

Project Report

HTP Drawing Analysis Platform

Team - Social Sigmas

November 17, 2025

Contents

1	Problem Statement	1
1.1	Limitations of the Current Workflow	1
1.2	Addressing These Challenges	1
2	Architecture (Technical and ML Pipelines)	2
2.1	Technical Pipeline	2
2.1.1	Frontend Layer	2
2.1.2	API Gateway	3
2.1.3	Backend Services	3
2.1.4	Data Layer	3
2.1.5	Overall Data Flow	4
2.1.6	Summary	4
2.2	System Architecture Overview	5
2.3	AI-ML Pipeline Workflow	6
2.3.1	Phase 1: Parallel Analysis	6
2.3.2	Phase 2: Sequential Processing	6
2.4	Workflow State Machine	8
2.4.1	State Transition Diagram	8
2.4.2	State Definitions	9
2.5	Agent Architecture	9
2.5.1	7 Specialized Agents	9
2.6	LLM Models and Configuration	10
2.6.1	Gemini 2.5 Flash Configuration	10
2.6.2	Model Capabilities	10
2.6.3	Token Usage Tracking	10
2.6.4	Caching Strategy	10
2.7	Data Flow and Processing	10
2.7.1	Input Handling	10
2.7.2	Parallel Stage Processing	11
2.7.3	Sequential Processing Pipeline	11
2.7.4	Output Structure	11
2.8	Implementation Details	12
2.8.1	Core Classes	12
2.8.2	LangChain Chains	13
2.8.3	Concurrency Model	13
2.9	PLUTO Workflow	13
2.9.1	Simplified Person-Only Analysis	13
2.10	Error Handling and Safety	14
2.10.1	Input Validation	14
2.10.2	Safety Signal (FIX_SIGNAL)	14

2.10.3	Token Usage Monitoring	14
2.10.4	Caching Benefits	15
2.10.5	Exception Handling	15
2.11	Technology Stack Summary	15
2.12	Prompt Template Locations	15
2.13	Key Insights	15
2.13.1	Architecture Type	15
2.13.2	Workflow Efficiency	15
2.13.3	Safety & Quality	16
2.13.4	Scalability	16
3	Entity Relationship Diagram (ERD)	16
3.1	Entities and Attributes	17
3.1.1	users	17
3.1.2	drawings	17
3.1.3	ai_analysis	17
3.1.4	assessments	18
3.2	ERD Summary	18
4	User Flow Implementation	18
4.1	Registration and Login	18
4.2	Student Portal	19
4.3	Facilitator Portal	21
4.4	Psychologist Portal	23
4.5	Assessment Status Lifecycle Summary	24
5	APIs (Signatures, Status Codes, Responses, Requests)	25
5.1	Authentication Endpoints	25
5.1.1	POST /api/register	25
5.1.2	POST /api/token	25
5.1.3	GET /api/users/me	25
5.2	Student Endpoints	26
5.2.1	POST /api/drawings/upload	26
5.2.2	GET /api/my-submissions	26
5.3	Facilitator Endpoints	26
5.3.1	GET /api/assessments/facilitator	26
5.3.2	GET /api/psychologists	26
5.3.3	PUT /api/drawings/drawing id/assign/psychologist id	26
5.4	Psychologist Endpoints	27
5.4.1	GET /api/assessments/psychologist	27
5.4.2	POST /api/drawings/drawing id/evaluate	27
5.5	Common Status Codes and Notes	27
6	Folder Structure	27
7	Setup Instructions	28
7.1	Guide to Running the HTP Platform on a New Linux Machine	28
7.1.1	Step 1: Install Prerequisites	28
7.1.2	Step 2: Prepare the Project Files	29
7.1.3	Step 3: Build and Run the Application	29
7.1.4	Step 4: Access the Application	30
7.1.5	Stopping the Application	30
7.2	Guide to Running the HTP Platform on a New Windows Machine	30
7.2.1	Step 1: Install Prerequisites	30
7.2.2	Step 2: Prepare the Project Files	31

7.2.3	Step 3: Build and Run the Application	31
7.2.4	Step 4: Access the Application	32
7.2.5	Stopping the Application	32

8	Individual Contribution:	32
----------	---------------------------------	-----------

1 Problem Statement

The traditional House–Tree–Person (HTP) psychological assessment is a widely used projective technique for understanding emotional, cognitive, and behavioral functioning. In the context of this project, the primary emphasis is placed on the **Person drawing**, as it is often the most psychologically revealing component of the HTP test. However, the manual process of administering the assessment, collecting person drawings, interpreting symbolic features, and generating structured reports remains slow, subjective, and difficult to scale especially in environments involving large student groups or distributed mental-health programs.

1.1 Limitations of the Current Workflow

- **Manual collection of person drawings** leads to logistical delays and increases the likelihood of missing, incomplete, or damaged submissions.
- **Subjective interpretation of human-figure features** such as posture, proportions, clothing, facial detail, and limb emphasis varies significantly across psychologists, resulting in inconsistent evaluations.
- **High interpretive workload** makes it challenging for mental-health professionals to analyze a large number of person drawings, especially when each drawing requires detailed symbolic and feature-level inspection.
- **Lack of digitization** limits the ability to store drawings securely, extract fine-grained visual features, track longitudinal changes, or integrate analytical tools for psychological insight.
- **Disconnected coordination** between students, facilitators, and psychologists causes delays and reduces efficiency in the overall assessment and reporting cycle.

1.2 Addressing These Challenges

The **Person-Focused HTP Analysis Platform** offers a modern, digital, and scalable solution tailored specifically for automated analysis of the Person drawing. By integrating structured automation, intelligent role-based workflows, and AI-assisted interpretation, the platform ensures faster, more consistent, and more objective psychological evaluations. Specifically, the system enables:

- **Students** to create or upload their Person drawings through an intuitive, guided digital interface.
- **Facilitators** to set up assessment sessions, manage student participation, and track drawing submissions in real time.
- **Psychologists** to access AI-generated insights for person-drawing features such as limb structure, shading intensity, figure proportions, emotional indicators, and symbolic attributes supported by annotated overlays and narrative summaries for deeper and more consistent evaluation.

The platform integrates a microservices backend, a role-based portal system, an automated Person-drawing analysis pipeline, and a secure database architecture to ensure fast, reliable, and high-quality assessment workflows. Using Gemini-based person-feature segmentation with psychological feature interpretation, and AI-driven summarization, the system significantly reduces manual effort while retaining essential professional oversight.

Overall, the platform aims to digitize and modernize the Person-drawing component of the HTP assessment, enhance consistency in psychological interpretation, improve scalability for large institutions, and support psychologists with AI-assisted insights while upholding ethical and professional standards in mental-health evaluation.

2 Architecture (Technical and ML Pipelines)

2.1 Technical Pipeline

The system follows a microservices-based architecture to ensure scalability, modularity, and maintainability. A detailed explanation of each layer and service is provided below. The overall system is divided into several logical layers, as illustrated in Figure 1.

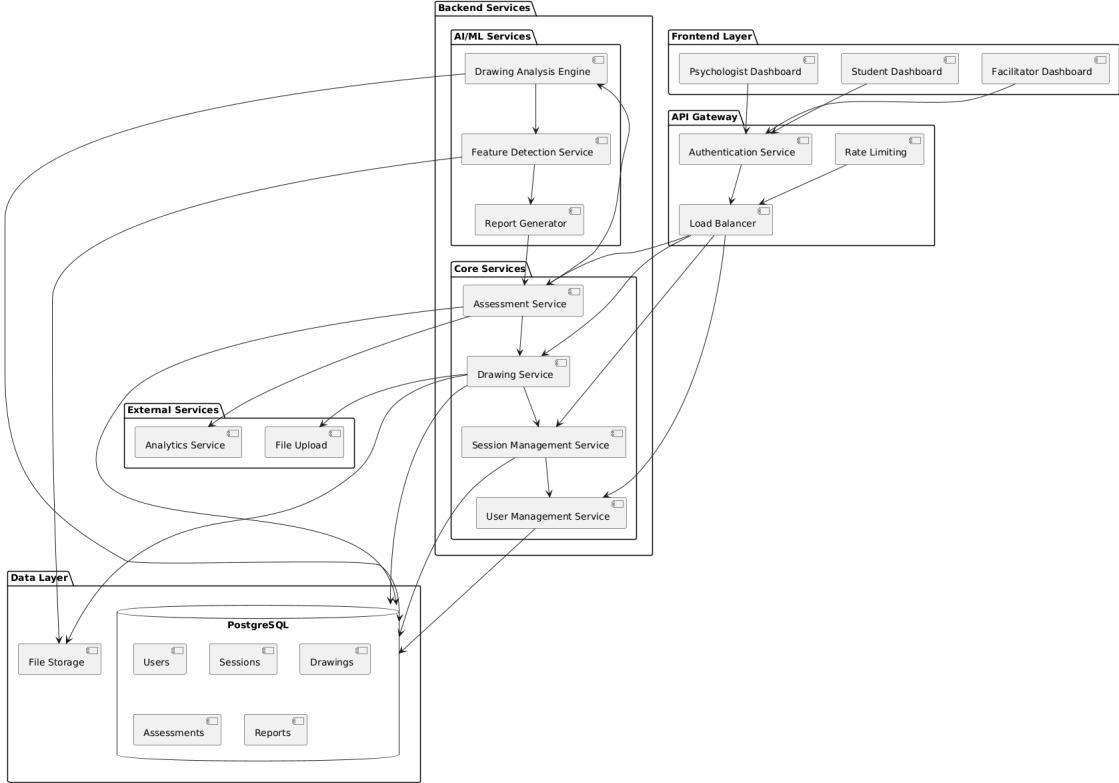


Figure 1: Overall System Architecture Diagram

2.1.1 Frontend Layer

The **Frontend Layer** consists of user-facing dashboards tailored to specific roles:

- **Psychologist Dashboard:** Allows psychologists to review drawings, inspect AI-generated features, validate interpretations, and finalize reports.
- **Student Dashboard:** Enables students to log in, complete the House–Tree–Person (HTP) drawing task, upload drawings, and track session progress.
- **Facilitator Dashboard:** Used by facilitators to manage student accounts, monitor assessment completion, and upload scanned drawings when required.

All dashboards interact exclusively with the API Gateway for security and standardization.

2.1.2 API Gateway

The **API Gateway** acts as the single entry point for all incoming client requests. It consists of:

- **Authentication Service:** Provides login, session validation, role-based access control, and JWT token generation.
- **Rate Limiting:** Prevents misuse or accidental flooding of requests by enforcing request throttling.
- **Load Balancer:** Distributes incoming traffic across multiple backend service instances to ensure high availability and scalability.

Once validated, requests are routed to the appropriate backend microservices.

2.1.3 Backend Services

The backend is composed of two major groups of microservices: **Core Services** and **AI/ML Services**. This separation ensures clean modularity and efficient scaling of computation-intensive components.

Core Services:

- **Assessment Service:** Orchestrates the full assessment workflow, coordinating drawing submissions, AI analysis, report generation, and review processes.
- **Drawing Service:** Manages drawing uploads, preprocessing, file storage, and the communication between file storage and AI services.
- **Session Management Service:** Tracks the progress of every student session, ensuring completion of the HTP sequence and managing lifecycle events.
- **User Management Service:** Handles user account creation, authentication details, permissions, and role assignments.

AI/ML Services: These computational services process drawings and generate structured analysis outputs:

- **Drawing Analysis Engine:** The main model responsible for analyzing student drawings using Gemini. It extracts psychological indicators, spatial layouts, and stroke-related features.
- **Feature Detection Service:** Performs low-level and mid-level feature extraction, including object detection, line smoothness estimation, geometric patterns, and structural markers.
- **Report Generator:** Combines psychologist-approved interpretation rules with AI predictions to generate complete textual reports. These reports include:
 - Summaries of extracted features,
 - AI-generated interpretations,
 - Auto-summarized insights using AI summarization.

2.1.4 Data Layer

The **Data Layer** consists of both structured and unstructured data storage systems, designed for robustness and scalability.

PostgreSQL (Structured Storage) Stores all essential relational data including:

- User records (students, facilitators, psychologists),
- Assessment sessions,
- Drawing metadata, including student and assigned psychologist relationships, timestamps, and status (e.g., submitted, in_review, reviewed).
- Extracted features,
- Final reports and interpretations.

File Storage (Unstructured Data) A local file system volume within the Docker environment is used for storing raw image files. This includes:

- Raw drawing images,
- This system is designed to be easily swappable with a cloud-based object storage service like AWS S3 for production-level scalability.

2.1.5 Overall Data Flow

The end-to-end pipeline proceeds as follows:

1. The user logs into their respective dashboard. The API Gateway authenticates the request and issues a secure JWT token for session management.
2. The student uploads a drawing via the Drawing Service. The backend API saves the image file to the designated file storage and creates a new drawing record in PostgreSQL with a status of submitted. No AI analysis is triggered at this stage.
3. A facilitator views the dashboard of all submitted drawings and assigns a case to an available psychologist.
4. This assignment action triggers the AI/ML pipeline as a background task. The drawing's status is updated to processing, and the pluto_workflow is executed.
5. Upon completion, the entire AI-generated JSON report is saved to the ai_analysis table in PostgreSQL, and the drawing's status is updated to in_review.
6. The assigned psychologist logs in and sees the case in their queue. They review the drawing, the AI-generated report, and add their own professional notes. Upon saving, the notes are stored and the drawing's status is updated to reviewed.
7. The student can then view their submission history and see the final notes from the psychologist for any reviewed assessments.

2.1.6 Summary

This modular architecture ensures:

- **Scalability** by separating the frontend, backend, and database into independent, containerized services.,
- **Secure access** via centralized, token-based authentication system with role-based permissions.
- **Efficient, on-demand AI analysis** triggered by facilitator action, conserving computational resources

- **Robust data management** using PostgreSQL for structured data and a flexible file system for images.
- **Maintainability** through a clear and decoupled architecture that separates the frontend user interface from the backend business logic.

2.2 System Architecture Overview

The system's architecture is designed for modularity, precision, and performance, centered around a sophisticated multi-agent workflow. It utilizes a unified Large Language Model (LLM)—specifically, Google's Gemini 2.5 Flash model—which capably handles both visual perception for feature extraction and structured text generation for psychological analysis. The architecture is built upon a series of specialized agents, where each agent is a distinct prompt-driven task assigned a unique responsibility within the analysis pipeline. This modular design is orchestrated by a central HTPModel class. To ensure user well-being, the full workflow incorporates safety mechanisms that automatically detect potential warning signs in the analysis and trigger appropriate signals for professional review. A key feature of the platform is its support for multiple workflow configurations:

- **Full Comprehensive Workflow:** The original architecture designed to analyze the House, Tree, Person, and Overall composition in parallel, followed by sequential stages for merging, summarizing, and risk classification.
- **Streamlined PLUTO Workflow:** A customized and optimized workflow developed for this project, which performs a targeted, serial analysis on only the "Person" drawing, delivering a concise and structured report tailored for the psychologist's review portal.

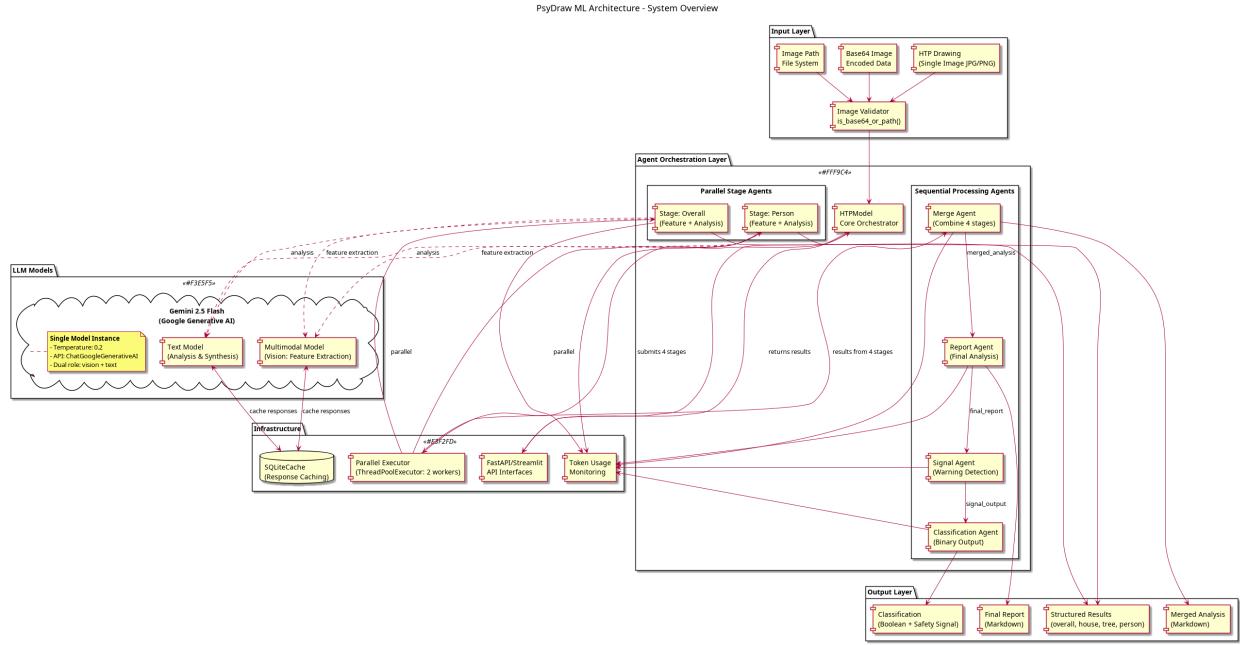


Figure 2: System Architecture - Component Overview showing all major layers and components

The system is constructed across four primary layers. The Input Layer is designed to accept HTP drawings in multiple formats, accommodating both local file paths and Base64 encoded strings for flexible integration. These inputs are processed by the LLM Models Layer, which leverages Google Generative AI's Gemini model to provide the necessary vision and text capabilities. The Agent Orchestration Layer is the

core of the system, consisting of specialized agents coordinated by the central HTPModel class to execute the multi-stage analysis. Supporting these components is the Infrastructure Layer, which handles essential background services such as database caching (SQLite), application logging for monitoring, and the FastAPI interface that exposes the entire workflow as a secure REST API.

2.3 AI-ML Pipeline Workflow

Complete Workflow Execution:- The complete workflow consists of two main phases.

2.3.1 Phase 1: Parallel Analysis

The system initiates processing by executing two stages in parallel using a `ThreadPoolExecutor` configured with two concurrent workers.

- **Stage 1: Overall Analysis**

Feature Extraction: The vision model analyzes the overall composition of the drawing using the `overall_feature.txt` prompt. It extracts compositional elements, spatial relationships, and the general drawing style, outputting the data as Markdown-formatted feature descriptions.

Psychological Analysis: Following extraction, the text model interprets the features using the `overall_analysis.txt` prompt. This step provides the necessary psychological context and interpretations, resulting in a Markdown-formatted analysis.

- **Stage 2: Person Analysis**

Feature Extraction: Concurrently, the vision model analyzes the specific details of the person figure using the `person_feature.txt` prompt. It extracts figure characteristics, body parts, proportions, and other fine details, outputting them as Markdown-formatted feature descriptions.

Psychological Analysis: The text model then interprets these person-specific features using the `person_analysis.txt` prompt. This provides a psychological interpretation specific to the person drawing, outputting a Markdown-formatted analysis.

- **Parallel Processing Benefits**

$$\text{Total Time} = \max(T_{\text{overall}}, T_{\text{person}}) + T_{\text{sequential}} \ll T_{\text{overall}} + T_{\text{person}} + T_{\text{sequential}} \quad (1)$$

Executing stages in parallel reduces total pipeline execution time significantly compared to sequential processing.

2.3.2 Phase 2: Sequential Processing

After parallel completion, results flow through sequential synthesis stages:-

- **Merge Analysis Agent**

This agent accepts the feature and analysis results from both parallel stages as input. Using the `analysis_merge.txt` prompt, it combines the overall and person analyses into a coherent narrative. The output is an integrated markdown report formatted according to the `merge_format.txt` template.

- **Report Generation Agent**

The Report Generation Agent takes the merged analysis as input. Using the `final_result.txt` prompt, it generates a professional mental health assessment. The output is a formatted final report in markdown, which includes psychological interpretations and specific recommendations.

- **Signal Detection Agent**

This agent analyzes the final report using the `signal_judge.txt` prompt to identify mental health warning indicators. It outputs a signal summary detailing any detected warnings, which is used to trigger safety mechanisms if issues are found.

• Classification Agent

The Classification Agent processes the signal analysis output using the `clf.txt` prompt to perform a binary classification (Warning vs. Normal). It utilizes a `JsonOutputParser` to structure the result. The logic dictates that a `False` result indicates a warning, triggering the `FIX_SIGNAL` mechanism, while a `True` result indicates a normal state with no safety signal required.

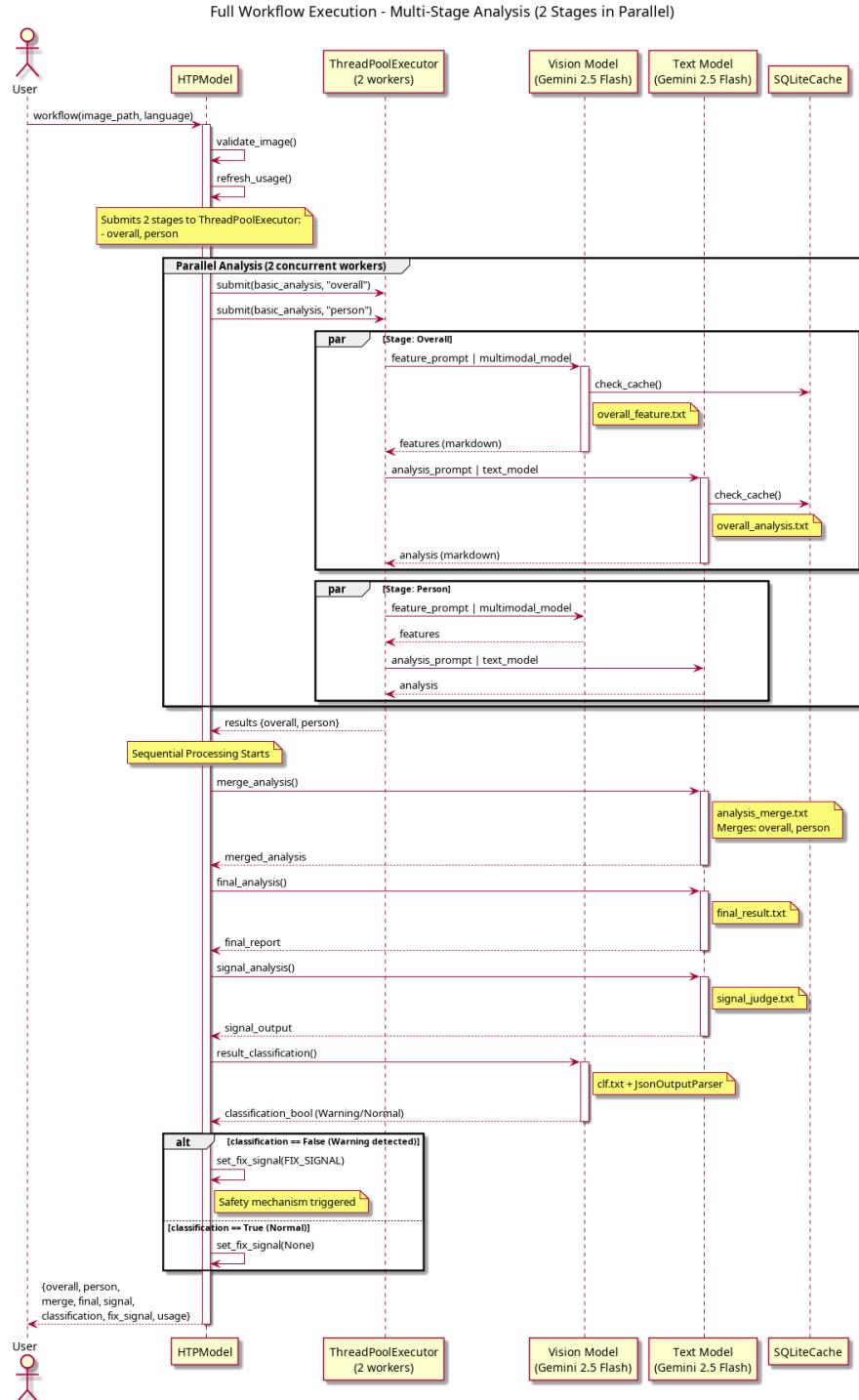


Figure 3: Sequence Diagram - Full Workflow Execution with 2 Parallel Stages and Sequential Processing

2.4 Workflow State Machine

2.4.1 State Transition Diagram

The workflow follows a well-defined state machine:-

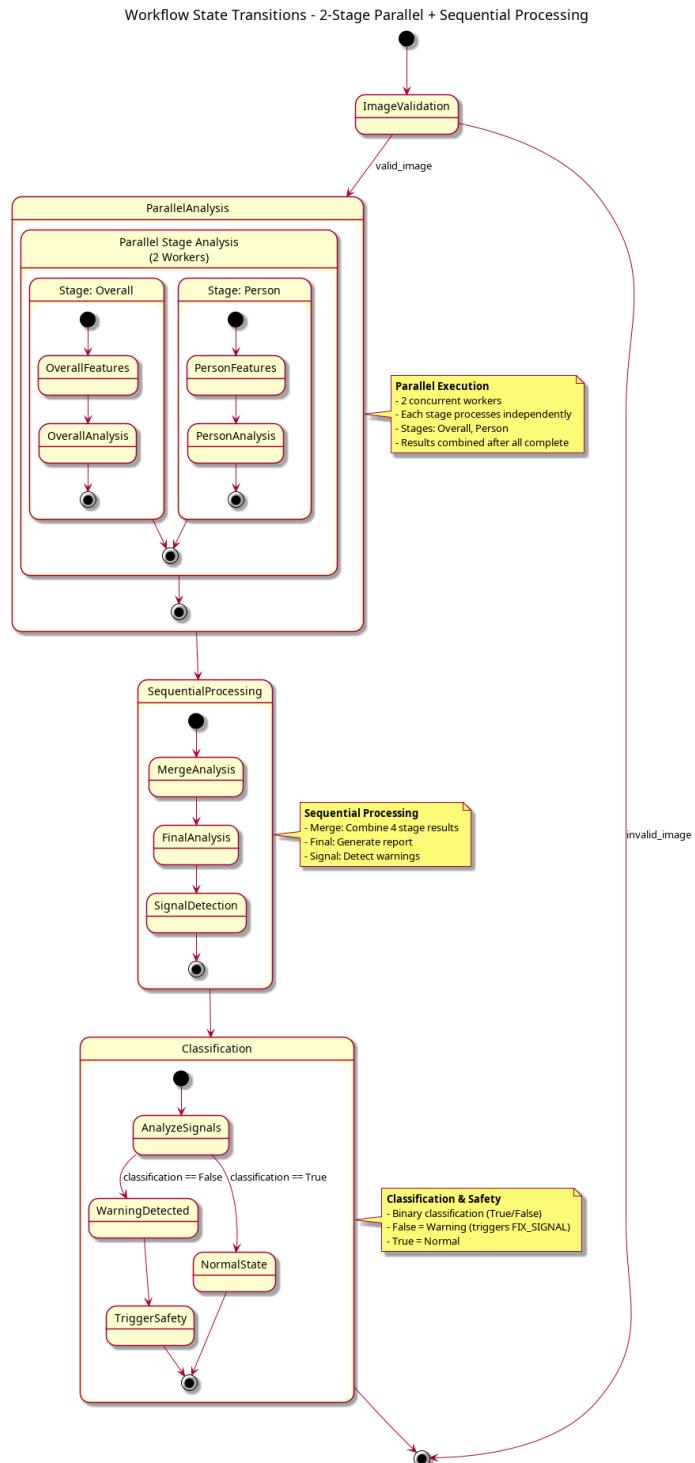


Figure 4: State Diagram - Workflow transitions from validation through classification

2.4.2 State Definitions

- **Vision Model Agents**

1. **Feature Extraction Agent (Vision)**:- Analyzes visual features from drawings using the Gemini 2.5 Flash model.

2. **Classification Agent (Vision)**:- Binary classification using the `clf.txt` prompt, outputs JSON boolean.

- **Image Validation State**

The process begins by validating the input image format and encoding. It checks for a valid file path or Base64 string, encoding the image to Base64 if necessary. On success, it branches to the parallel analysis state; otherwise, it returns anState error.

- **Parallel Analysis State (2 Workers)**

Stage: Overall This stage handles feature extraction via the vision model and psychological analysis via the text model. It runs concurrently with the Person stage.

Stage: Person This stage handles feature extraction via the vision model and psychological analysis via the text model. It runs concurrently with the Overall stage.

- **Sequential Processing State**

The workflow proceeds linearly through merging analyses, generating the final report, and detecting signals before looping back to finalize the process.

- **Classification & Safety State**

The system analyzes the classification result. If the result is a Warning (False), it triggers the `FIX_SIGNAL` safety mechanism. If the result is Normal (True), it sets the signal to None.

2.5 Agent Architecture

2.5.1 7 Specialized Agents

- **Vision Model Agents (2)**

1. **Feature Extraction Agent (Vision)**: This agent analyzes visual features from drawings using the multimodal Gemini 2.5 Flash model. It processes both the overall and person stages, outputting structured markdown descriptions.

2. **Classification Agent (Vision)**: This agent performs binary classification using the multimodal Gemini 2.5 Flash model. Utilizing the `clf.txt` prompt, it outputs a JSON structured boolean result.

- **Text Model Agents (4)**

3. **Analysis Interpretation Agent (Text)**: This agent interprets features psychologically using the text-based Gemini 2.5 Flash model. It processes the analysis for both overall and person stages, outputting psychological interpretations.

4. **Synthesis/Merge Agent (Text)**: This agent combines multiple analyses using the `analysis_merge.txt` prompt. It merges the overall and person data into a single integrated report.

5. **Report Generation Agent (Text)**: This agent creates professional reports using the `final_result.txt` prompt, outputting the final assessment document.

6. Signal Detection Agent (Text): This agent identifies warning indicators using the `signal_judge.txt` prompt, outputting a signal summary.

- **Orchestration Agent (1)**

7. Parallel Execution Agent: This agent utilizes a `ThreadPoolExecutor` with two concurrent threads to coordinate the overall and person stages. It serves as the synchronization point before the sequential phase begins.

2.6 LLM Models and Configuration

2.6.1 Gemini 2.5 Flash Configuration

Model Details The system uses the official `gemini-2.5-flash` model provided by Google Generative AI. It is integrated via `langchain_google_genai.ChatGoogleGenerativeAI` with a temperature setting of 0.2 to ensure deterministic and consistent outputs.

Dual Capabilities

Role	Capability	Use Cases
Text Model	Language generation	Analysis, reports, synthesis
Vision Model	Image understanding	Feature extraction, visual analysis

Both roles are handled by the same model instance for maximum efficiency.

2.6.2 Model Capabilities

The model provides image understanding via a vision transformer, text generation via a language model, and multimodal reasoning combining image and text. It supports structured output (JSON/function calling), long context windows, and strict instruction following.

2.6.3 Token Usage Tracking

Token usage is monitored via `response.usage_metadata`, tracking both `input_tokens` and `output_tokens`. These metrics are stored in the `HTPModel.usage` dictionary and provided in the final output.

2.6.4 Caching Strategy

The system employs an `SQLiteCache` strategy that persists to `cache.db`. This reduces API calls and costs, integrates with the LangChain community cache, and operates transparently to the agents.

2.7 Data Flow and Processing

2.7.1 Input Handling

```
1 Input Image
2   |
3   is_base64_or_path() validator
4   |
5   +-- If path -> encode_image() -> Base64
6   +-- If Base64 -> use directly
7   +-- If invalid -> raise ValueError
8   |
9 Base64 Image Data (ready for LLM)
```

2.7.2 Parallel Stage Processing

```
1 Base64 Image Data
2   |
3 ThreadPoolExecutor (2 workers)
4     +-- Worker 1: Overall Stage
5       |   +-- Vision Model: Extract features
6       |   +-- Text Model: Analyze features
7       |
8     +-- Worker 2: Person Stage
9       +-- Vision Model: Extract features
10      +-- Text Model: Analyze features
11      |
12 Results: {overall: {...}, person: {...}}
```

2.7.3 Sequential Processing Pipeline

```
1 Parallel Results {overall, person}
2   |
3 Merge Agent (analysis_merge.txt)
4     +-- Input: overall analysis + person analysis
5     +-- Output: merged_analysis (markdown)
6   |
7 Report Agent (final_result.txt)
8     +-- Input: merged_analysis
9     +-- Output: final_report (markdown)
10    |
11 Signal Agent (signal_judge.txt)
12    +-- Input: final_report
13    +-- Output: signal_output (text)
14    |
15 Classification Agent (clf.txt)
16    +-- Input: signal_output
17    +-- Parser: JsonOutputParser(ClfResult)
18    +-- Output: classification (boolean)
19    |
20 Safety Decision
21    +-- If False: fix_signal = FIX_SIGNAL
22    +-- If True: fix_signal = None
```

2.7.4 Output Structure

```
1 {
2   overall : { feature : ... , analysis : ... } ,
3   person : { feature : ... , analysis : ... } ,
4   merge : ... ,
5   final : ... ,
6   signal : ... ,
7   classification : True/False ,
8   fix_signal : None or FIX_SIGNAL ,
9   usage : {
10     total : int ,
11     prompt : int ,
12     completion : int
13   }
14 }
```

2.8 Implementation Details

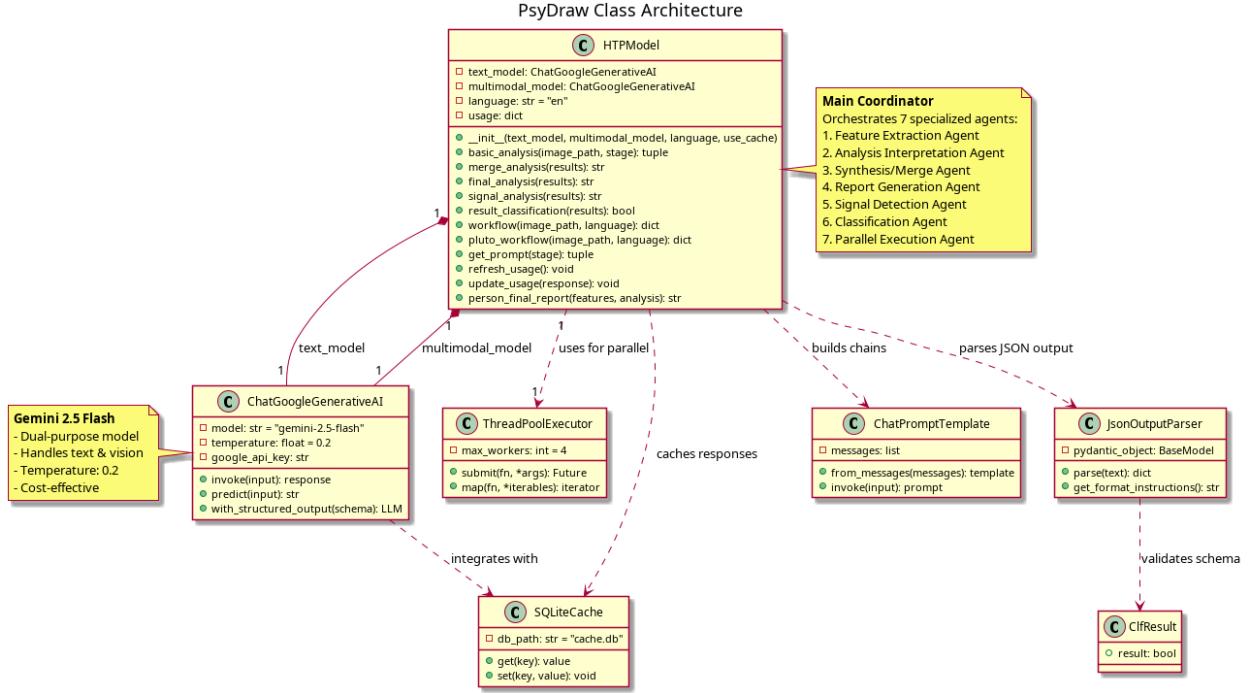


Figure 5: Class Diagram - PLUTO Workflow for Person-Only Analysis

2.8.1 Core Classes

HTPModel (Orchestrator)

Attributes The class maintains the `text_model` and `multimodal_model` as `ChatGoogleGenerativeAI` instances. It tracks the language setting (currently "en") and token usage statistics via a dictionary.

Key Methods The class implements several methods: `workflow` for full execution, `pluto_workflow` for person-only analysis, and `basic_analysis` for individual stages. It also includes specific methods for merging results, generating final reports, analyzing signals, and classifying results.

ChatGoogleGenerativeAI

Integration : LangChain wrapper

Configuration The model is configured using `gemini-2.5-flash` with a temperature of 0.2, utilizing the API Key from environment variables.

Methods It provides `invoke(input)` for LLM inference and `with_structured_output(schema)` for structured operations.

```

1 ChatPromptTemplate.from_messages([
2     ( system , system_prompt ),
3     ( user , [
4         { type : image_url , image_url : ... },
5         { type : text , text : user_input }
6     ])
7 ])

```

```
1 JsonOutputParser(pydantic_object=ClfResult)
```

This parser is responsible for validating and parsing structured JSON responses.

2.8.2 LangChain Chains

Feature Extraction Chain

```
feature_prompt | multimodal_model -> markdown features
```

Analysis Chain

```
analysis_prompt | text_model -> markdown analysis
```

Merge Chain

```
merge_prompt | text_model -> integrated analysis
```

Classification Chain

```
clf_prompt | multimodal_model | JsonOutputParser -> boolean
```

2.8.3 Concurrency Model

The system utilizes a `ThreadPoolExecutor` with a maximum of 2 workers, allocating one per parallel stage. Synchronization is handled by `as_completed()`, which waits for all futures to return. Exception handling is robust, with errors captured in futures and raised upon the `.result()` call.

2.9 PLUTO Workflow

2.9.1 Simplified Person-Only Analysis

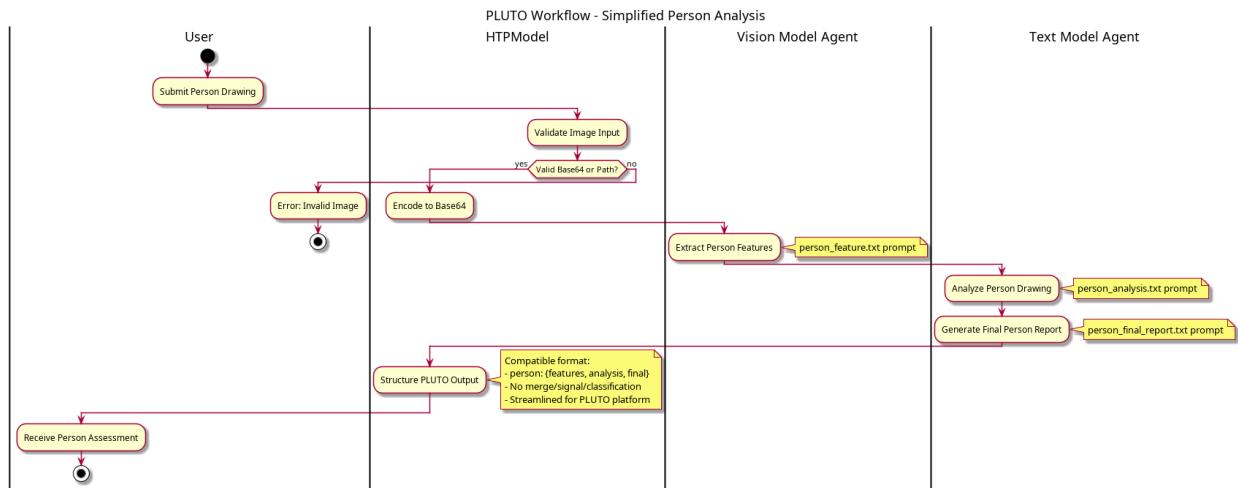


Figure 6: Activity Diagram - PLUTO Workflow for Person-Only Analysis

Purpose: Streamlined workflow for PLUTO platform deployment.

Stages The workflow proceeds through four distinct stages: image validation, person feature extraction, person analysis, and final person report generation. **Output Format** (compatible with full workflow)

```
1  {
2      overall : { feature : Not analyzed. , analysis : Not applicable. } ,
3      house : { feature : Not analyzed. , analysis : Not applicable. } ,
4      tree : { feature : Not analyzed. , analysis : Not applicable. } ,
5      person : { feature : ... , analysis : ... } ,
6      merge : Not applicable for Person-only analysis. ,
7      final : final_report_content ,
8      signal : Please review the final report for a qualitative summary. ,
9      classification : None ,
10     fix_signal : None ,
11     usage : {...}
12 }
```

Benefits This workflow offers faster execution by running a single stage, reduces API calls and token consumption, and follows a simpler reasoning path compared to the full analysis.

2.10 Error Handling and Safety

2.10.1 Input Validation

```
1 if is_base64_or_path(image_path) == path :
2     image_data = encode_image(image_path)
3 elif is_base64_or_path(image_path) == base64 :
4     image_data = image_path
5 else:
6     raise ValueError( Invalid image path or base64 string. )
```

2.10.2 Safety Signal (FIX_SIGNAL)

This signal is triggered when the classification indicates a warning:

IMPORTANT NOTICE

The analysis has detected unusually intense negative emotions in the drawing. This has triggered a safety mechanism.

We strongly recommend seeking immediate assistance from a qualified mental health professional.

2.10.3 Token Usage Monitoring

```
1 def update_usage(self, response):
2     if hasattr(response, 'usage_metadata') and response.usage_metadata:
3         self.usage[ total ] += response.usage_metadata.get('input_tokens', 0) + \
4                             response.usage_metadata.get('output_tokens', 0)
5         self.usage[ prompt ] += response.usage_metadata.get('input_tokens', 0)
6         self.usage[ completion ] += response.usage_metadata.get('output_tokens',
0)
```

2.10.4 Caching Benefits

Caching significantly reduces redundant API calls, decreases token consumption, improves response times, and lowers operational costs.

2.10.5 Exception Handling

The system manages exceptions by capturing them within `ThreadPoolExecutor` futures and raising them when `.result()` is called. Additionally, try-catch blocks are used in HTTP endpoints, and detailed error logging is enabled for debugging.

2.11 Technology Stack Summary

Component	Technology	Purpose
LLM Provider	Google Generative AI	Cloud-based AI services
Model	Gemini 2.5 Flash	Text + Vision (dual-purpose)
Framework	LangChain	LLM orchestration
Integration	langchain-google-genai	Google AI integration
Prompts	ChatPromptTemplate	Dynamic prompt rendering
Output Parsing	JsonOutputParser	Structured output validation
Caching	SQLiteCache	LLM response caching
Concurrency	ThreadPoolExecutor	Parallel task execution
Image Processing	PIL/Pillow	Base64 encoding
Data Validation	Pydantic	Schema validation
API Framework	FastAPI	REST API endpoints
UI Framework	Streamlit	Web interface
Report Generation	python-docx	Professional DOCX reports

2.12 Prompt Template Locations

All prompts are stored in: `src/prompt/en/`

Overall Analysis: This subsection includes the `overall_feature.txt` for feature extraction and `overall_analysis.txt` for feature interpretation.

Person Analysis: This subsection includes `person_feature.txt` for feature extraction, `person_analysis.txt` for interpretation, and `person_final_report.txt` for report generation.

Processing: This category contains prompts for synthesis (`analysis_merge.txt`), formatting (`merge_format.txt`), report generation (`final_result.txt`), signal detection (`signal_judge.txt`), and binary classification (`clf.txt`).

2.13 Key Insights

2.13.1 Architecture Type

The system uses a Multi-Agent Orchestration approach, with seven specialized agents coordinated through the HTPModel. It employs a Hybrid Model Usage strategy, where a single Gemini 2.5 Flash model handles both text and vision. The workflow combines Sequential and Parallel processing, using concurrent feature extraction workers followed by sequential synthesis.

2.13.2 Workflow Efficiency

Parallel execution of independent stages reduces latency, while caching minimizes redundant API calls. The streamlined PLUTO workflow provides an optimized path for performance-critical scenarios.

2.13.3 Safety & Quality

Quality and safety are ensured through automatic warning detection via signal analysis, binary classification with boolean output, conditional safety signal triggering, and professional report generation grounded in psychological insights.

2.13.4 Scalability

The system is designed for scalability with a configurable ThreadPoolExecutor worker count and a cloud-based LLM that requires no local inference. The stateless agent design supports horizontal scaling, while token usage monitoring aids in cost management.

3 Entity Relationship Diagram (ERD)

The Entity Relationship Diagram (ERD), shown in Figure 7, represents the complete logical structure of the database used in the House-Tree-Person (HTP) Assessment Platform. The schema captures all essential relationships among users, sessions, drawings, AI outputs, and the final assessment workflow.

Each entity is designed to support a modular, scalable, and audit-friendly workflow, ensuring traceability from raw student inputs to final psychological reports.

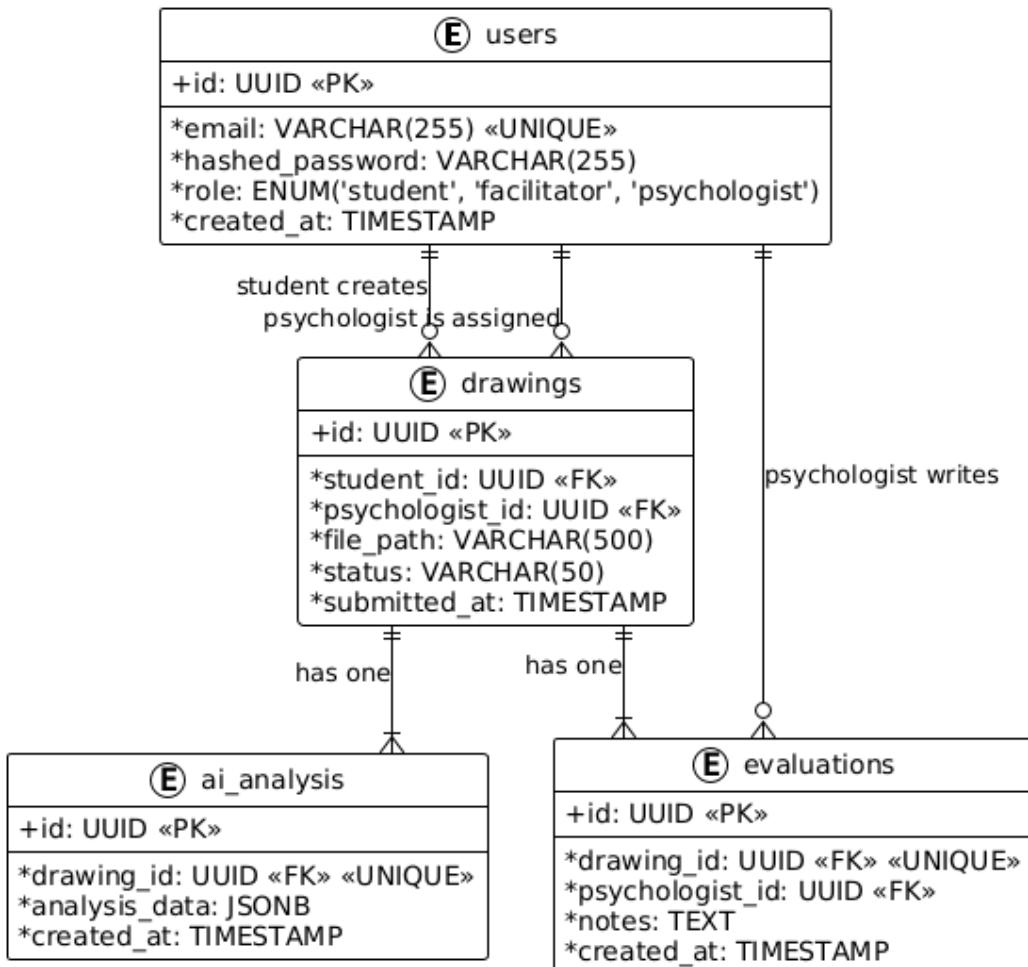


Figure 7: Entity Relationship Diagram (ERD)

3.1 Entities and Attributes

3.1.1 users

The **users** table stores all individuals interacting with the system across three distinct roles: students, facilitators, and psychologists.

- **id (UUID, PK)**: Unique identifier for each user.
- **email (String, Unique)**: The user's email, used for login.
- **hashed_password**: Securely stored hashed password.
- **role**: ENUM field indicating `student`, `facilitator`, or `psychologist`.
- **timestamps (created_at, updated_at)**: Track lifecycle events.

Relationships:

- Facilitators role assigns drawings to psychologists.
- Students participate in sessions and submit drawings.
- Psychologists review assessments and generate evaluation reports.

3.1.2 drawings

The **drawings** table is the central entity, tracking every submitted drawing and its progress through the assessment workflow

- **id (UUID, PK)**: Unique identifier for the drawing.
- **student_id (FK to users.id)**: Identifies the student who created the drawing.
- **psychologist_id (FK to users.id)**: Identifies the psychologist assigned to review the drawing. This field is the trigger for the AI analysis.
- **file_path (String)**: The server path to the stored image file.
- **status (String)**: Tracks the workflow state: submitted, processing, in_review, reviewed, or failed.
- **submitted_at (Timestamp)**: Tracks when the drawing was uploaded.

Relationships:

- Each drawing has one `ai_analysis` record (created after assignment).
- Each drawing has one final evaluation record (created by a psychologist).

3.1.3 ai_analysis

Stores the complete, raw JSON output from the AI model for a single drawing.

- **id (UUID, PK)**
- **drawing_id (FK to drawings.id)**: Links the analysis to a specific drawing.
- **analysis_data (JSONB)**: Stores the entire JSON object generated by the AI's pluto_workflow, including the formatted final report.
- **created_at (Timestamp)**: Tracks when the AI analysis was completed.

Relationships:

- Belongs to exactly one drawing.

3.1.4 assessments

Stores the final notes and conclusion written by a human psychologist after reviewing the drawing and the AI report.

- **id (UUID, PK)**
- **drawing_id (FK to drawings.id)**: Links the evaluation to a specific drawing.
- **psychologist_id (FK to users.id)**: Identifies the psychologist who wrote the notes.
- **notes (Text)**: The professional notes and final assessment provided by the psychologist.
- **created_at (Timestamp)**: Tracks when the evaluation was saved.

Relationships:

- Represents the final human review for one drawing.

3.2 ERD Summary

The database schema models the full assessment lifecycle implemented in the application:

1. A **student** registers and uploads a drawing, which is saved with a status of submitted.
2. A facilitator views all submitted drawings and assigns one to a specific psychologist.
3. This assignment updates the drawing's psychologist_id, changes its status to processing, and triggers the AI analysis in the background.
4. Once the AI analysis is done, a new ai_analysis record is created and the drawing's status becomes in_review.
5. The assigned psychologist views the drawing and its AI report, writes their professional notes, and saves their evaluation.
6. This creates an evaluations record and updates the drawing's status to reviewed, completing the feedback loop.

This structure is simpler than the original design but robustly supports the entire role-based workflow, traceability, and final data management required by the platform.

4 User Flow Implementation

The **House–Tree–Person (HTP) Psychological Assessment Platform** features three distinct, role-based web applications for Students, Facilitators, and Psychologists. These interfaces are integrated via a secure RESTful API to manage the lifecycle of an assessment.

The design emphasizes usability and minimalism, ensuring each role can complete tasks efficiently. The core of the system is the **Assessment** object, which transitions through a defined status lifecycle: **submitted** → **processing** → **in_review** → **reviewed**.

4.1 Registration and Login

Secure access is managed for all three roles, beginning with registration.

- **Step 1: Registration (Student)** A student (or other user) creates an account using the **Register** form. This collects an **Email**, **Password**, and **Role** (student, facilitator, or psychologist). This action calls `POST /api/register` to create the new user record.
- **Step 2: Login (All Roles)** All users authenticate via the **Login** screen. Submitting an email and password to `POST /api/token` (using `application/x-www-form-urlencoded`) validates credentials. Upon success, the API returns a **JWT access_token** and user details, including their **role**. The application stores this token and redirects the user to their respective dashboard.

Create Your Account

Join the HTP Assessment Platform

Email Address
student1@test.com

Password
.....

I am a...
Student

Sign Up

Already have an account? [Log In](#)

HTP Platform

Drawing Analysis for Social Innovation

Email Address
student1@test.com

Password
.....

Login

Don't have an account? [Sign Up](#)

Test Users (password is 'password'): student1@test.com, facilitator@test.com, psychologist1@test.com

Figure 8: User registration screen.

Figure 9: Login screen for all roles.

4.2 Student Portal

The Student Portal is focused on submission and tracking.

- **Step 1: Dashboard and Upload** After login, the student lands on their **Dashboard**. This screen provides two primary functions: viewing past submissions and uploading/drawing a new one. To submit, the student uses the "Upload Drawing" or "I'm finished" feature, which calls POST /api/drawings/upload with the image file (`multipart/form-data`). The backend creates a new **Assessment** record with the status **submitted**.

My Drawing Space

sahu@student.com Logout

Welcome to your HTP Drawing Assessment

You will be asked to create a drawing of a Person. Please upload your drawing below.

Session ID: A-124321
Date: October 2, 2025

Draw a Person

Clear Undo I'm Finished!

Instructions

- Step 1**
Draw a person on a piece of paper as well as you can.
- Step 2**
Take a clear photo or scan of your drawing.
- Step 3**
Click the upload box to select your image file.
- Step 4**
Click "I'm Finished!" to submit your drawing for analysis.

Figure 10: Student drawing their HTP drawing.

- **Step 2: Monitoring Submissions** The student tracks their assessment's progress on the "My Submissions" list, which fetches data from GET /api/my-submissions. The student can observe the status changes as the assessment is processed by the facilitator and psychologist.

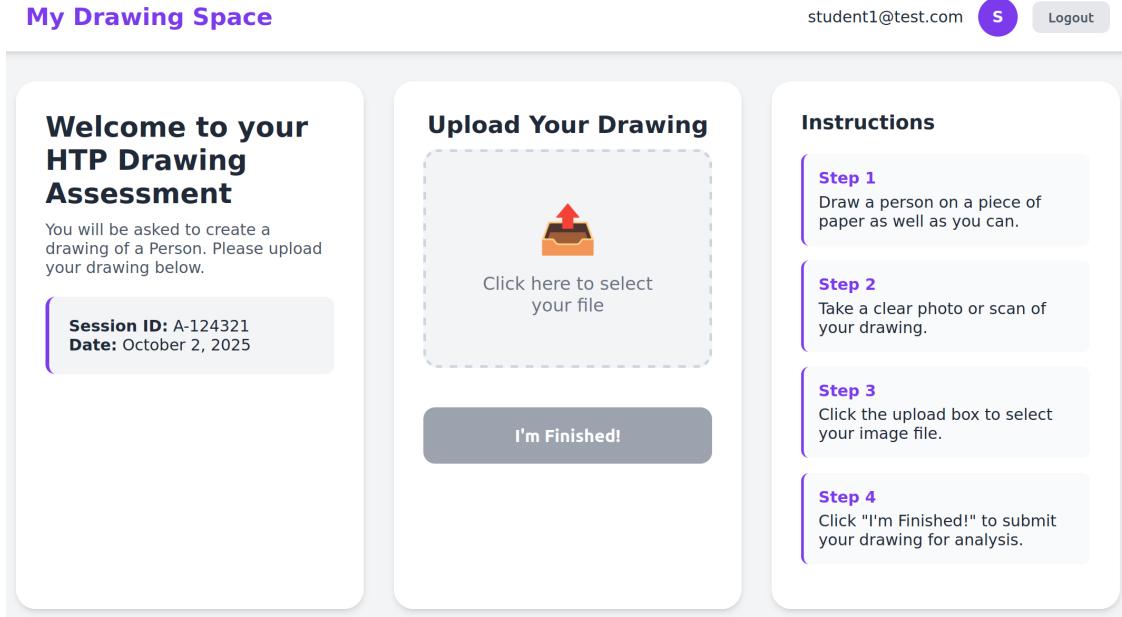


Figure 11: Student uploading their HTP drawing.

- **Step 3: Viewing Results** Once the status changes to **in_review**, the student can view the initial AI-generated report. After the psychologist completes their work, the status changes to **reviewed**, and the final **Evaluation Notes** from the psychologist become visible.

The screenshot shows a student dashboard. At the top, it says "My Drawing Space" and displays the email "student1@test.com". There is a purple circular icon with a white letter "S" and a "Logout" button.

Welcome to your HTP Drawing Assessment

Please choose how you would like to submit your drawing.

Draw on Screen **Upload Drawing**

Your Submission History

SUBMISSION ID	DATE	STATUS	ACTION
6a1c1289...	17/11/2025	REVIEWED	View Notes
b3c7c32d...	17/11/2025	SUBMITTED	-

Figure 12: Student dashboard showing a 'submitted' item.

Welcome to your HTP Drawing Assessment

Please choose how you would like to submit your drawing.

Psychologist's Notes

Take great care of yourself

Close

Your Submission History

SUBMISSION ID	DATE	STATUS	ACTION
6a1c1289...	17/11/2025	REVIEWED	View Notes
b3c7c32d...	17/11/2025	SUBMITTED	-

Figure 13: Student view of a finalized assessment.

4.3 Facilitator Portal

The Facilitator Portal is the administrative hub for managing and assigning assessments.

- **Step 1: Dashboard and Triage** The facilitator logs in to a **Dashboard** that displays all student submissions from across the system, retrieved via `GET /api/assessments/facilitator`. From this list, the facilitator can see which items are **submitted** and require assignment.

Facilitator Portal

facilitator@test.com F Logout

Dashboard

Monitor student progress and manage assessments.

3 TOTAL SUBMISSIONS	2 PENDING REVIEWS	1 COMPLETED ASSESSMENTS	0 FAILED ANALYSES
-------------------------------	-----------------------------	-----------------------------------	-----------------------------

Recent Submissions

STUDENT	SUBMITTED AT	STATUS	ASSIGNED TO	ACTION
student2@test.com	17/11/2025, 6:58:11 pm	SUBMITTED	Unassigned	Assign
student1@test.com	17/11/2025, 6:32:15 pm	REVIEWED	psychologist1@test.com	-
student1@test.com	17/11/2025, 6:31:37 pm	SUBMITTED	Unassigned	Assign

Figure 14: Facilitator dashboard with a list of all assessments.

- **Step 2: Assigning Assessments (Triggers AI)** This is the facilitator's key action.
 1. They select an assessment and choose a psychologist from a list (populated by GET /api/psychologists).
 2. They confirm the assignment, which calls PUT /api/drawings/{drawing_id}/assign/{psychologist_id}.
 3. **Crucially, this API call triggers two actions:** it assigns the drawing to the psychologist and schedules the background AI analysis (pluto_workflow). The assessment status is immediately set to **processing**.

The screenshot shows the Facilitator Portal dashboard. At the top, there are four summary boxes: 'TOTAL SUBMISSIONS' (3), 'PENDING REVIEWS' (1), 'COMPLETED ASSESSMENTS' (1), and 'FAILED ANALYSES' (0). Below this is a section titled 'Recent Submissions' with a table:

STUDENT	SUBMITTED AT	STATUS	ASSIGNED TO	ACTION
student2@test.com	17/11/2025, 6:58:11 pm	PROCESSING	psychologist2@test.com	-
student1@test.com	17/11/2025, 6:32:15 pm	REVIEWED	psychologist1@test.com	-
student1@test.com	17/11/2025, 6:31:37 pm	SUBMITTED	Unassigned	<button>Assign</button>

Figure 15: Facilitator assigning a drawing to a psychologist.

- **Step 3: Monitoring Progress** The facilitator monitors the dashboard as the assessment status automatically updates from **processing** to **in_review** (once the AI analysis is complete) and finally to **reviewed** (once the psychologist submits their evaluation).

The screenshot shows the Facilitator Portal dashboard, identical to Figure 15 but with a different status for the first row. The 'SUBMITTED AT' column shows '17/11/2025, 6:58:11 pm'. The 'STATUS' column for the first row now shows 'IN REVIEW' in a blue box, indicating the AI analysis has completed.

Figure 16: Facilitator view showing an assessment as 'in_review'.

4.4 Psychologist Portal

The Psychologist Portal is the expert review interface, combining AI analysis with clinical expertise.

- **Step 1: Dashboard and Queue** The psychologist logs in to their **Dashboard**, which displays a queue of assessments assigned specifically to them (`GET /api/assessments/psychologist`). They will only see items with the status `in_review`.

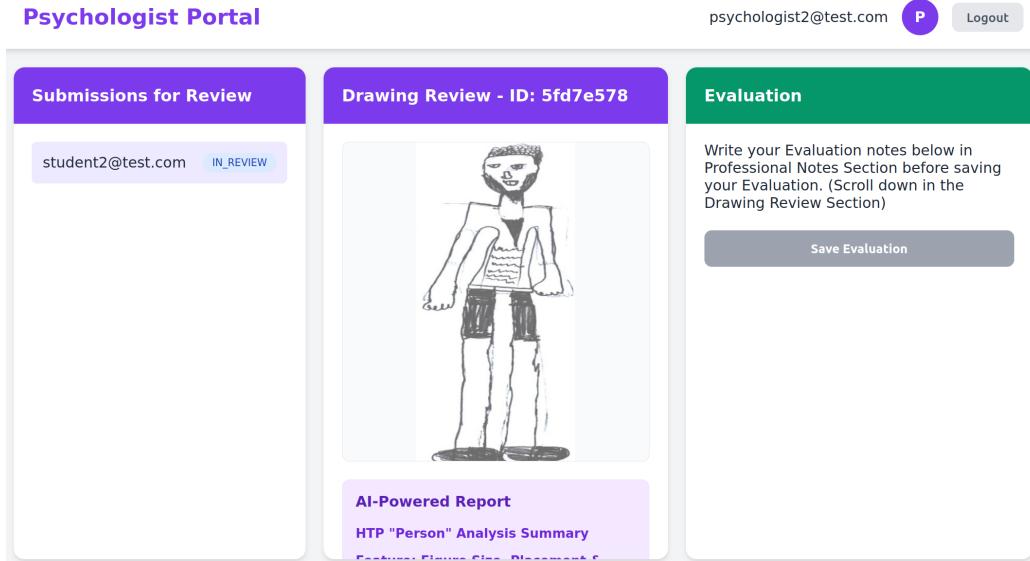


Figure 17: Psychologist's personal dashboard and work queue.

- **Step 2: Reviewing AI Analysis** The psychologist selects an assessment to open the **Detail View**. This screen presents the student's original drawing alongside the structured AI analysis (from the `ai_analysis` object), which includes classifications, signals, and generated text.

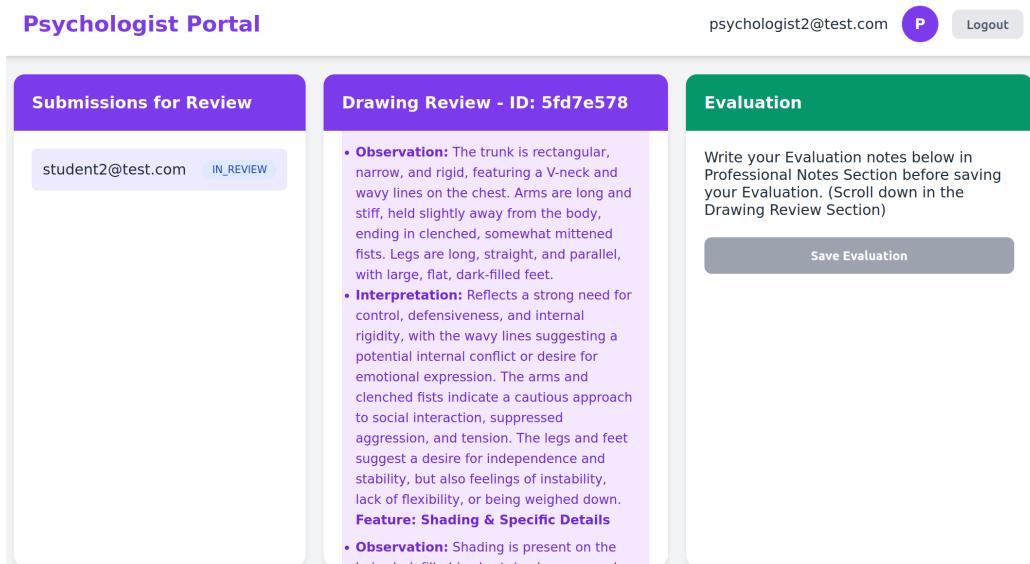


Figure 18: Psychologist reviewing the drawing and AI-generated report.

- **Step 3: Submitting Final Evaluation** After reviewing the drawing and the AI report, the psychologist adds their own clinical notes into an **Evaluation** text area. Submitting this form calls `POST /api/drawings/{drawing_id}/evaluate`. This API call saves the psychologist's notes and automatically updates the assessment's status to **reviewed**, marking the workflow as complete.

The screenshot shows the Psychologist Portal with the following layout:

- Submissions for Review:** Shows a submission from "student2@test.com" in the "IN REVIEW" state.
- Drawing Review - ID: 5fd7e578:** Displays a detailed psychological analysis of the drawing, mentioning defense mechanisms, anxiety, and aggression.
- Evaluation:** A green header section where the psychologist can write evaluation notes. It includes a note about saving notes before saving the evaluation and a "Save Evaluation" button.

Figure 19: Psychologist entering their final clinical evaluation.

4.5 Assessment Status Lifecycle Summary

The entire user flow is governed by the `Drawing.status` field, which dictates visibility and available actions for each role.

```
submitted -> (assign) processing -> (AI success) in_review -> (psych eval) reviewed
\-> (AI failure) failed
```

- **Submitted:** Initial state after student upload. Visible to Student and Facilitator.
- **Processing:** State after facilitator assignment. AI analysis is running.
- **In_Review:** State after AI analysis is complete. Visible to Psychologist for review.
- **Reviewed:** Final state after psychologist submits evaluation. Visible to all roles.
- **Failed:** State if the background AI analysis encounters an error.

5 APIs (Signatures, Status Codes, Responses, Requests)

All endpoints use JSON unless otherwise stated. Protected endpoints require an `Authorization: Bearer <token>` header.

5.1 Authentication Endpoints

5.1.1 POST /api/register

- **Description:** Create a new user account.
- **Request Body:** (UserCreate)

```
{  
    "email": "user@example.com",  
    "password": "plaintext",  
    "role": "student" || "facilitator" || "psychologist"  
}
```

- **Response (200 OK):** (User)

```
{  
    "id": "<uuid>",  
    "email": "user@example.com",  
    "role": "student"  
}
```

- **Errors:** 400 Bad Request if email is already registered.

5.1.2 POST /api/token

- **Description:** Obtain an access token (OAuth2 password grant).
- **Request:** application/x-www-form-urlencoded with form fields `username` (email) and `password`.
- **Response (200 OK):**

```
{  
    "access_token": "<jwt>",  
    "token_type": "bearer",  
    "user": { "id": "<uid>", "email": "...", "role": "..." }  
}
```

- **Errors:** 401 Unauthorized for incorrect credentials.

5.1.3 GET /api/users/me

- **Description:** Get current authenticated user's information.
- **Auth:** Bearer token required.
- **Response (200 OK):** User object.
- **Errors:** 401 Unauthorized.

5.2 Student Endpoints

5.2.1 POST /api/drawings/upload

- **Description:** Student uploads an HTP drawing image.
- **Auth:** Bearer token required; user role must be `student`.
- **Request:** `multipart/form-data` with a `file` field containing the image.
- **Response (200 OK):** Assessment object, including `id`, `file_path`, `status`, timestamps, and nested `student` object.
- **Errors:** 403 Forbidden if not a student; 400 Bad Request for invalid file type.

5.2.2 GET /api/my-submissions

- **Description:** Get a list of the current student's submissions.
- **Auth:** Bearer token required; user role must be `student`.
- **Response (200 OK):** A list of `Assessment` objects.
- **Errors:** 403 Forbidden if not a student.

5.3 Facilitator Endpoints

5.3.1 GET /api/assessments/facilitator

- **Description:** Get a list of all assessments for the facilitator dashboard.
- **Auth:** Bearer token required; user role must be `facilitator`.
- **Response (200 OK):** A list of `Assessment` objects.
- **Errors:** 403 Forbidden if not a facilitator.

5.3.2 GET /api/psychologists

- **Description:** Get a list of all users with the `psychologist` role (for assignment).
- **Auth:** Bearer token required; user role must be `facilitator`.
- **Response (200 OK):** A list of `User` objects.
- **Errors:** 403 Forbidden if not a facilitator.

5.3.3 PUT /api/drawings/{drawing_id}/assign/{psychologist_id}

- **Description:** Assign a drawing to a psychologist. This triggers the background AI analysis.
- **Auth:** Bearer token required; user role must be `facilitator`.
- **Response (200 OK):** The updated `Assessment` object.
- **Side Effects:** Sets the assessment's `psychologist_id`, updates `status = "processing"`, and enqueues the `run_ai_analysis` background task.
- **Errors:** 403 Forbidden if not facilitator; 404 Not Found if drawing or psychologist ID is invalid.

5.4 Psychologist Endpoints

5.4.1 GET /api/assessments/psychologist

- **Description:** Get a list of assessments assigned to the current psychologist.
- **Auth:** Bearer token required; user role must be `psychologist`.
- **Response (200 OK):** A list of `Assessment` objects (includes `ai_analysis` when available).
- **Errors:** `403 Forbidden` if not a psychologist.

5.4.2 POST /api/drawings/{drawing_id}/evaluate

- **Description:** Psychologist submits their final evaluation notes for an assessment.
- **Auth:** Bearer token required; user role must be `psychologist`.
- **Request Body:** (`EvaluationCreate`)

```
{  
    "notes": "Observations & interpretation..."  
}
```
- **Response (200 OK):** The newly created `Evaluation` object.
- **Side Effects:** Updates the assessment's `status = "reviewed"`.
- **Errors:** `403 Forbidden` if not the assigned psychologist; `404 Not Found` if drawing ID is invalid.

5.5 Common Status Codes and Notes

- `401 Unauthorized`: Invalid, expired, or missing token.
- `403 Forbidden`: Authenticated user does not have the required role for the endpoint.
- `404 Not Found`: The requested resource (e.g., drawing, user) does not exist.
- `400 Bad Request`: Invalid request data, such as a duplicate email during registration or a bad file upload.
- `500 Internal Server Error`: AI provider failures or other unhandled exceptions.

Implementation Notes: Endpoints return Pydantic models defined in `backend/app/schemas.py`. The `Assessment` schema includes nested `User` models for `student` and (optional) `psychologist`, as well as optional `ai_analysis` and `evaluation` objects. Uploaded image files are served statically from the `/uploads/<filename>` path.

6 Folder Structure

The project is organized into two main top-level directories: `Frontend` and `Backend`.

- `pluto-platform/`
 - `.env` — Environment variables (API keys, DB URL, secrets).
 - `docker-compose.yml` — Orchestration for all services (backend, frontend, DB).
 - `backend/`
 - * `Dockerfile` — Container definition for the backend.
 - * `main.py` — Main FastAPI application: routes, background tasks, static file mounts.

```

* requirements.txt — Python dependencies for the backend.
* app/
    · __init__.py
    · auth.py — Password hashing, JWT creation, and dependency injection for auth.
    · crud.py — Core database (Create, Read, Update, Delete) operations.
    · database.py — SQLAlchemy engine, session management, and Base model.
    · models.py — SQLAlchemy table models (User, Drawing, AIAnalysis, Evaluation).
    · schemas.py — Pydantic models for API data validation and serialization.
* src/
    · model_langchain.py — Core AI workflow logic and prompt orchestration.
    · prompt/en/
    · ... — All .txt files for prompt engineering.
- frontend/
    * Dockerfile — Multi-stage container definition for the React app (build & serve).
    * nginx.conf — Custom Nginx configuration for single-page application routing.
    * package.json — Node.js dependencies and project scripts.
    * public/
        · index.html — The HTML shell for the React application.
* src/
    · App.css — Unified stylesheet for all components.
    · App.js — Main component with client-side routing.
    · index.js — Entry point for the React application.
    · pages/
        · pages/LoginPage.js
        · pages/RegisterPage.js
        · pages/StudentDashboard.js
        · pages/FacilitatorDashboard.js
        · pages/PsychologistDashboard.js

```

7 Setup Instructions

7.1 Guide to Running the HTP Platform on a New Linux Machine

This section provides a complete, step-by-step guide for deploying the `pluto-platform` on any new Linux-based system. Because the platform has been fully containerized using Docker, installation is extremely simple and does not require manual setup of Python, Node.js, PostgreSQL, or other dependencies.

7.1.1 Step 1: Install Prerequisites

The only prerequisite on the new machine is **Docker**. Follow the steps below.

1. Open a Terminal on the Linux PC.
2. Update the Package Manager:

```
1 sudo apt update
```

3. Install Docker Engine:

```

1 sudo apt install -y apt-transport-https ca-certificates curl software-
  properties-common
2 curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
3   sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
4
5 echo deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
  docker-archive-keyring.gpg] \
6 https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable |
7   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
8
9 sudo apt update
10 sudo apt install -y docker-ce docker-ce-cli containerd.io

```

4. Install Docker Compose:

```

1 sudo apt install -y docker-compose-plugin

```

5. Add the User to the Docker Group:

This allows Docker commands to run without using `sudo`.

```

1 sudo usermod -aG docker ${USER}

```

Log out and log back in to apply the change.

6. Verify the Installation:

```

1 docker --version
2 docker compose version

```

7.1.2 Step 2: Prepare the Project Files

1. Unzip the Project:

```

1 unzip pluto-platform.zip
2 cd pluto-platform

```

Navigate to the project root:- `cd /path/to/pluto-platform`

2. Configure the API Key:

Edit the `.env` file located in the project root.

```

1 nano .env
2
3 Replace:- your_actual_google_gemini_api_key_here

```

with the valid API key. Save using `Ctrl+X`, then `Y`, then Enter.

7.1.3 Step 3: Build and Run the Application

1. Ensure you are in the `pluto-platform` root directory.
2. Run Docker Compose:

```

1 sudo docker compose up --build

```

This command:

- Builds container images for the frontend and backend.
- Starts all services: `db`, `backend`, and `frontend`.

You will see logs such as:

- `http_postgres_db` | database system is ready
- `http_backend` | Uvicorn running on `http://0.0.0.0:8000`
- `http_frontend` | starting worker process

7.1.4 Step 4: Access the Application

1. Open any web browser on the machine.
2. Navigate to:

```
1 http://localhost:3000
```

The platform is now fully operational. You may log in as different users, upload drawings, assign tasks, and review reports.

7.1.5 Stopping the Application

- Go to the terminal running Docker Compose.
- Press:

```
1 CTRL + C
```

- To remove containers and network:

```
1 docker compose down
```

The Dockerized architecture ensures the entire system can be deployed with a single command on any compatible Linux system.

7.2 Guide to Running the HTP Platform on a New Windows Machine

This section provides a complete, step-by-step guide for deploying the `pluto-platform` on any Windows system. Because the platform is fully containerized using Docker Desktop, the setup process is simple and does not require manual installation of Python, Node.js, PostgreSQL, or other dependencies.

7.2.1 Step 1: Install Prerequisites

The only prerequisite on a new Windows machine is **Docker Desktop**. Follow the steps below.

1. Check System Requirements:

- Windows 10 or 11 (64-bit), Pro, Enterprise, or Education edition.
- Ensure Windows Subsystem for Linux 2 (WSL 2) is enabled. If not, open PowerShell as Administrator and run:

```
1 wsl --install
```

Then restart your computer.

2. Download and Install Docker Desktop:

- Visit the official Docker Desktop page: <https://www.docker.com/products/docker-desktop/>

- Download the Windows installer.
- Run the downloaded .exe file and follow the installation wizard.
- Restart your system if prompted.

3. Start and Verify Docker Desktop:

- Launch Docker Desktop from the Start Menu.
- Wait until the status reads “*Docker Desktop is running*”.
- Open PowerShell or Command Prompt and verify Docker installation:

```
1 docker --version
2 docker compose version
```

7.2.2 Step 2: Prepare the Project Files

1. Unzip the Project Files:

- Locate pluto-platform.zip.
- Right-click to Extract All....
- Choose a folder such as C:\Users\YourUser\Documents\Projects.

2. Navigate to the Project Directory:

```
1 cd C:\Users\YourUser\Documents\Projects\pluto-platform
```

3. Configure the API Key:

- Open the .env file with Notepad.
- Replace:
1 your_actual_google_gemini_api_key_here
with the valid Google API key.
- Save and close the file.

7.2.3 Step 3: Build and Run the Application

This single command runs the entire platform using Docker.

1. Ensure your PowerShell or Command Prompt is inside the pluto-platform directory.
2. Run Docker Compose:

```
1 docker compose up --build
```

3. This command:

- Builds container images for the backend and frontend.
- Starts all three services: db, backend, frontend.

4. Wait for logs such as:

- http_postgres_db | database system is ready
- http_backend | Uvicorn running on http://0.0.0.0:8000
- http_frontend | start worker process

7.2.4 Step 4: Access the Application

1. Open a browser (Chrome, Firefox, Edge).
2. Navigate to:

```
1 http://localhost:3000
```

The platform is now fully operational on Windows. You may log in using test accounts, upload drawings, assign tasks, and view reports.

7.2.5 Stopping the Application

- Press:

```
1 CTRL + C
```

in the terminal running Docker.

- To remove containers and the network:

```
1 docker compose down
```

Docker Desktop ensures the entire system can be run with one command, making deployment on Windows as easy as on Linux.

8 Individual Contribution:

The development of the HTP Drawing Analysis Platform was divided into two primary phases: foundational research and design, followed by full-stack implementation and integration. The contributions of the six team members are detailed as follows:

Aditya Raj Singh - Project Lead, System Architect & Full-Stack Developer

As the Project Lead, Aditya orchestrated the entire technical execution of the platform, from initial backend design to the final deployment. This individual was responsible for bridging all components, solving critical integration challenges, and ensuring the project evolved from a collection of parts into a cohesive, functional application.

- **Lead System Architect:** Designed the complete end-to-end system architecture, defining the communication protocols between the React frontend, FastAPI backend, and PostgreSQL database. Key decisions on the technology stack were made, including the use of JWT for authentication, SQLAlchemy for database modeling, and Docker for containerization.
- **Full-Stack Implementation & Integration:** Wrote and integrated the most critical pieces of both the frontend and backend. This included developing the core user authentication logic, building the file upload and background task processing for the AI, and connecting the React components to the backend API endpoints.
- **DevOps & Deployment:** Managed the entire containerization process by authoring the `Dockerfiles` for both services and the `docker-compose.yml` file that orchestrates the application. Debugged and solved all major deployment issues, including networking, dependency conflicts, and server configurations (Nginx), ensuring the project's portability.
- **Team Leadership & Mentorship:** Guided the implementation team (Ayush Sahu and Himanshu Gupta) through the development process, assigning specific tasks for the backend API and frontend components. Led the debugging efforts and provided technical solutions to overcome roadblocks, ensuring the implementation team worked efficiently.

Ayush Sahu - Lead Backend Developer

Ayush was a key contributor to the server-side development, focusing on translating the architectural design into a robust, data-driven API.

- **Database Implementation:** Implemented the complete set of database models using SQLAlchemy, accurately translating the project's Entity Relationship Diagram (ERD) into Python code. This included defining tables for users, drawings, AI analyses, and psychologist evaluations with all their relationships.
- **API Endpoint Development:** Developed the majority of the CRUD (Create, Read, Update, Delete) operations and the corresponding FastAPI endpoints. This involved writing the business logic for fetching role-specific data (e.g., `get_assessments_for_facilitator`) and handling data persistence for all user actions.
- **Authentication Logic:** Assisted in implementing the security layer by creating the functions for password hashing with `passlib` and integrating the user retrieval logic into the JWT authentication flow.

Himanshu Gupta - Lead Frontend Developer

As a core member of the implementation team, Himanshu was responsible for bringing the user interface to life, working closely with the project lead to build out the dynamic React components and connect them to the backend API.

- **React Component Development:** Built the primary React components for all three user dashboards (Student, Facilitator, and Psychologist). This included managing component state, handling user input, and structuring the JSX to create interactive and responsive interfaces.
- **API Integration:** Implemented the client-side API calls using axios for all frontend features, such as logging in, uploading files, fetching dashboard data, assigning tasks, and submitting evaluations.
- **Dynamic UI Features:** Developed key interactive features, including the "Assign" dropdown for facilitators, the "View Notes" modal for students, and the "Save Evaluation" functionality for psychologists, ensuring an intuitive user experience.

Harshit Lalwani - AI Research & Prompt Engineering

Harshit was part of the foundational team and focused on the core AI engine, the PsyDraw model. This member's work provided the "brains" of the platform's analysis capabilities.

- **PsyDraw Model Adaptation:** Took the lead on researching and adapting the `psydraw_fork` project, being responsible for understanding its multi-agent architecture and preparing it for integration.
- **Initial Prompt Engineering:** Authored the first drafts of the specialized prompts for analyzing the "House," "Tree," and "Person" elements, establishing the basis for the AI's analytical framework.
- **AI Output Validation:** Performed initial tests on the raw output of the PsyDraw model to validate its psychological coherence and relevance, ensuring it was a viable engine for the platform.

Soham Chaterjee - UI/UX Design & Prototyping

Soham led the initial design phase of the project, focusing on user experience and creating the visual blueprint for the application.

- **User Interface Design:** Designed the user interfaces for the Login page, Student Portal, Facilitator Dashboard, and Psychologist Portal, creating the visual language, including color schemes, typography, and layout.

- **HTML/CSS Mockups:** Developed the initial static HTML and CSS prototypes for all user-facing pages. These high-fidelity mockups served as the direct reference for the final React implementation, ensuring the end product matched the design vision.
- **User Flow Mapping:** Outlined the primary user journeys and workflows, which informed the navigational structure of the application and the features required for each user role.

Ananth - Foundational Research & Documentation

Ananth was responsible for the project's initial research and documentation, which grounded the project in established psychological principles and ensured clarity in project goals.

- **HTP Indicator Research:** Conducted a literature review on HTP test indicators from psychological research. This research was crucial for informing the prompt engineering process and ensuring the AI's analysis was based on established domain knowledge.
- **Project Documentation:** Wrote the initial project documentation, including the problem statement, user stories, and the initial draft of the final report. This provided a clear reference for the entire team throughout the development process.
- **Content Creation:** Authored the user-facing text and instructional content that appears throughout the application's user interface.