**API Documentation**

**Overview**

The API is built using Flask and provides endpoints for Small Angle Scattering (SAS) analysis. It integrates with a machine learning model to predict particle morphology and dimensions from scattering data.

**Key Components**

**1. API Endpoints**

**/upload (POST)**

* **Purpose**: Upload and process scattering data files
* **Input**: Form data with file upload
* **Output**: JSON response with processing results
* **Error Handling**: Returns appropriate error messages for invalid files or processing failures

**/save\_to\_database (POST)**

* **Purpose**: Save processed data to the database
* **Input**: JSON data containing scan information
* **Output**: Success/failure status
* **Error Handling**: Database operation errors are caught and logged

**/simulate\_graph (POST)**

* **Purpose**: Generate simulated scattering curves
* **Input**: JSON data with model parameters
* **Output**: JSON response with simulated data points
* **Error Handling**: Validates input parameters and model types

**/health\_check (GET)**

* **Purpose**: Check API health status
* **Output**: JSON with API status information

**2. Logging and Monitoring**

The API implements comprehensive logging using structlog with:

* Request tracking
* Performance metrics
* Error logging
* Endpoint usage statistics

**Machine Learning Model Integration**

**Model Architecture**

The ML model consists of two main components:

1. **Morphology Classifier**: Hierarchical SVM-based classifier
2. **Dimension Regressor**: Kernel Ridge Regression models

**Model Initialization**

The model is initialized through startup.py which:

1. Loads training data
2. Initializes classifiers and regressors
3. Sets up the hierarchical decision structure

**Replacing the ML Model**

To replace the current ML model with a new one, follow these steps:

1. **Prepare New Model Files**

* Place new model files in the hierarchical\_SAS\_analysis-main 2 directory
* Ensure the new model follows the same interface as the current one
* Also make sure to replace the imports in the files.

2. **Update Model Initialization**

* Modify startup.py to load your new model:
* def init():
* *# Your new model initialization code*
* *# Must return (classifiers, hierarchical\_map, regressors)*
* *return* (new\_classifiers, new\_hierarchical\_map, new\_regressors)

1. **Update Prediction Functions**

Ensure your new model implements these key functions:

Apply to full\_send.py

   def predict\_morphology(*tspec*, *classifiers*, *hierarchical\_map*):

*# Your morphology prediction code*

*return* predicted\_morphology

def predict\_dimensions(*tspec*, *regressors*, *morphology*):

*# Your dimension prediction code*

*return* predicted\_dimensions

1. **Model Requirements**

Your new model must:

* Accept scattering data in the same format (log10 scaled)
* Return predictions in the same structure
* Support the same morphology types:
* cylinder
* disk
* sphere
* cs\_cylinder (core-shell cylinder)
* cs\_disk (core-shell disk)
* cs\_sphere (core-shell sphere)

1. **Testing**

* Test the new model with the existing API endpoints
* Verify predictions match the expected format
* Ensure performance meets requirements

**Current Model Details**

The current model uses:

* SVM classifiers with RBF kernel for morphology prediction
* Kernel Ridge Regression for dimension prediction
* Hierarchical decision structure for morphology classification
* Training data from lowar16 dataset

**Data Format Requirements**

* Input scattering data should be log10 scaled
* Background constant of 0.001 is applied
* Data is normalized using scale\_highq function

**Error Handling**

The API includes comprehensive error handling for:

* Invalid input data
* Model prediction failures
* Database operations
* File operations

**Performance Monitoring**

The API tracks:

* Request durations
* Error rates
* Endpoint usage
* Morphology-specific metrics