy:** This module also follows a sequential training approach, first training on Phase 1 datasets (moderate bots) and then incrementally training on Phase 2 datasets (advanced bots).
  - **Output:** The final trained CNN is saved as a **.h5 file**.

---

5. Model Fusion & Final Verification (fusion.py)

This is the central nervous system of the project, where intelligence from both modules is combined for a final verdict.

1. **Model Loading:** The BotDetectionFusion class initializes by loading the trained .pkl web log model and the .h5 mouse movement model.
2. **Score Prediction:** For a given user session, it uses the loaded models to generate a web_log_score and a mouse_score, each representing the probability of the user being a bot.
3. **Decision-Level Fusion Logic:** The core fuse_scores method implements a precise, two-tier logic:
   - **Tier 1 (High Confidence Rule):** If the mouse_score is highly definitive (i.e., greater than **0.7** or less than **0.3**), the system trusts it exclusively, and the final score is simply the mouse_score. This prioritizes the harder-to-spoof biometric data.

    - **Tier 2 (Weighted Average):** If the mouse_score falls within the moderate range [0.3, 0.7], the system hedges its bet by calculating a weighted average: final_score = (0.5 * mouse_score) + (0.5 * web_log_score).

4. **Final Classification & Verification:** The classify_session method compares this final_score against a **final threshold of 0.5**. If the score exceeds this threshold, the user is flagged as a bot (is_bot: True), which would trigger secondary verification methods (e.g., an interactive puzzle CAPTCHA).

---

6. Implementation and Continuous Improvement

The provided scripts form a complete, end-to-end system ready for deployment and iteration.

- **System Orchestration:** The fusion.py script acts as the main entry point for processing a live session. Its process_session function orchestrates the entire pipeline: scoring with individual models, applying fusion logic, and returning a comprehensive dictionary with the final decision and all intermediate data.
- **Performance Monitoring:** The evaluate_fusion_performance method in fusion.py provides a built-in way to monitor the system's health by calculating key metrics like **accuracy, precision, recall, and F1-score**, ensuring the system remains effective and fair over time.