Low-Level Design Report: Cryptocurrency Liquidity Prediction

1. Introduction

This Low-Level Design (LLD) document provides a detailed view of the components involved in the Cryptocurrency Liquidity Prediction project.

The objective is to design a robust system that processes historical cryptocurrency data, engineers relevant features, trains predictive models,

and serves predictions through a web API built with Flask.

2. Technology Stack

- Programming Language: Python 3.x
- Libraries: pandas, scikit-learn, xgboost, matplotlib, seaborn, Flask
- Deployment: Flask web server, optionally Docker
- Tools: Jupyter Notebooks, GitHub for version control

3. Project Structure

The repository is structured as follows:

- app.py : Main Flask application entry point
- src/
    - data_preprocessing.py : Handles missing values and data cleaning
    - feature_engineering.py : Feature extraction and transformations
    - model_trainer.py : Training machine learning models and saving them
    - model_evaluator.py : Evaluates model performance metrics

- templates/
  - index.html : Web UI for user interaction
- static/ : Stores CSS or JS files

4. Module-Level Design

- app.py:
  - Defines endpoints /train and /predict.
  - Integrates model loading, preprocessing and prediction steps.
  - Uses Flask's routing mechanism.

- data_preprocessing.py:
  - Functions: handle_missing_values, convert_datetime, normalize_data
  - Input: Raw CSV dataset
  - Output: Cleaned pandas DataFrame

- feature_engineering.py:
  - Creates new features (rolling averages, volatility, ratios).
  - Encodes categorical variables if needed.

- model_trainer.py:
  - Trains XGBoost and BaggingRegressor models.
  - Performs GridSearchCV tuning.
  - Saves the best model as model.pkl

- model_evaluator.py:
  - Calculates RMSE, MAE, R2 score
  - Compares model predictions against true values

5. Data Flow Description

1. User uploads dataset via Flask UI or terminal.
2. Data flows into preprocessing and cleaning pipeline.
3. Feature engineering enhances the dataset.
4. Model training builds and tunes models.
5. Evaluator checks performance and saves metrics.
6. Trained model used in /predict endpoint for real-time inference.

6. Error Handling and Logging

- try-except blocks used in each module to catch and log errors.
- Logging module is used to log runtime information, errors, and warnings.

7. Deployment Notes

- Install dependencies using requirements.txt.
- Run Flask app with `python app.py`.
- For production, consider containerizing the app using Docker.