



**Final Project Report**

**Advanced Computer Programming**

**Crypto Tracker with Trend Prediction**

**Group : 9**

**Instructor : DINH-TRUNG VU**

**2025-06**

# Introduction

## Group Information

1. **Group Project Repository**: [Jantsagdorj/ACP-AU-1132: Project for Advanced Computer Programming Course - AU-1132 - Group](https://github.com/Jantsagdorj/ACP-AU-1132)
2. **Group members**:
   1. Bayarmagnai Munkhbat– 113021195 (leader)
   2. Chinzorig Battulga – 113021191
   3. Yesuijin Batmunkh – 113021192
   4. Ariun-Erdene Sodnombayar – 113021196
   5. Tselmeg Gantulga – 113021201

## Objective

The objective of this project is to develop a fully functional and user-friendly interactive web application that enables users to monitor cryptocurrency markets in real time, review historical price data, and generate short-term price predictions using machine learning algorithms. This tool aims to serve:

* Data science students exploring time series modeling and Flask web development.
* Cryptocurrency investors and traders looking for a simple, visual, and predictive interface.
* Anyone interested in combining financial data with AI-based forecasting models.

# System Architecture and Technologies

## Overview

The application is structured into four key components:

* **Frontend Interface** for user interaction and visualization.
* **Backend Server** using Flask to route requests and serve data.
* **Data Collector** module that fetches historical cryptocurrency data from external APIs.
* **Machine Learning Engine** that processes historical data and generates future predictions.

## Technologies Used

### Backend

* **Flask:** Used as the primary web framework for routing, templating, and serving dynamic content.
* **requests:** Enables HTTP calls to fetch live data from the CoinGecko API.

### Frontend

* **HTML/CSS:** Provides the structure and visual style of the site.
* **Chart.js:** A JavaScript library used to render historical trend line charts dynamically within the browser. It supports responsive and interactive data visualization.
* **Jinja2:** Embedded within Flask templates for dynamic content rendering.

### Data Processing and Machine Learning

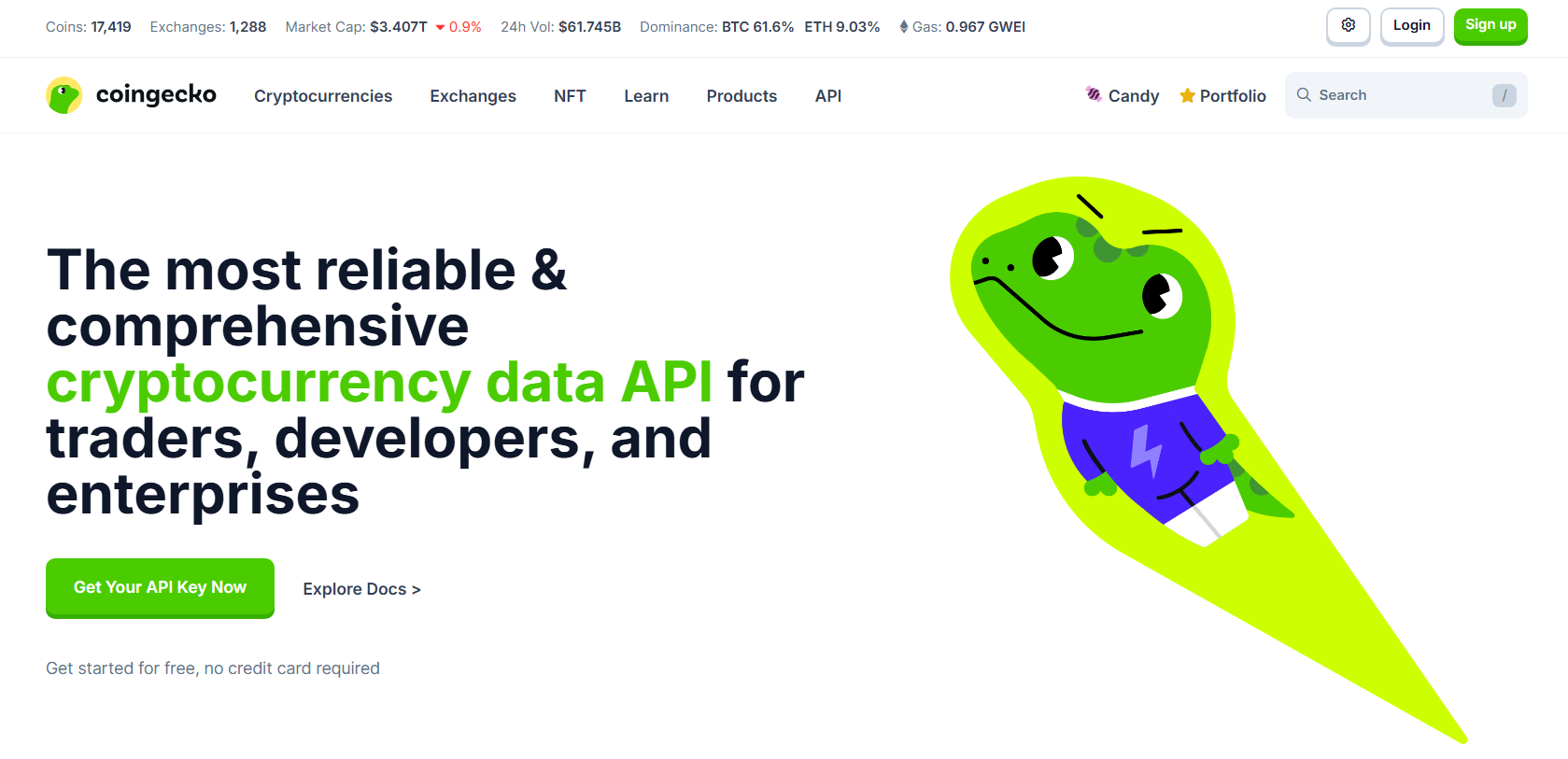
* **pandas:** Used extensively to clean, structure, and analyze the CSV data extracted from the API.
* **numpy:** Supports numerical feature engineering and computation.
* **xgboost:** Implements the core regression model used to forecast near-future cryptocurrency prices.
* **scikit-learn:** Provides tools for preprocessing, pipeline integration, and model persistence via joblib.

# Features and Functionality

## Real-Time Price Tracking

### Data Source

Real-time cryptocurrency prices are retrieved from CoinGecko's market chart endpoint, which provides up-to-date prices, market capitalization, and volume data. This data is parsed using Python's requests module in the backend.



Source: https://www.coingecko.com/en/api

### Display

The real-time prices are injected into the index.html template using Flask. The values are refreshed each time the page is loaded, ensuring users see the latest available data. Prices are formatted and styled with CSS to ensure clarity and emphasis. Additional visual cues such as color indicators (e.g., green for gains, red for losses) are used to highlight market trends at a glance. The layout is responsive, allowing the display to adapt well across different screen sizes.

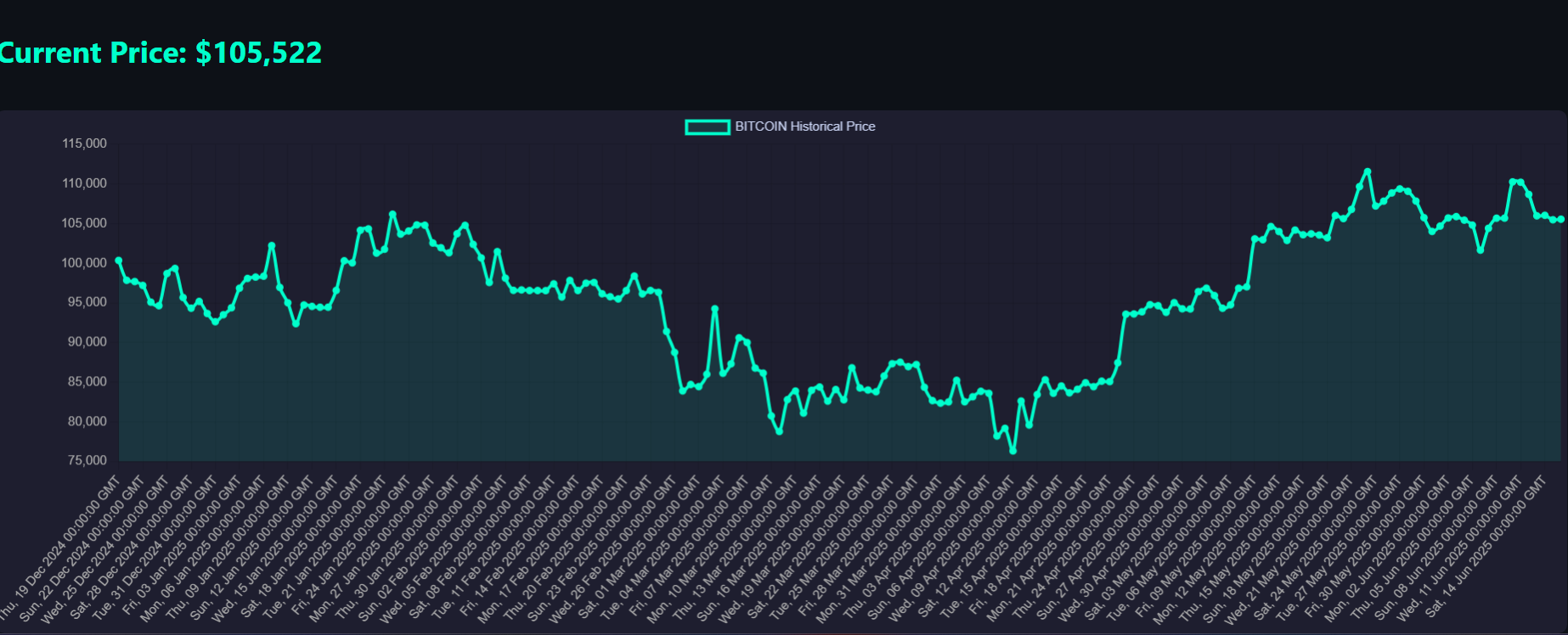
## Historical Trend Visualization

### Data Collection

The init\_data.py script fetches up to one year of historical price data for each selected cryptocurrency (Bitcoin, Ethereum, Dogecoin). Data is downloaded in JSON format and converted into structured CSVs using pandas.

### Visualization

Within the index.html, Chart.js dynamically plots the historical data as line graphs. Each graph is color-coded by coin and includes interactive features such as hover tooltips, zoom, and responsive resizing. This helps users visualize long-term trends and market fluctuations.



## Price Prediction

### Training Data

The XGBoost model is trained using features extracted from the historical CSV files. These features include price time steps, previous values, percentage changes, and moving averages. The data is normalized and prepared using pandas and numpy.

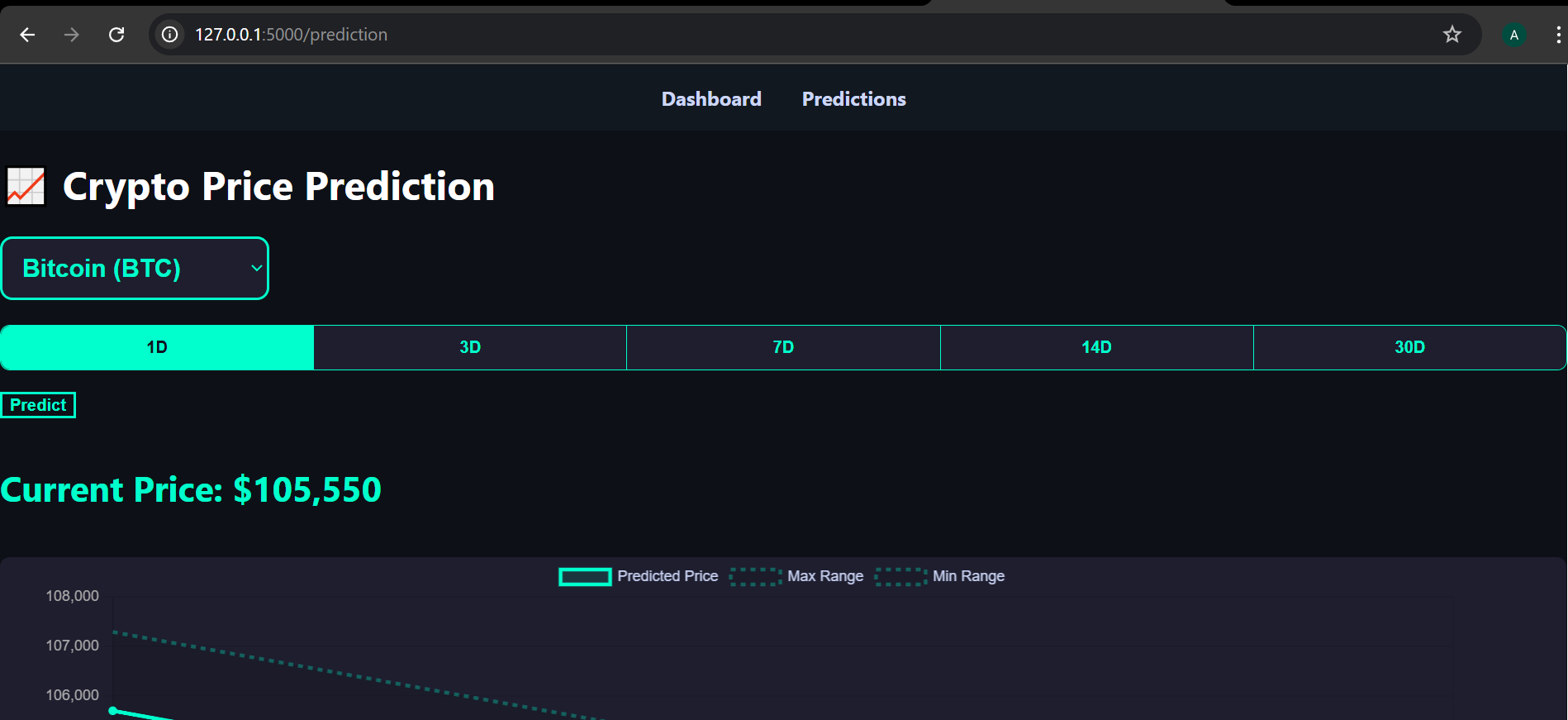
### Prediction Route

When a user accesses prediction.html and selects a cryptocurrency, a POST request is sent to the /predict route in Flask. This triggers logic in utils/model.py which:

* Loads the selected CSV data
* Prepares it with the required features
* Feeds it to the pretrained XGBoost model
* Returns the predicted value to the frontend

### Output

The result is dynamically displayed on the web page without needing a page reload. The prediction includes the forecasted price and may be styled based on confidence or comparison to previous prices.



# Folder and File Structure

To ensure clarity, maintainability, and separation of concerns, the Crypto Tracker with Trend Prediction project is organized into a modular directory structure. Each component is placed according to its functional role in the system, facilitating collaborative development, debugging, and future scalability.

## Root

* app.py: Flask app setup and route definitions.
* init\_data.py: Downloads and formats historical data from CoinGecko API.

## /data

Contains historical CSV data used for training and predictions:

* bitcoin\_1y.csv
* ethereum\_1y.csv
* dogecoin\_1y.csv

## /templates

* base.html: Base layout for template inheritance.
* index.html: Homepage UI with current prices and charts.
* prediction.html: Form interface for selecting coin and submitting prediction requests.

## /static

* style.css: Custom styles for layout, typography, and interactive visuals.

## /utils

* api.py: Includes helper functions for API calls.
* model.py: Contains the logic for loading the model, preprocessing input, and generating predictions.

# How It Works

## Initialization

To ensure local availability of data, the init\_data.py script must be run once before starting the server. This script connects to CoinGecko, requests daily data for the past year, converts it into a DataFrame, and saves it as CSV.

## Running the App

* Start the app using python app.py.
* The server will start on http://127.0.0.1:5000 by default.

## Data Flow

1. User visits the homepage.
2. Backend fetches the latest prices and loads historical data.
3. Chart.js renders this data on the frontend.
4. When the user goes to the prediction page and selects a coin, a request is sent to /predict.
5. Backend processes the coin's CSV, applies the ML model, and returns a prediction.
6. Prediction is injected back into the page for the user to view.

## Installation Requirements

Ensure Python 3.10+ is installed. Required libraries:

| pip install flask requests pandas numpy xgboost scikit-learn |
| --- |

## 

## Installation Requirements

| # Step 1: Initialize historical data python init\_data.py # Step 2: Start Flask server python app.py # Step 3: Open in browser http://127.0.0.1:5000 |
| --- |

# Challenges and Resolutions

## Missing Dependencies

Initially, key modules such as scikit-learn and xgboost were not present in the environment, which led to runtime errors when predictions were attempted. This was resolved by identifying and installing all required packages using pip. A requirements.txt file was also created to ensure smooth setup on other machines and to prevent similar issues during deployment or future development.

## API Rate Limits

Another major challenge encountered was the limitation imposed by CoinGecko’s free-tier public API. CoinGecko enforces rate limiting to prevent abuse, which means frequent requests (e.g., during refresh cycles or iterative development) can quickly exceed the allowable threshold. This results in temporary bans or 429 Too Many Requests errors.

To overcome this, we restructured the system to separate live data and historical data usage. For historical data which remains static for our use case — we developed init\_data.py, a script that fetches one year of historical data in a single call and stores it locally as CSV files. This allows the machine learning model to access data directly from disk, avoiding unnecessary API calls. Additionally, this structure ensures consistent input formatting for training and evaluation.

## Model Integration

Integrating machine learning with Flask required careful structuring. The model had to be lightweight and efficient, and input features had to be aligned consistently with training features. Serialization and deserialization of models were handled using joblib.

# Conclusion

## In-Depth Summary

The Crypto Tracker with Trend Prediction project represents a complete integration of real-time data pipelines, historical data analysis, interactive data visualization, and machine learning forecasting all built within a unified Flask-based web application. From concept to execution, it explores key challenges faced in real-world data science projects: API data management, time series modeling, server-client communication, and user interface design.

By leveraging the CoinGecko API, the project accesses reliable and up-to-date cryptocurrency information. This data is then processed using pandas and numpy, and visualized with Chart.js to provide an intuitive view of price history. The system further demonstrates applied machine learning using XGBoost models for short-term forecasting, integrating prediction results seamlessly into the UI via dynamic routing and form submissions.

The modular architecture separating data fetching, visualization, modeling, and interaction makes this project scalable and easy to maintain. Furthermore, the use of open-source technologies ensures reproducibility, transparency, and adaptability for future improvements.

More than just a functional app, this project showcases the capability of a small, well-structured codebase to deliver real-time insights using modern Python tooling. It bridges gaps between backend programming, frontend presentation, and AI-powered analytics.

## Future Improvements

* Expand support to additional cryptocurrencies and fiat currencies.
* Add interactive time range selection for historical trends.
* Enable periodic automatic updates using background jobs.
* Store historical user predictions in a database for analytics.
* Incorporate LSTM or other deep learning models for improved time series prediction.
* Deploy on cloud platforms for public access (Render, Vercel, etc.).