TECHNICAL, GRAPHICAL, AND FUNCTIONAL DESIGN

Facial Recognition and Pain Detection in Stroke Patients

Adam Sabsabi	585693	585693@student.inholland.nl
Fejsal Aziz	581934	581934@student.inholland.nl
Maria Diaconescu	683265	683265@student.inholland.nl
Mohamed Dinle	700152	700152@student.inholland.nl

Client: Inholland University of Applied Sciences, Robotics Department

Company supervisor: Alistair Vardy & Margo van Kemenadel

Date: 02/05/2025

Version: 2.0

Contents

1. Graphical Design

This section elaborates on how the user interface was visually structured for clarity and usability in a healthcare setting. The Streamlit interface follows a step-by-step interaction model starting with image upload, followed by stroke detection, automatic routing to the correct pain model, and then display of pain intensity results. The layout has been optimized for clinical workflows where simplicity and efficiency are crucial. Design mockups were created using Figma-style wireframes to visualize how users engage with the system. These visual assets helped simulate and validate design logic before full implementation.

Model UI Design Breakdown:

The Stroke Detection Model determines the facial side affected by stroke. The UI reflects this output visually to help clinicians interpret the result quickly.

The Pain Detection Models (left and right) are automatically selected based on the stroke output, enabling dynamic routing. The interface remains unified but intelligently responsive to side detection.

The graphical design of our proof-of-concept is built using Streamlit, a powerful and fast Python-based frontend tool. The layout is minimal, with step-by-step components:
- Step 1: Upload interface for full-face image
- Step 2: Automatic stroke-affected side detection
- Step 3: Conditional routing to left or right pain model
- Step 4: Displaying the PSPI pain score visually

1.1 User Journey and System Workflow Explanation

The Stroke Pain Detection App is designed to offer a smooth and user-friendly experience for clinicians, researchers, or any user assessing pain in stroke patients. The following steps describe the full working process from the moment the user lands on the application to the final prediction output:

1.1.1 Landing Page – Model Initialization

When the application is launched, the user is greeted by a simple and informative landing page titled “Stroke Patient Pain Intensity Detector.” This page briefly explains that the app uses a full-face image of a stroke patient to determine the affected side of the face and uses the unaffected side to predict pain intensity.

At this stage:

- The system automatically begins downloading and initializing two key models:
  - A stroke detection model for identifying the affected side.
  - A pain detection model for predicting the pain intensity.
- Visual logs and progress icons (such as checkmarks) confirm each model's status (e.g., “Models loaded”).

This creates transparency and builds user confidence before interaction begins.

1.1.2 Image Upload by the User

Once the models are loaded, the app prompts the user to upload a full-face image of the patient.

- Supported formats: JPG, JPEG, PNG.
- After upload, the image is resized to the model’s expected input size (e.g., 224x224).
- Processing begins immediately with system messages like:
  - “Running stroke model prediction…”
  - “Predicting pain score…”

This provides real-time feedback and helps the user understand what the system is doing behind the scenes.

1.1.3 Prediction Output and Result Display

After processing is complete, the app returns a detailed and visually clear output:

- Affected side: The side of the face determined to be impaired by stroke.
- Unaffected side: The side used to predict pain, based on symmetry analysis.
- Cropped Image Display: A visual of the selected unaffected side, confirming the region used for prediction.
- PSPI Score: The predicted pain intensity level based on the Prkachin and Solomon Pain Intensity (PSPI) metric.

This layout allows users—especially those in a medical context—to clearly understand both the decision-making logic and the final pain estimation.

1.1.4 Why This Design Matters

This design is simple, scalable, and highly interpretable. It ensures:

- Real-time feedback at every step of the pipeline.
- Visual validation of the model's logic (affected vs. unaffected side).
- Clinically relevant output (PSPI score) with a clean layout for decision-making.

Whether used in clinical research or development environments, the interface is intuitive and practical.

1.2 Figma User Interface Prototype (see 1.3 streamlit design, is better working than figma)

To support the graphical design of the Stroke Pain Detection App, a full user interface prototype was created using Figma, a collaborative interface design tool. This prototype mirrors the layout and behavior of the deployed Streamlit application.

The Figma design serves multiple purposes:

- It allows stakeholders to preview the interaction flow before implementation.
- It simulates the model loading, file upload, and prediction results, giving a realistic sense of usability.
- It supports iterative improvements in UI/UX by allowing visual feedback during the design phase.

The prototype exactly follows the functional structure implemented in the Streamlit version. From model initialization to result visualization, each interaction step is clearly represented.
https://www.figma.com/community/file/1500165037324368712

The layout of the Figma interface follows the actual Streamlit workflow, starting from:

- Initial model loading feedback.
- Image upload area with restrictions clearly visible.
- Processing states shown dynamically.
- Final prediction results with clearly labelled
  - Affected side
  - Unaffected side used
  - Predicted PSPI pain score

This ensures that the prototype remains faithful to the functionality and logic of the final product.

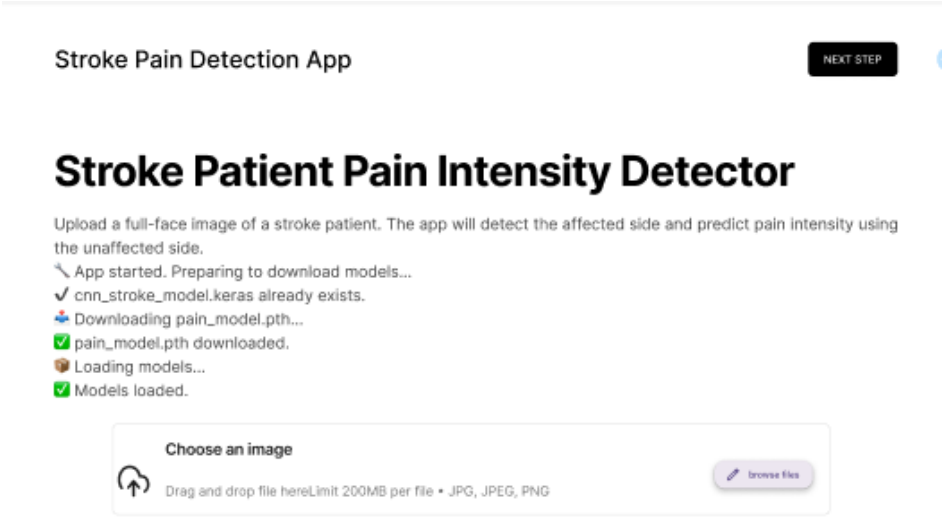

Figure 1.1: Home interface showing model loading and image upload options

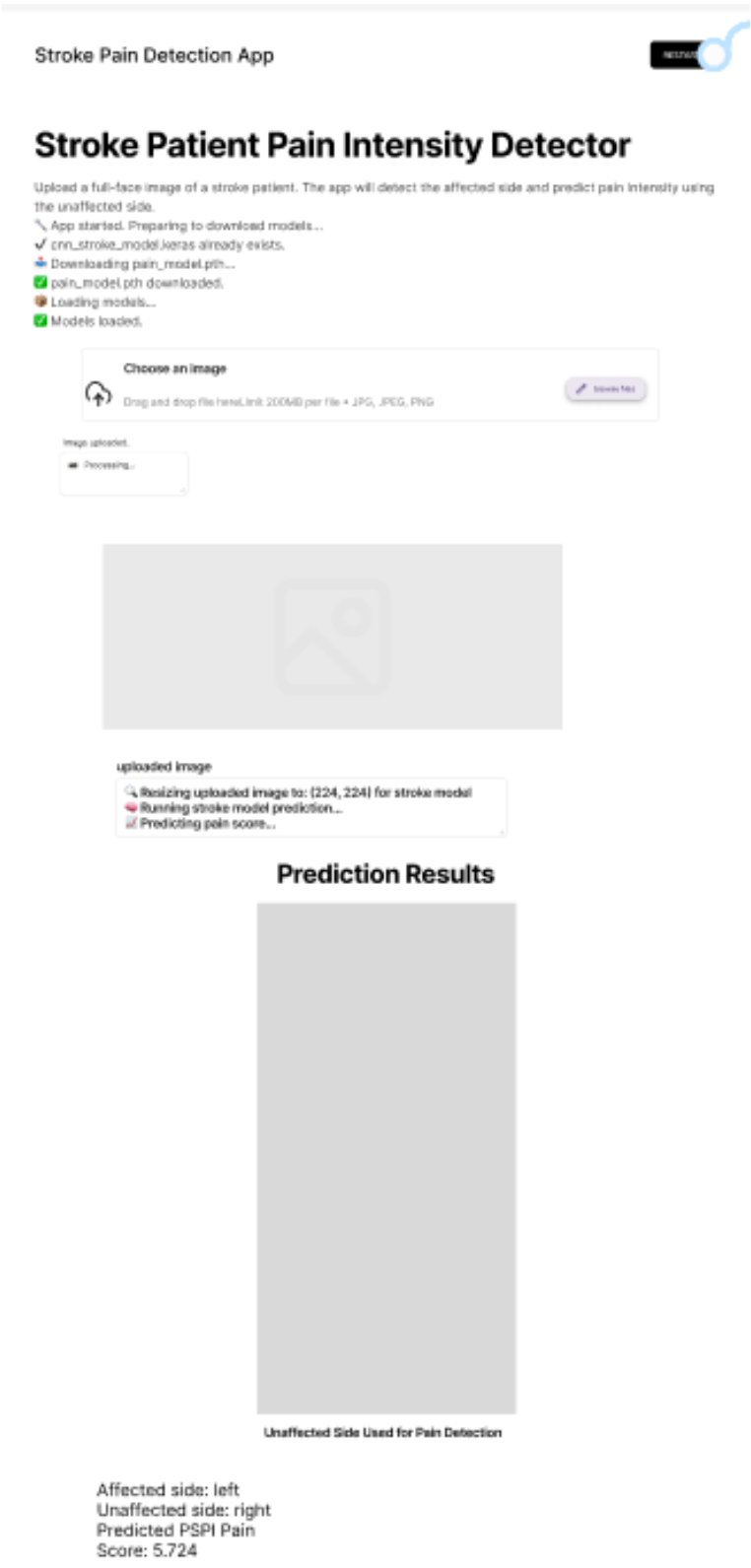

Figure 1.2: Prediction interface showing the cropped face and PSPI output

1.3 Streamlit Application Deployment & Testing (Better working design)

To demonstrate and test the full functionality of the pain detection pipeline, a Streamlit web application was developed and deployed on Hugging Face Spaces. This app serves as a working proof-of-concept that takes a full-face image of a stroke patient, detects the stroke-affected side, and predicts pain intensity from the unaffected side using a deep learning model.

This application provides a simple, intuitive interface for users to upload images, and it delivers real-time feedback on pain detection results. It is publicly available and can be accessed at: https://huggingface.co/spaces/AdhamQQ/StrokePatientsModel

The deployment uses dynamic model downloading from Hugging Face, making it scalable and lightweight for production use.

1.3.1 Application Interface Overview

Below is a screenshot of the Streamlit app during startup and image upload:

Figure 9: Streamlit interface showing model loading and image input.

Users can upload JPG or PNG facial images. The app automatically loads the pretrained models. It provides logs on each step, from image preprocessing to stroke side detection.

1.3.2 Prediction Output

Once the image is processed, the system outputs:

- The detected affected side (left or right)
- The unaffected side used for pain prediction
- A cropped view of the unaffected side
- The predicted PSPI pain score

Figure 10: Prediction result showing detected side and PSPI score.

This output allows users and clinicians to visually verify which half of the face was used and to interpret pain intensity based on the model's score.

2. Technical Design

This section explains the technical backbone of the solution. The system integrates multiple deep learning models, each trained separately and hosted remotely. The stroke model is a CNN built using TensorFlow and is trained on over 1250 stroke patient images. It analyzes full-face inputs to determine the stroke-affected side. The pain detection models (right and left) are based on ResNet18 architecture, implemented using PyTorch. These were trained using half-face datasets annotated with PSPI pain intensity scores. Additionally, Dlib is used for facial landmark detection to ensure accurate splitting of faces, and OpenCV is used for image transformation. The full stack is packaged with Streamlit for web interface, and Hugging Face Spaces is used for deployment and model hosting.

2.1 Model Technical Breakdown:

- ❖ Stroke Model: A CNN built in TensorFlow, trained on 1259 labeled stroke images. Uses full-face input and predicts side classification (left/right).
- ❖ Pain Models: Two ResNet18-based regressors implemented in PyTorch, trained on right and left cropped halves respectively using PSPI-labeled datasets of healthy individuals.

The architecture integrates three deep learning models into a unified pipeline:
- Stroke Model (CNN) trained on real patient images to determine the affected side
- Left and Right Pain Detection Models trained on facial expressions labeled with PSPI scores

2.2 Key Libraries And Tools

- Python 3.10
- TensorFlow for the stroke model, PyTorch for pain models
- Dlib for 68-point facial landmark detection
- OpenCV for image manipulation
- Hugging Face Hub for model storage and versioning
- Streamlit for the frontend interface

2.3 Scalability

The architecture of the Stroke Pain Detection pipeline was intentionally designed with scalability in mind. Each component of the system—stroke side detection, facial image processing, and pain prediction—operates in a modular and stateless manner. This makes it easy to scale the system horizontally (e.g., run multiple instances in parallel) to handle high-volume clinical use cases.
By using pre-trained models hosted externally (via Hugging Face), and deploying the front-end with Streamlit on a cloud platform, the application can be easily integrated into larger health tech systems, expanded to support more users, or adapted to run on edge devices (e.g., hospital tablets or portable diagnostic units).
Additionally, the model inference process is efficient and lightweight, enabling real-time predictions even on modest hardware. Future enhancements, such as API endpoints or mobile integration, can also be added without restructuring the current pipeline.

3. Functional Design

This section outlines how each part of the system works together to process a user-submitted image and return a pain prediction. Once an image is uploaded, the stroke model determines which side is affected. Then, facial landmarks are used to crop and extract the unaffected side. Depending on which side was detected, either the left or right pain model is invoked. The PSPI score is predicted and displayed to the user with a labeled scale. The system also supports re-uploads and restart workflows, supporting batch processing and edge deployment for future upgrades. This modular and condition-based logic makes the

pipeline scalable, testable, and extendable to include additional biometric feedback in the future.

Model Functional Logic:

- Upon upload, the system runs stroke side classification first.
- Depending on the result, the image is split using dlib's facial landmarks, and the unaffected half is routed to the corresponding PSPI regression model.
- Both left and right pain models use identical architecture but differ in training data and learned features.

The application is functionally structured to reflect a modular pipeline that follows clinical workflow:
1. Upload full-face stroke patient image
2. Use stroke model to detect affected side
3. Crop the unaffected side using facial landmarks
4. Feed to the correct pain model (right/left)
5. Predict and display the PSPI score to the user

To support this flow, the following UML and design diagrams are used:

3.1 Pipeline flow diagram

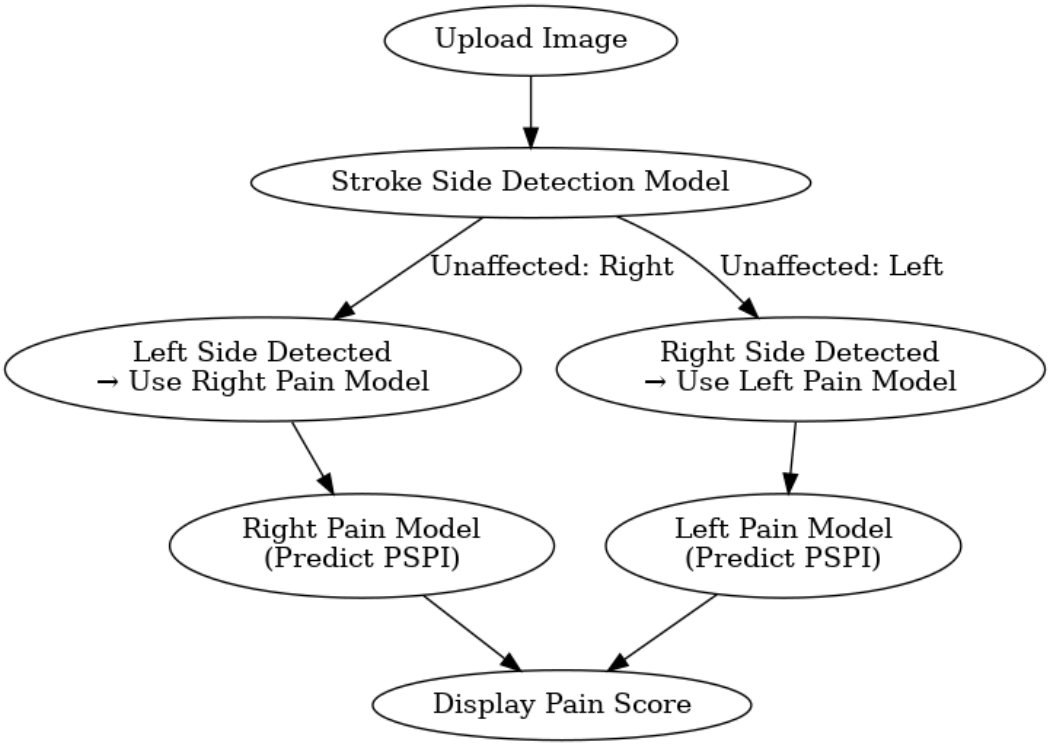

Figure 2: Pipeline flow diagram showing the sequential steps from input to pain prediction.

3.2 Use Case Diagram

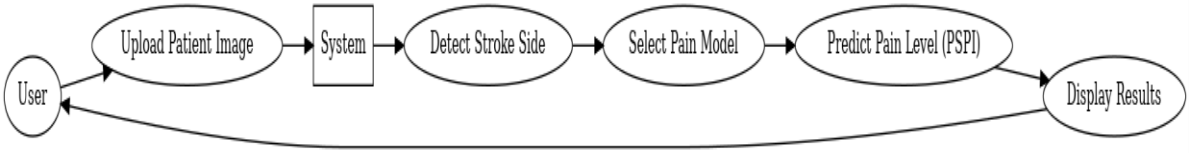

Figure 3: Use case diagram demonstrating user interaction with the system.

3.3 Class Diagram

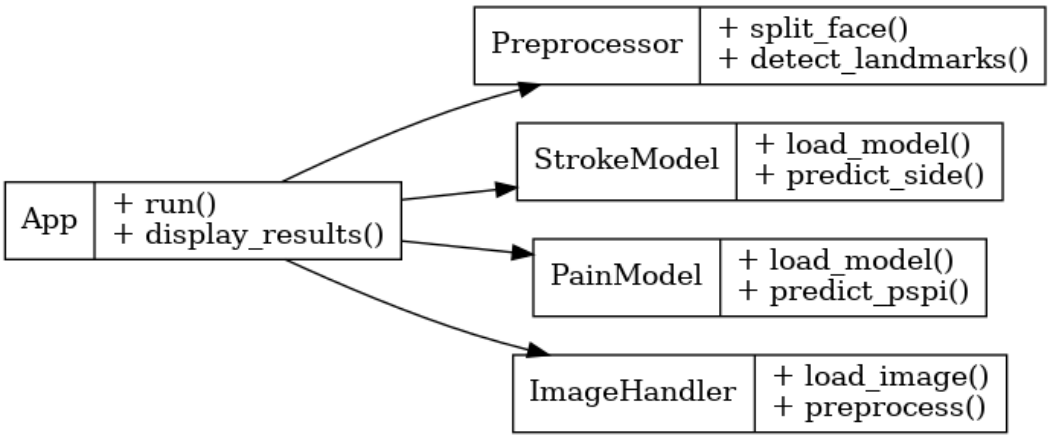

Figure 4: Class diagram outlining modular structure and responsibilities in the codebase.

3.4 Sequence Diagram

The sequence diagram represents the time-ordered flow of interaction between the user, the frontend UI, and backend models. It helps clarify the call-and-response mechanism from image upload to model prediction and result display.

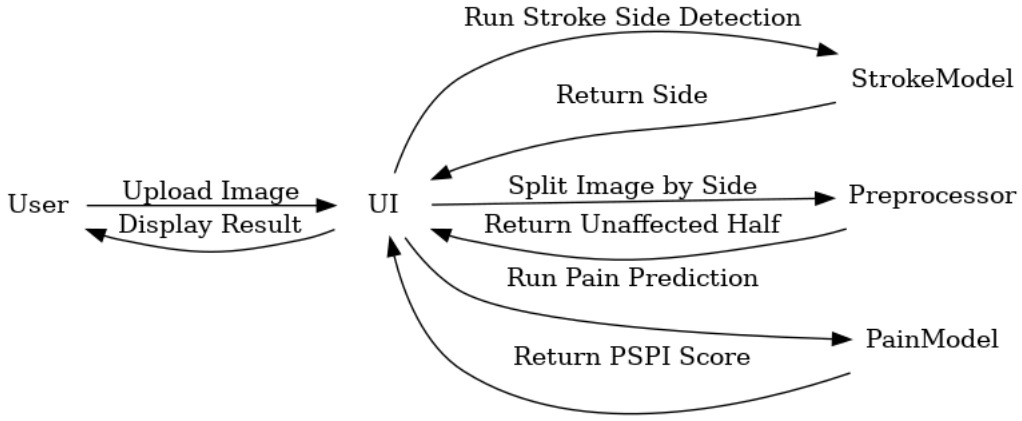

Figure 5: Sequence diagram showing component-level interaction flow.

3.5 State diagram

The state diagram illustrates the various operational states the system transitions through. Starting from idle, the system processes uploads, performs detection, prediction, and returns to idle after displaying results. This is essential for modeling system behavior in response to events.

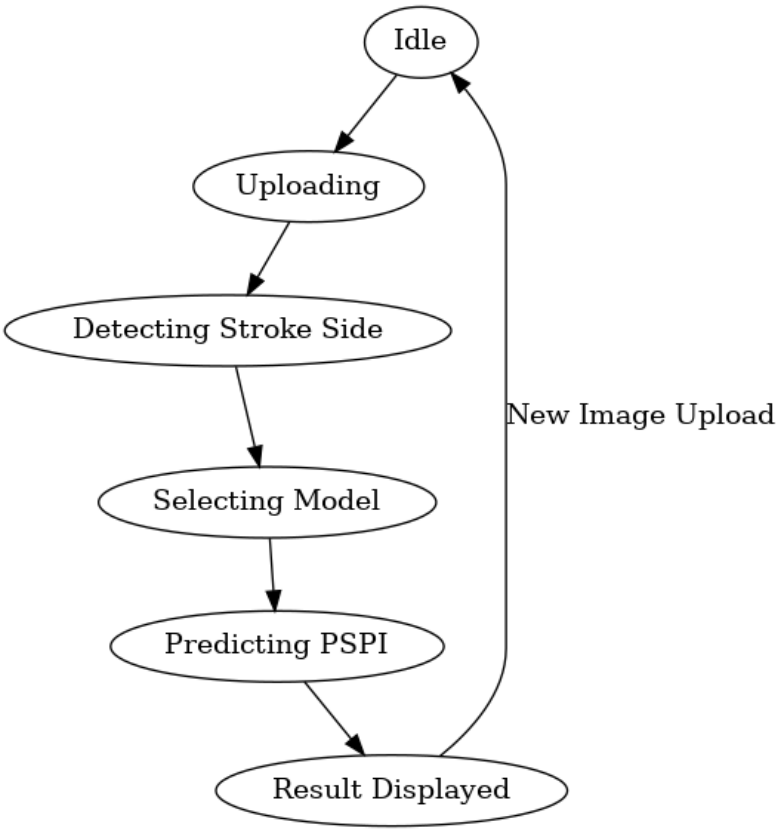

Figure 6: State diagram showing lifecycle states of the app.

3.6 Process Diagram

This process diagram details the end-to-end flow of data and operations. Each block represents a core system step and is useful for understanding the implementation at a workflow level. This includes detection, routing, and scoring.

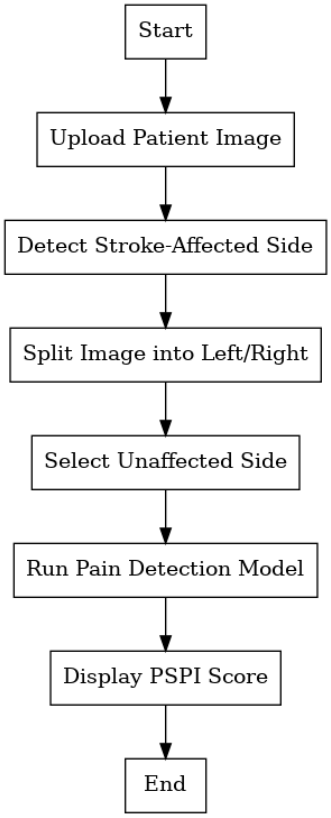

Figure 7: Process diagram outlining system logic from upload to output.

3.7 Activity Diagram

The activity diagram emphasizes both user and system activities. It represents the actions taken in sequence and helps define responsibilities of different system components during pain prediction operations.

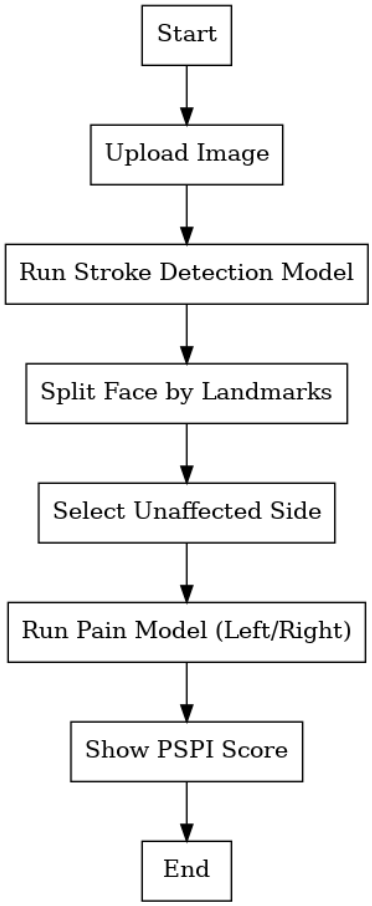

Figure 8: Activity diagram showing user/system task progression.