Project Context and Knowledge Base: BTC-Price-Predict

This document serves as a comprehensive knowledge base for the Bitcoin Price Prediction project, using a Bi-Directional LSTM (BiLSTM) model. It outlines the project's core objective, technical specifications, and the functionality of every file, enabling seamless transition and future support.

1. Project Goal and Core Strategy

Area	Description
Objective	Predict the price of **BTC/USDT** using 1-minute (1m) candlestick data.
Model	Stacked Bi-Directional Long Short-Term Memory (BiLSTM) network.
Forecasting	**Multi-step prediction:** Uses a sequence of 60 minutes (`t-60` to `t-1`) to forecast the next 5 minutes (`t+1` to `t+5`).
Ensemble	The prediction process leverages multiple trained models (`.h5` files) to provide an averaged, robust **Consensus View**.
Key Features	Close Price and Relative Strength Index (RSI). RSI is calculated over a 14-period lookback.
Environment	Python 3.9+, TensorFlow 2.x, Pandas, CCXT (for data fetching).

2. Global Configuration and Hyperparameters

The following constants are critical and must be consistent across `train.py` and `predict.py`.

Constant	Value	Description
`LOOKBACK_TIMESTEPS`	`60`	The length of the input sequence (in minutes) fed to the LSTM.
`PREDICTION_HORIZON`	`5`	The number of future minutes (timesteps) the model is trained to predict in a single output.
`NUM_FEATURES`	`2`	The number of features used: `['Close', 'RSI']`.
`RSI_PERIOD`	`14`	The lookback period used for calculating the RSI feature.
`SYMBOL`	`'BTC/USDT'`	The trading pair being analyzed.
`TIMEFRAME`	`'1m'`	The time granularity of the candlestick data.
`EXCHANGE_ID`	`'binance'`	The cryptocurrency exchange used for data fetching.
`EPOCHS`	`20`	The number of training epochs (in `train.py`).

3. Detailed File Breakdown

File Name	Primary Function	Key Logic
`train.py`	**Model Training and Persistence.**	1. Calls `data_loader` to fetch and prepare data (scaling, sequence generation). 2. Calls `model_builder` to create the BiLSTM model. 3. Trains the model for 20 epochs. 4. Saves the trained model (e.g., `cat2_trained_model.h5`) to the `models/` directory.

`predict.py`	**Ensemble Real-Time Prediction.**	1. Loads the original training data (`raw_btc_data.csv`) to re-fit the `MinMaxScaler` . 2. Fetches the latest live data sequence (60 minutes) from Binance. 3. Loads ALL `.h5` models in `models/` . 4. Runs predictions for each model. 5. **Crucial Fix:** Handles TensorFlow logging attribute errors to ensure model loading succeeds. 6. Displays comparative results and the **Consensus View**.
`data_loader.py`	**Data Preparation Pipeline.**	1. `fetch_data` **:** Uses `ccxt` to download OHLCV data and caches it locally. 2. `calculate_optimized_rsi` **:** Vectorized Pandas function to compute RSI. 3. `preprocess_data` **:** Applies feature selection, RSI calculation, and normalization (`MinMaxScaler`). 4. `create_sequences` **:** Generates the time series input (X, shape `(N, 60, 2)`) and output (Y, shape `(N, 5, 2)`) matrices for supervised learning.
`model_builder.py`	**BiLSTM Architecture Definition.**	1. Defines the **Stacked BiLSTM** structure. 2. Uses `tf.keras.layers.Bidirectional(LSTM(...))` for sequence processing. 3. Includes `BatchNormalization` and `Dropout` layers for stability and regularization. 4. Output structure is reshaped to match the required `(5, 2)` prediction horizon.
`rolling_backtest_strategy.py`	**Walk-Forward Validation.**	Script used to systematically test the model's performance on unseen data by retraining or recalibrating the model at set intervals, moving forward in time. (Used for performance evaluation and graph generation like `Figure_4.png`).
`sequential_prediction_viz.py`	**Visualization Utility.**	Used to compare the model's multi-step predictions against actual observed values over a historical window, generating plots like those in the `Graphs/` folder.
`requirements.txt`	**Dependency Manifest.**	Frozen list of exact Python package versions (`tensorflow` , `ccxt` , `pandas` , `numpy` , `scikit-learn`) required for the environment.

4. Data Flow and Consistency

A. Scaling Consistency (CRITICAL)

The `MinMaxScaler` **must** be fitted *only* on the training data (`data/raw_btc_data.csv`).

- **Training (`train.py`):** The scaler is fitted and then used to scale both X and Y.
- **Prediction (`predict.py`):** The training data is reloaded, a *new* scaler is fitted on it, and this scaler is then used to normalize the latest live input sequence and denormalize the model's output prediction.

B. Feature Consistency

Both `train.py` and `predict.py` **must** use the following steps in order:

1. Load OHLCV data.
2. Calculate RSI (`RSI_PERIOD=14`).
3. Select features: `['Close', 'RSI']` (`NUM_FEATURES=2`).
4. Generate sequences (`LOOKBACK_TIMESTEPS=60`).

C. Model Saving

Trained models are saved using the Keras HDF5 format (`.h5`) in the `models/` directory. Filenames are typically `[name]_trained_model.h5`.

5. Repository Maintenance

- `.gitignore`: Essential for excluding large binary files and environment folders:
  - `venv/`
  - `data/`
  - `models/*.h5`
- **Update Procedure:** If new dependencies are added, run `pip freeze > requirements.txt` to update the manifest before