

ExoHabAI: Technical Reference Manual

Project Name: ExoHab AI-Powered Habitability Analyzer

Version: 1.0.0

Developer: Avula Maneeswara Venkata Sai

Stack: Python (Flask), Scikit-Learn, XGBoost, Plotly.js

1. System Architecture

ExoHab utilizes a **Monolithic MVC (Model-View-Controller)** architecture designed for low-latency inference and high interpretability.

High-Level Data Flow

- Input Layer:** User uploads a raw NASA Archive CSV or inputs single planet parameters via the Web UI.
- Preprocessing Layer (Physics Engine):** Missing astronomical data is imputed using Keplerian and Stefan-Boltzmann physics laws.
- Inference Layer (AI Core):** The processed vector is fed into an XGBoost Classifier trained on balanced data.
- Explainability Layer (XAI):** A SHAP (Shapley Additive Explanations) TreeExplainer calculates feature contribution scores.
- Presentation Layer:** Results are rendered via Server-Side Templating (Jinja2) and interactive JavaScript (Plotly).

2. Core Modules

2.1 The Physics Engine (physics_engine.py)

Unlike standard ML pipelines that use statistical imputation (mean/median), ExoHab uses **Domain-Specific Imputation**.

- Kepler’s Third Law: Used to derive Orbital Period (P) or Semi-Major Axis (a) if one is missing.

$$P^2 \propto a^3$$

- Luminosity Derivation: Used to calculate Star Luminosity (L) if Radius (R) and Temperature (T) are known.

$$L = 4\pi R^2 \sigma T^4$$

- **Habitable Zone Calculation:** Estimates the "Goldilocks" boundaries based on calculated luminosity to generate a preliminary Habitability_Score.

2.2 Machine Learning Pipeline (model_utils.py)

The classification engine is built on **XGBoost (Extreme Gradient Boosting)**, selected for its execution speed and performance on structured data.

- **Algorithm:** XGBClassifier
- **Training Strategy:**
 - **Imbalance Handling:** Utilized **SMOTE (Synthetic Minority Over-sampling Technique)** to generate synthetic examples of "Habitable" planets, addressing the severe 99:1 class imbalance in raw NASA data.
 - **Hyperparameter Tuning:** Optimized using RandomizedSearchCV to find the ideal learning rate, max depth, and estimators.
- **Performance Optimization:**
 - **Vectorization:** Bulk predictions use Pandas to_dict('records') conversion to bypass slow DataFrame indexing (iloc), reducing processing time for 4,000 records from ~45s to ~2s.
 - **Robustness:** Custom safe_float() sanitization prevents NaN or Infinity values from crashing the JSON serialization layer.

2.3 Explainability Engine (explainability.py)

To solve the "Black Box" problem, ExoHab integrates **SHAP**.

- **Method:** shap.TreeExplainer
- **Functionality:**
 - Calculates the marginal contribution of each feature (e.g., *Stellar Temp*) towards the final prediction probability.
 - **Global Interpretation:** Generates summary plots (Beeswarm) to show overall model behavior.
 - **Local Interpretation:** Generates individual "Force Plots" for single-planet analysis (The "Why?" button feature).

3. Data Science Workflow

3.1 Data Selection (Acquisition)

The dataset was sourced from the NASA Exoplanet Archive (Planetary Systems Composite Data), recognized as the global standard for confirmed exoplanet parameters.

- **Source:** NASA Exoplanet Archive API / Bulk Download.
- **Feature Selection:** Out of 300+ available columns, we selected 10 critical physical features that directly influence habitability, based on astrobiological constraints:
 - **Planetary:** Radius, Mass, Orbital Period, Equilibrium Temperature.
 - **Stellar:** Star Radius, Star Mass, Star Temperature, Luminosity, Metallicity.
- **Target Variable:** A custom "Habitability Label" was derived based on the Kopparapu Habitable Zone boundaries (Conservative vs. Optimistic).

3.2 Data Cleaning & Preprocessing

Real-world astronomical data is sparse and noisy. We implemented a rigorous cleaning pipeline:

1. **Physics-Based Imputation:** Instead of using statistical mean/median (which is scientifically inaccurate for astronomy), we used a custom Physics Engine:
 - **Missing Period:** Calculated using Kepler's Third Law.
 - **Missing Luminosity:** Derived from Stefan-Boltzmann Law ($L = 4\pi R^2 \sigma T^4$).
2. **Handling NaN:** Remaining missing values were handled using KNN Imputation or dropped if critical parameters (like Planet Radius) were absent.
3. **Scaling:** Features were normalized using StandardScaler to ensure that features with large magnitudes (like Star Temperature) did not dominate the model gradients.

3.3 Model Selection

We evaluated multiple algorithms before finalizing the architecture:

- **Logistic Regression:** Discarded due to inability to capture non-linear relationships in orbital mechanics.
- **Random Forest:** Good performance, but slower inference time and large model size.
- **XGBoost (Selected):** Chosen for three reasons:
 1. **Gradient Boosting:** Superior performance on structured/tabular data.
 2. **Speed:** Faster training and inference time for real-time web usage.
 3. **Feature Importance:** Native support for ranking features, which aids explainability.

3.4 Training Strategy (Handling Imbalance)

A major challenge was the Class Imbalance. Habitable planets are extremely rare (<1% of the dataset), causing standard models to be biased towards "Non-Habitable."

- **Solution:** We applied SMOTE (Synthetic Minority Over-sampling Technique) during training.
- **Process:** SMOTE generated synthetic examples of "Habitable" and "Optimistic" planets in the feature space, balancing the dataset to a 1:1:1 ratio. This ensured the model learned to recognize rare Earth-like worlds effectively.

3.5 Testing & Evaluation

The model was evaluated on a strictly held-out Test Set (20% of data) that was *not* seen during training or SMOTE augmentation.

Metrics Achieved:

- **Accuracy:** ~98% (Overall correctness).
 - **Precision (Habitable Class):** High precision was prioritized to minimize "False Positives" (telling users a dead rock is habitable).
 - **Recall (Habitable Class):** High recall ensured we didn't miss any potentially Earth-like candidates.
 - **F1-Score:** Balanced score indicating robust performance across all three classes (Habitable, Optimistic, Non-Habitable).
-

4. API Documentation

The application exposes RESTful endpoints for internal and external consumption.

POST /predict_single

Analyzes a single planet based on JSON input.

- **Input:** JSON object containing P_RADIUS, S_TEMP, P_PERIOD, etc.
- **Output:** JSON object containing:
 - prediction: "Habitable" | "Optimistic" | "Non-Habitable"
 - score: 0-100 (derived from probability)
 - reasons: List of top 3 features influencing the decision.

POST /predict_bulk

Processes a CSV file containing multiple planets.

- **Input:** multipart/form-data (CSV file).
- **Output:** JSON Array of result objects.
- **Constraint:** Requires gunicorn --timeout 120 for files >5,000 rows.

POST /visualize_user_data

Generates statistical charts for the uploaded dataset.

- **Input:** JSON Array of planet data + limit boolean.
 - **Output:** Base64 encoded images (PNG) for:
 - Correlation Matrix (Heatmap)
 - PCA Projection (2D)
 - t-SNE Clusters (2D)
 - Verdict Distribution (Pie Chart)
-

5. Deployment Infrastructure

- **Platform:** Render (Cloud PaaS)
 - **Web Server:** Gunicorn (Green Unicorn) WSGI server.
 - **Workers:** Configured with sync workers (due to Free Tier CPU limits).
 - **Environment:**
 - Python 3.10+
 - Dependencies managed via requirements.txt.
 - Port binding via `os.environ.get('PORT')`.
-

6. Future Roadmap

1. **Spectral Analysis:** Integration of CNNs to analyze spectral light curves for atmospheric composition detection.
2. **Real-Time API Sync:** Automate the daily fetching of new confirmed planets from the NASA Exoplanet Archive API.
3. **Mobile Application:** Development of a React Native companion app for visualization.