BACK2SERIES: Designing a Backtesting Framework for Time Series Forecasting into a RESTful API

Eduardo Olinto[1], **Augusto Baffa**[1], **Juliana Alves Pereira**[1]

[1]Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rio de Janeiro, RJ – Brasil

duolinto@gmail.com, {abaffa, juliana}@inf.puc-rio.br

Abstract. Research Context: *Time series forecasting plays a crucial role in economics, finance, and other domains where accurately anticipating future outcomes is essential for informed decision-making. However, evaluating time series regression models requires specific methods to preserve temporal dependencies and ensure realistic assessments.* ***Scientific and/or Practical Problem:*** *Common validation approaches, such as random cross-validation, ignore time order, generating look-ahead bias and unreliable results. Existing libraries like skforecast and sktime provide partial solutions but lack comprehensive backtesting features, standardized procedures, and user-friendly interfaces for non-programmers.* ***Proposed Solution and/or Analysis:*** *This study introduces* BACK2SERIES*, a modular RESTful API for time series regression backtesting. It supports multiple scikit-learn estimators, expanding and rolling windows, periodic hyperparameter tuning, and proper data standardization to prevent information leakage. Its API architecture allows integration with Large Language Models (LLMs) for automated workflows.* ***Related IS Theory:*** *This work is grounded in Socio-technical Theory, as it integrates technical automation with human interaction, balancing system efficiency and user accessibility. It also draws on Process Virtualization Theory, enabling remote, automated, and reproducible workflows that make time series forecasting more accessible to both technical and non-technical users.* ***Research Method:*** *Following an exploratory case study methodology, the system was implemented in Python using FastAPI, Pandas, and scikit-learn. A case study was conducted forecasting the USD/BRL exchange rate using other emerging market currencies to validate the framework's reproducibility and automation capabilities.* ***Summary of Results:*** *The API produced unbiased evaluations, automated hyperparameter tuning, and consistent model comparisons. Tests of LLM interaction confirmed that structured prompts enable complete, valid API calls.* ***Contributions and Impact to IS Area:*** BACK2SERIES *advances the Information Systems field by bridging rigorous time series evaluation with accessible deployment. It enables reproducible, automated, and explainable forecasting workflows, supporting both technical and non-technical users in data-driven decision-making.*

1. Introduction

Supervised machine learning is widely used by data scientists to uncover relationships between variables and build models that can predict future outcomes. Regression

techniques are powerful for such tasks, supporting both explanatory analysis and predictive modeling, especially in the context of time series data [James et al. 2023]. [Hyndman and Athanasopoulos 2021] defines a time series as *"anything that is observed sequentially over time"*. The study of time series is crucial to understand historical patterns of temporal data and anticipate future dynamics. In domains such as economics, energy, logistics, and healthcare, accurate forecasting is critical for anticipating risks, guiding policy, and improving strategic decision-making. However, time series modeling presents unique challenges [Hyndman and Athanasopoulos 2021].

The sequential nature of time series data requires specific care in model design and evaluation. Incorporating future observations — even indirectly — into model training introduces the well-known *look-ahead bias*, a methodological flaw that undermines the validity of the results. Similarly, applying traditional validation approaches such as random *K-Fold* cross-validation ignores temporal order and can lead to *overfitting*, yielding predictions that fail to generalize. Therefore, when assessing how different models would have performed in the past, a process commonly referred to as *backtests*, it is crucial to adopt evaluation strategies that properly account for temporal dependencies [Bergmeir et al. 2018].

Backtesting is a fundamental task in forecasting. Its main objective is to identify models that are both accurate and robust, while minimizing the risk of look-ahead bias [Kenton and Kvilhaug 2023]. To achieve this, models must be evaluated as if they were applied in the past, using only the data available at each point in time, a step that is often overlooked in practice. Moreover, regression models such as Random Forest and Lasso are highly sensitive to their hyperparameter configurations. Hyperparameters define the behavior of a model (*e.g.*, the number of trees in a Random Forest) and selecting the optimal combination is known as tuning. Hyperparameter tuning is a cornerstone of the backtesting process. Among the various approaches, Grid Search Cross-Validation (GSCV) is considered one of the most reliable, as it systematically explores all possible combinations of hyperparameters [James et al. 2023]. Importantly, tuning should not be treated as a one-off task: ideally, it must be repeated periodically so that models can adapt to evolving data distributions over time [Zhan et al. 2018]. Another crucial aspect of valid backtesting is data scaling, also referred to as standardization. If scaling methods, such as *z-scores*, are computed over the full dataset and then applied retrospectively, they inadvertently incorporate future information, introducing a clear case of look-ahead bias [Gupta and Hewett 2019]. To ensure valid results, standardization must always be performed within the proper temporal context, using only past data available up to each evaluation point.

A few tools offer resources for developers to perform backtests of time series regression models directly in code. Among them, `Scikit-learn` stands out as the most widely used Python library for machine learning models [Pedregosa et al. 2025]. However, its native functionality offers limited support for the specific requirements of time series experiments. To address this gap, libraries such as `skforecast` and `sktime` provide specialized functionalities and extensive documentation tailored to time series tasks [Amat Rodrigo and Escobar Ortiz 2025, Löning et al. 2019]. Yet, both libraries require advanced technical knowledge and considerable adaptation to implement realistic backtesting workflows, which limits their accessibility. Moreover, several

important features previously discussed are unavailable, compromising the validity and reproducibility of experiments. These requirements include, for instance: (i) preserving strict temporal ordering during model training and evaluation; (ii) periodically re-tuning hyperparameters to account for non-stationarity; (iii) applying data standardization only within each training window to avoid information leakage; and (iv) recording the temporal evolution of selected hyperparameters and model configurations for post-hoc analysis. In practice, widely used libraries such as skforecast and sktime do not natively support all these requirements simultaneously. In particular, they lack built-in mechanisms for periodic hyperparameter analysis, longitudinal tracking of model configurations, and API-based automation, which limits reproducibility and accessibility in real-world workflows.

This work introduces a new *backtesting* framework called BACK2SERIES for time series regression models, encapsulated in a modular REST API with support for several scikit-learn estimators. The framework allows users to input datasets, configure validation windows, perform per-window hyper-parameter tuning, mitigate temporal bias, and generate forecasts across different horizons. Encapsulating the framework as a REST API simplifies usage, and facilitates integration with modern Artificial Intelligence (AI) tools. Large Language Models (LLMs) can interpret user intent and interact with the API autonomously, removing the need for users to write code or learn new scripting languages. The tool was designed for accessibility; without requiring repeated code adaptations. Looking ahead, the HTTP interface offers a straightforward pathway for integration with web-based dashboards, paving the way for a no-code platform that enables analysts without programming expertise to conduct time series modeling.

BACK2SERIES addresses key challenges in time series forecasting, including temporal dependencies, look-ahead bias, and appropriate validation schemes. The main contributions of BACK2SERIES can be summarized as follows:

- The framework natively supports multiple regression models from the scikit-learn ecosystem.
- The framework implements both expanding and rolling window validation, ensuring realistic evaluation of models over time.
- The framework performs hyperparameter optimization within each validation window, reflecting the dynamic nature of time series data and reducing risks of model miscalibration. Proper standardization methods are applied within each evaluation window to prevent information leakage, a common source of invalid results in time series experiments.
- By providing the framework through a modular REST API, we simplify access for users, lower the entry barrier for non-programmers, and enable interoperability with other systems and tools.

Target Audience. BACK2SERIES is designed to support a broad range of professionals and researchers involved in predictive modeling and time series analysis. Its features address the specific needs of users who require rigorous temporal validation and transparent model comparison processes. Data scientists and business analysts can benefit from BACK2SERIES's ability to validate models under strict temporal constraints and systematically compare different predictive techniques under realistic scenarios. The automated backtesting workflow enables teams to assess the robustness and consistency of

models over multiple rebalancing windows or forecast horizons, leading to more informed decision. Researchers and students can leverage BACK2SERIES as an educational platform for exploring how different validation strategies, feature selection methods, or hyperparameter tuning approaches affect model performance in temporal contexts.

Main Contributions. BACK2SERIES contributes to the virtualization of analytical processes by encapsulating complex forecasting workflows into standardized, auditable, and reproducible services. This enables integration into organizational decision-support systems, data pipelines, and compliance-oriented environments where transparency and traceability are critical.

2. Background and Related Works

In this section, we present an in-depth overview of backtesting principles, and we discuss existing tools and their shortcomings in supporting time series regression workflows.

2.1. Backtesting

Backtesting is the process of evaluating how a predictive model or strategy would have performed using historical data, thereby simulating real-time decision-making in the past. The core principle is that models should be trained exclusively on information available up to a given point in time and then tested on subsequent, unseen observations. This method mirrors a step-wise temporal prediction, ensuring that no future information is inadvertently used, thereby preventing look-ahead bias.

According to Investopedia, backtesting *"assesses the viability of a trading strategy by discovering how it would have played out retrospectively using historical data"* [Kenton and Kvilhaug 2023]. It is widely recognized as a structured protocol for evaluating robustness and analyzing model behavior under different situations [Hyndman and Athanasopoulos 2021, Bergmeir et al. 2018]. The design of a backtesting procedure typically hinges on two key methodological decisions: *(i)* the choice of a window structure used for training and testing, and *(ii)* the strategy for hyperparameter optimization. Each of these elements is detailed in the following sections.

2.1.1. Window Structure

Mainly, there are two primary approaches to replicating the past performance of a model: the rolling (or sliding) window and the expanding window [Hyndman and Athanasopoulos 2021]. Figure 1 illustrates the difference between these two frameworks over time [Bergmeir et al. 2018, Hyndman and Athanasopoulos 2021].

A rolling window maintains a fixed-length training set, such as the most recent two years of data. This strategy ensures that the model reflects more current conditions while keeping computational cost bounded. In contrast, an expanding window begins with a small initial sample and incrementally grows over time, incorporating all historical data available up to the current point. The choice between these approaches depends largely on the application domain. If the data exhibit stable, long-term patterns, expanding windows may yield better performance by leveraging the entire historical record. Conversely,

in dynamic contexts such as financial markets, rolling windows often provide greater adaptability to regime shifts and structural changes.

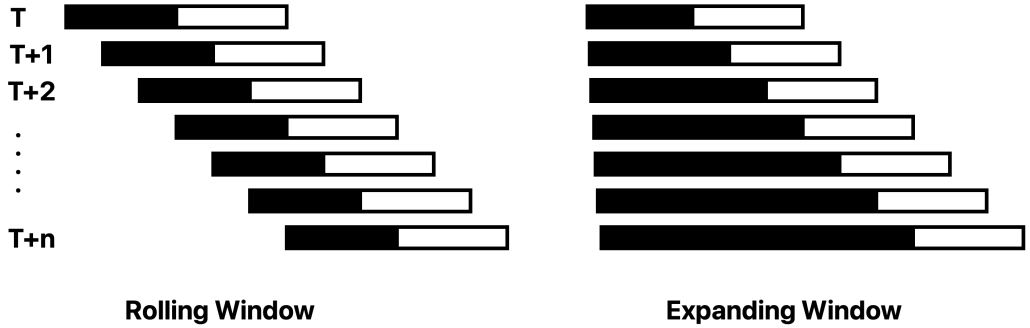

Figure 1. Window Structures: training data in black and testing data in white.

As Figure 1 shows, each method leverages data from different time periods, which may introduce bias in distinct ways. Therefore, careful consideration of the window structure is crucial for producing meaningful backtesting results. Modern machine learning libraries such as `scikit-learn` [Pedregosa et al. 2025] and `sktime` [Löning et al. 2019] offer built-in utilities to implement both rolling and expanding window backtesting in time series analysis.

2.1.2. Hyperparameter Optimization

At each iteration of a backtest (whether using a rolling or expanding window), it can be advisable to re-optimize the model's hyperparameters. This can be achieved through grid search, employing either cross-validation (GSCV) or a dedicated validation split within the window [James et al. 2023]. Ideally, tuning should be repeated whenever the window advances (*e.g.*, at each monthly or quarterly rebalance), ensuring that the model remains adapted to evolving data patterns [Zhan et al. 2018].

However, GSCV is computationally expensive, especially when performed every step. To balance accuracy and efficiency, practitioners often reduce the tuning frequency (e.g., every N steps) or adopt more efficient search strategies such as random search or Bayesian optimization, both of which are widely supported in academic research and practice [James et al. 2023].

2.2. Existing Tools

Several tools and libraries have been developed to support predictive modeling workflows for the integration of machine learning models into time series forecasting tasks. In this section, we review how these tools address the concepts discussed earlier and highlight the ways in which they differ from our proposed solution, BACK2SERIES. Table 1 provides a comparative overview, including the framework introduced in this paper.

`Skforecast` is an open-source Python library for time series forecasting using machine learning regressors compatible with the scikit-learn API. It provides tools to build models supporting single and multi-series forecasting, with optional exogenous variables and probabilistic outputs. Its design focuses on simplifying the use of general-purpose regressors in time series contexts [Amat Rodrigo and Escobar Ortiz 2025].

Table 1. Comparison between tools for time series regression backtesting

Feature	skforecast	sktime	BACK2SERIES API
Time series regression backtesting	Yes	Yes	Yes
Compatibility with `scikit-learn`	Yes	Yes	Yes
Hyperparameter tuning	Yes (not periodic)	No	Yes (periodic)
Periodic standardization	No	No	Yes
Hyperparameter periodic analysis	No	No	Yes
API structure	No	No	Yes

In terms of backtesting, `Skforecast` includes dedicated methods to evaluate models using historical data and with stepwise validation. It supports both expanding and rolling windows, allowing users to simulate the re-training and evaluation of models over time. It also offers hyperparameter tuning, including grid search, to find the best model configurations [Amat Rodrigo and Escobar Ortiz 2025].

`Sktime` is also an open-source Python library designed to provide a unified interface for machine learning tasks involving time series. Its architecture is modular and object-oriented, and it follows interface conventions similar to scikit-learn. The framework has standard Python tools (*e.g.*, pandas, NumPy), and it is aimed at facilitating educational and research use as well as practical applications [Löning et al. 2019].

Regarding backtesting, `Sktime` includes built-in functionality for evaluating forecasting models using temporally consistent strategies. It allows for the specification of forecasting horizons and resampling strategies, enabling the implementation of both expanding and rolling windows via its temporal train-test split and backtesting utilities. These mechanisms integrate with `sktime`'s pipeline structure, supporting automated workflows for model tuning and validation.

Both tools, `Skforecast` and `Sktime` are built on top of the `scikit-learn` framework, a widely used Python library providing regressors, classifiers and cross-validation [Amat Rodrigo and Escobar Ortiz 2025].

These tools allow users to perform backtesting once, after defining a model and a hyperparameter grid. While useful, this approach is limited: data scientists often need to evaluate how models would have performed over specific periods and determine the optimal hyperparameters for each period. Consequently, both libraries lack a backtesting cycle framework, in which such longitudinal analysis is automated. Another critical limitation is data standardization. In many datasets, scaling must be performed within the backtesting cycle, as applying transformations on the full dataset can introduce *look-ahead* bias.

Usability is also a key concern. Both `skforecast` and `sktime` require proficiency in Python, which can hinder adoption by economists, business analysts and other professionals who prefer simplified interfaces. This limitation is particularly relevant for experiments involving multiple models simultaneously, where computational efficiency and parallelization are important.

No-code platforms such as Orange and KNIME provide interfaces for machine learning workflows, with support for simultaneous execution and evaluation of multiple

models. However, these platforms offer limited support for the specific requirements of time series modeling and often struggle with large datasets [Demšar, Janez et al. 2025, Berthold, Michael R. et al. 2025]. Furthermore, they lack the flexibility that software delivers in forms such as APIs.

In this paper, we introduce BACK2SERIES, a tool grounded in the core principles of backtesting with additional emphasis on maintaining temporal validity. BACK2SERIES is designed to enable data scientists to effectively analyze and tune models without requiring programming skills. By encapsulating the framework as a REST API, it simplifies user interaction by structuring the backtesting flow into standardized requests. APIs offer an efficient and structured interface for connecting with LLMs as well, as they enable interactions through standardized schemas such as REST and JSON without requiring local code execution or programming expertise. Song et al. [Song et al. 2023] demonstrates this advantage by showing that LLMs can plan, select, and execute complex API flows while interpreting structured responses and adapting dynamically to context. BACK2SERIES aims to enable this as well.

3. BACK2SERIES

This paper introduces BACK2SERIES, a modular and extensible RESTful API designed to automate the end-to-end backtesting of time series regression models. The framework was implemented in Python using the FastAPI framework, the Pandas data manipulation library, and the scikit-learn ecosystem [Pedregosa et al. 2025].

BACK2SERIES enables users to upload datasets and execute backtesting workflows in which strict temporal constraints are explicitly enforced. The framework supports the systematic analysis of model behavior over time by recording historical configurations, hyperparameter choices, and evaluation results across successive training windows. In addition, the API exposes a structured and standardized set of endpoints that facilitate automation, reproducibility, and integration with external systems.

3.1. Research Method

This study adopts an exploratory case study methodology to evaluate the proposed BACK2SERIES framework. Rather than statistically comparing forecasting performance against competing libraries, the objective is to assess whether the API supports realistic, reproducible, and automated backtesting workflows under strict temporal constraints.

The case study was designed to demonstrate the complete end-to-end use of the system, including dataset ingestion, data transformation, model configuration, backtesting execution, and result inspection. Accordingly, the methodological focus lies on technical feasibility, workflow validity, and reproducibility, rather than on predictive optimality.

3.2. Features

BACK2SERIES was structured into three main features: (1) dataset management, (2) data transformation, and (3) the backtesting engine. Figure 2 illustrates how the system was architected in order to provide the main functionalities to the user via API endpoints. Each component has a designated function, which improves the organization and accessibility to the user. In the workflow, the user must first upload a dataset to the API using the dataset endpoint, which then gets stored in the database. If needed, the user can choose

a transformation to apply to the data stored and then finally, execute the backtest in the transformed dataset, with the variables they choose. The processed data is then always returned as a JSON to the user. Notably, each module of the API can access the stored data but the user only interfaces with them via the endpoints.

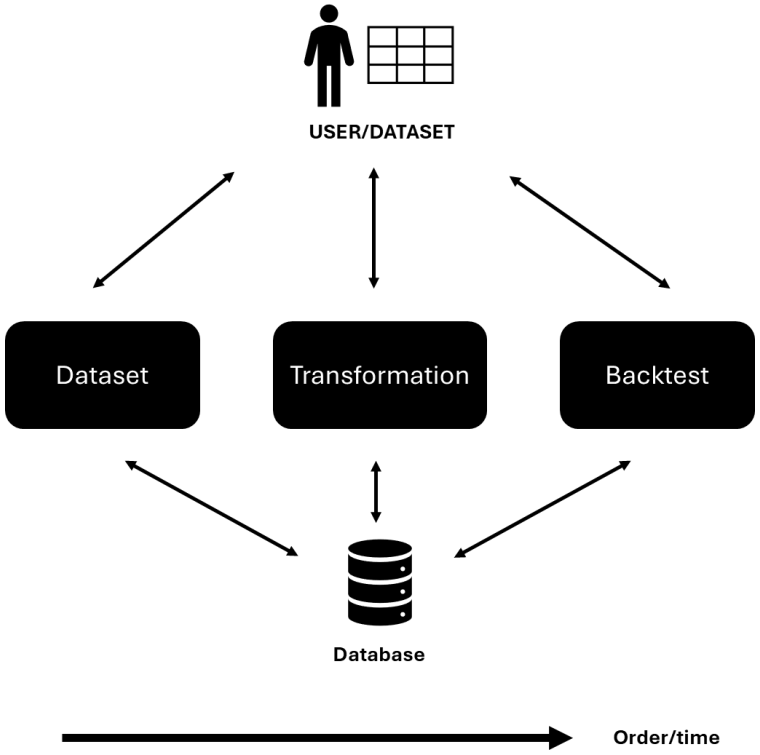

Figure 2. BACK2SERIES Design

(1) Dataset. This component allows users to interact with the datasets they wish to model. Users are able to upload '.csv' files containing time series data. This corresponds to POST requests to the API server, which maps to a dedicated folder within the GitHub repository. Users can issue GET requests to preview basic dataset information (*e.g.*, data samples), using a unique identifier.

(2) Transformation. Users can apply simple data transformations using commonly available Pandas functions:

- *Percentage Change*: calculates relative changes over consecutive periods. Useful for understanding short- and long-term trends.
- *Difference*: computes the difference between consecutive periods. Essential for making series stationary.
- *Moving Average*: smooths data over a rolling window. Helps reduce short-term noise and reveal underlying patterns.
- *Resample*: aggregates higher-frequency data into lower-frequency intervals. Useful for aligning series with different granularities.
- *Cumulative Sum*: converts flow data into stock format, effectively the inverse of the difference transformation.

(3) Backtest. This is the core component. It allows users to perform backtests by:

- Identifying the dataset.
- Selecting the model to use: Linear Regression, Ridge, Lasso, Random Forest Regressor, and Gradient Boosting Regressor.
- Identifying the set of hyperparameters values to be tested.
- Defining the tuning frequency of running the GSCV to find the optimal hyperparameters.
- Configuring the window type (rolling or expanding) and size (if rolling)
- Configuring the forecast horizon.
- Defining if the data should be Standardized in each iteration.
- Defining if the data should be parallelized.
- Defining the set of metrics to consider (*e.g.* RMSE, MAE, MAPE, etc).

BACK2SERIES builds a core temporal modeling pipeline capable of receiving tabular data, performing preprocessing, applying feature selection, and feeding different regression algorithms. For each time step, the algorithm selects a training window, which can be either expanding (gradually including more data) or rolling (a fixed-size window moving through time). If enabled, standardization is applied to the features and target variables within the training and test sets at the selected window respecting the sequential structure of the data to prevent information leakage.

At specified intervals (according to the tuning frequency), the algorithm performs GSCV to optimize the model's hyperparameters based on the current training data. The selected model is then fitted, and a prediction is made for a future time point defined by the forecast horizon. If necessary, meaning if the data was standardized, predictions are inverse-transformed to return to the original data scale. For each step, the algorithm records and returns forecasts with full traceability. After iterating through the data, it returns the evaluation metrics such as RMSE and MAE, the model used, and the hyperparameters that produced the result. This provides a comprehensive overview of the model's performance over time and the effectiveness of the hyperparameter tuning process.

In the end, BACK2SERIES returns: *(i)* the time series of predictions generated by the model over time, *(ii)* the requested error metrics, and *(iii)* a trace of parameters used throughout the evaluation.

3.3. Architecture and Module Organization

This section provides a technical overview of the internal architecture of BACK2SERIES, focusing on how the Python modules are structured and how they interact to support each endpoint's functionality.

BACK2SERIES follows a modular design, structured around the FastAPI framework. FastAPI [Ramírez 2025] is a widely used Python framework for building web services. It leverages asynchronous routes with Starlette and the Uvicorn server, and utilizes Python's type hints along with Pydantic for automatic data validation and serialization. Each route automatically generates Swagger documentation, which simplifies testing and inspection. Its dependency injection system promotes modularity—such as separating authentication, database access, or messaging queues—making it well-suited for microservices and testable architectures.

FastAPI automatic integration with Swagger UI allows users to access a visual and interactive documentation interface by navigating to `http://127.0.0.1:8000/docs` in their web browser [Ramírez 2025]. The interface is intended primarily to facilitate testing during development, especially when debugging request formats or validating payloads. It provides:

- A list of all available endpoints (*e.g.*, `/datasets`, `/transformations`, `/backtests`).
- Input fields to manually enter parameters and upload files.
- Live execution of requests with instant feedback on the returned JSON.
- Descriptions of inputs, outputs, and response schemas derived automatically from the Pydantic models.

BACK2SERIES is organized into the following directories and components:

- `main.py`: entry point of the API application. It registers all routes and launches the Uvicorn server.
- `api/`: a folder that contains route definitions for each endpoint.
- `api/v1/backtest.py`: file containing the endpoint for the backtest structure.
- `api/v1/datasets.py`: file containing the endpoints for the dataset structure.
- `api/v1/transformations.py`: file containing the endpoints for the transformation structure.
- `services/`: houses core Python services that implement the logic behind each endpoint. The core of the system is the `backtest_service.py` which encapsulates all the functionality related to model evaluation and backtesting.
- `schemas/`: defines Pydantic models used for request and response validation, ensuring type safety and clean API design.
- `uploads/`: a local storage area where uploaded CSV files are saved before being read into memory.

Modules communicate by passing structured data and function calls. For example, a route in `backtests.py` will parse query parameters, validate them using a schema, and then invoke the corresponding service method from `backtest_service.py`, which handles model training, validation, and prediction.

Each API endpoint follows a similar pattern of operation:

1. *Request Reception*: The user sends a POST or GET request to a specific route (*e.g.*, `/backtests`, `/datasets`).
2. *Validation*: Parameters are validated and parsed using Pydantic schemas or custom functions.
3. *Service Invocation*: The request is delegated to a service module that contains the business logic.
4. *Processing*: The service interacts with internal modules, performs computations, and returns structured outputs.
5. *Response*: The result is serialized and returned to the user in JSON format, with optional metadata (*e.g.*, model metrics, execution time).

(1) Dataset Endpoint. When the `datasets` endpoint is used, the user uploads a CSV file via a POST request. The route stores the file in the `datasets/` directory, assigning it a unique identifier (UUID) as its *filename*. Subsequent GET requests to preview the dataset use this identifier to locate and read the file into memory using Pandas. In total, there are 3 endpoints for `datasets`:

- `/datasets`: POST request for uploading a CSV file. The CSV file **must** have a date column ideally called 'date'.
- `/datasets/info`: GET request for general metadata of the dataset, provided an id. Fields such as id, name, columns, number of rows and updated at time are returned in JSON format.
- `/datasets/sample`: GET request for a preview sample of the dataset, provided an id and the number of rows to be returned.

(2) Transformation Endpoint. The transformation module offers a set of endpoints that apply common data preprocessing techniques to specific columns of a dataset. Each endpoint is accessed through a `POST` request and returns a new CSV file, along with a preview of the transformed data. Next, we present a brief description of each available transformation:

- `/transform/pct_change`: Accepts a dataset identifier, a column name, and the number of periods. Computes the percentage change between the current and previous values. This transformation is often used to calculate returns or growth rates. For instance, a user wishing to compute Year-over-Year variation in a monthly dataset would set `periods=12`.
- `/transform/diff`: Accepts a dataset identifier, a column name, and the number of periods. Calculates the absolute difference between values separated by the specified number of periods.
- `/transform/rolling_mean`: Accepts a dataset identifier, a column name, and a rolling window size. Computes the moving average of the selected column over the defined window.
- `/transform/resample`: Accepts a dataset identifier, a column name, a frequency string (*e.g.*, "M" for monthly), and an aggregation method (*e.g.*, mean, sum, ffill, bfill). Aggregates the series to a lower frequency using the specified method.
- `/transform/cumsum`: Accepts a dataset identifier and a column name. It computes the cumulative sum of the selected column. Often used to transform flow variables into stock variables.

(3) Backtest Endpoint. The `/backtests` endpoint is the most complex. Upon receiving a POST request, it validates the input (*e.g.*, model type, hyperparameters, window type), loads the dataset, and prepares the training and test sets. It then invokes the backtesting service, which executes model training and evaluation either sequentially or in parallel, depending on the user's request. The results are returned as a dictionary of error metrics and prediction traces.

3.4. Extensibility and Customization

The modular structure of the system allows for easy extension. New models can be added to the backtesting engine by updating the `get_model_instance()` and `get_model_class()` functions. Likewise, new transformation functions can be integrated with minimal changes to the API layer.

The separation between API logic (routes), business logic (services), and data schema (Pydantic models) ensures that each component is independently testable and easily modifiable. This architecture also enables integration with cloud storage services or distributed computing tools.

4. Case Study

The results presented in this section are structured to highlight both the modeling capabilities and the practical workflow. We demonstrate how BACK2SERIES streamlines dataset upload, transformation, model configuration, backtesting, and results retrieval, facilitating automation and reproducibility. Hyperparameter choices and forecast results are automatically logged for each period, allowing for detailed post-hoc analysis without manual intervention.

While standard tools such as `skforecast` and `sktime` provide robust solutions for backtesting and hyperparameter tuning in time series regression, their design supports single-shot evaluations (see Section 2.2). They lack integrated mechanisms for periodic model selection, dynamic hyperparameter analysis, and API-based automation. Thus, running the same modeling workflow in `skforecast` or `sktime` would require writing and maintaining custom scripts for each model, manually managing data transformations, and manually aggregating the results for each backtest iteration. Neither library natively supports recording the evolution of selected hyperparameters over time, nor do they offer API-based endpoints for automated integration with data pipelines.

Next, we use a real-world modeling case study to demonstrate how BACK2SERIES facilitates reproducible forecasting experiments, as well as the temporal tracking of model configurations, aspects that are difficult or labor-intensive to implement with existing tools. All artifacts are available in our supplementary material (see Section 6)

4.1. Problem Setup

Emerging market (EM) currencies often share structural similarities. They tend to be influenced by global risk sentiment, commodity price cycles, capital flows, and monetary policy from developed economies, particularly the United States. As a result, their exchange rates often exhibit comovement patterns — both in levels and in returns — especially during periods of market stress or synchronized macroeconomic trends. Numerous studies have documented these relationships, suggesting that movements in one EM currency may help predict near-term shifts in others [European Central Bank 2019].

Motivated by this intuition, we investigate whether the weekly levels of four selected EM currencies — the Mexican peso (MXN), Chilean peso (CLP), Colombian peso (COP), and South African rand (ZAR) — contain predictive information about the behavior of the Brazilian real one week ahead. While not all currencies are from

Latin America, the group was chosen for their similar macroeconomic profiles: current account vulnerabilities, exposure to commodity cycles, and relatively high interest rate environments.

Goal. To forecast the weekly closing level of the Brazilian real (BRL) against the U.S. dollar using the historical behavior of other emerging market currencies.

4.2. Data Collection

The exchange rate data used in this experiment consists of weekly closing levels, extracted from Bloomberg's historical time series for each currency. All series represent the local currency per U.S. dollar ("Curncy" suffix in Bloomberg tickers), and were selected to ensure consistency in timing and liquidity. They were first loaded into the dataset endpoint.

The original dataset was then manipulated using the API's transformation endpoint. The initial step was to apply a `resample("W")` operation on each currency's time series, effectively converting them into weekly frequency with consistent timestamps. This process was performed using the `/transform/resample` endpoint of the API.

This combined dataset, starting in the year 2000 and going up to July 2025, was uploaded to the API through the `/datasets` endpoint. In the repository it is the CSV file named as `ems_ccys_resampled.csv`. Figure 3 shows the time series of the selected currencies from December 2014 onwards, with COP and CLP in the right axis and MXN, ZAR and BRL in the left axis. As we can see, in periods such as March 2020, all currencies exhibit a devaluation against the Dollar, something also seen last year (2024). This experiment will attempt to understand whether these periods are frequent and therefore can help predict the direction of one of these currencies (BRL), using the others.

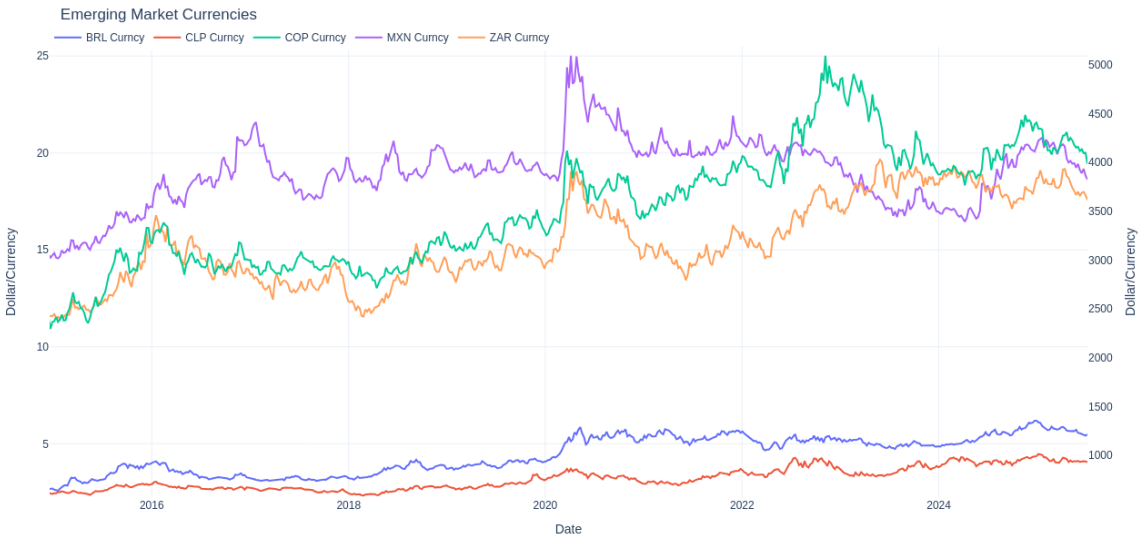

Figure 3. Time Series of Emerging Market Currencies: BRL, MXN, CLP, COP and ZAR (Since December 2014)

4.3. Backtest Configuration

To test the predictive power of these EM currencies over BRL, two models were applied using the `/backtests` endpoint:

- *Random Forest Regressor*: A flexible, non-parametric model known for handling non-linear interactions and capturing hidden patterns in the data. The following hyperparameter grid was used:
  - `n_estimators`: [100, 300]
  - `max_depth`: [None, 10]
  - `min_samples_split`: [2, 5]
  - `min_samples_leaf`: [1, 2]
  - `max_features`: ['sqrt', 'log2', None]

- *Lasso Regression*: A linear model with L1 regularization, capable of feature selection and well-suited for sparse predictive signals. The hyperparameter grid for Lasso included:
  - `alpha`: [0.0001, 0.001, 0.01, 0.1]

It is important to note that these hyperparameter values were chosen arbitrarily since the main objective of this study is not to evaluate each model but offer a practical case study on how BACK2SERIES offers a backtesting workflow for time series forecasting.

Both models used the same backtesting strategy:

- *Target*: BRL closing level one week ahead.
- *Features*: MXN, ZAR, CLP, COP weekly closing levels at time t.
- *Initial window*: 4 years.
- *Backtest type*: Rolling window.
- *Window size*: 4 years.
- *Tuning frequency*: 4 years.
- *Standardize*: True.

No percentage transformation or differencing was applied in this case study. The model worked directly with currency levels, allowing it to learn potential cointegration or mean-reversion patterns implicitly. Each of these parameters was formatted into a URL for the API call to be executed (see detail in our supplementary material – Section 6). The resulting response of the `/backtest` endpoint allows the user to analyze how both the predictions and the true level of the BRL evolved through each point in time. Furthermore, the hyperparameters chosen by GSCV are also returned in a temporal basis, to provide further analysis on how the model was trained throughout time.

In the next section, we discuss the results of this modeling exercise, including the predicted and actual values, performance metrics such as RMSE, and visual analysis of errors. The aim is not to optimize forecasting performance at all costs, but rather to validate whether the developed API supports realistic, reproducible workflows for time series modeling.

Table 2. Output API — Lasso

Date	y_pred	y_true	Hyperparams
2008-02-10	1.733	2.3733	alpha: 0.0001
2008-02-17	1.755	2.3059	alpha: 0.0001
2008-02-24	1.7867	2.3019	alpha: 0.0001
2008-03-02	1.863	2.2918	alpha: 0.0001
2008-03-09	1.977	2.2832	alpha: 0.0001

Table 3. Output API — Random Forest

Date	y_pred	y_true	Hyperparams
2008-02-10	1.7534	2.3733	{max_depth: None, max_features: None ..
2008-02-17	1.7075	2.3059	{max_depth: None, max_features: None ..
2008-02-24	1.6907	2.3019	{max_depth: None, max_features: None ..
2008-03-02	1.6834	2.2918	{max_depth: None, max_features: None ..
2008-03-09	1.7119	2.2832	{max_depth: None, max_features: None ..

4.4. Backtest Output

Tables 2 and 3 illustrate how BACK2SERIES returns the results from a backtesting experiment for the Lasso and the Random Forest models, respectively. For each week in the backtest, the API outputs the predicted (`y_pred`) and true (`y_true`) values of the target variable, along with the hyperparameters selected at that iteration.

To understand how the model adapts over time, we examine the most frequently selected hyperparameters in three distinct periods. The API performs hyperparameter optimization at fixed intervals, and the configuration most often selected in each period is summarized in Tables 4 and Table 5 for Lasso and Random Forest, respectively. For the Lasso model, the table shows a pattern for different 7-year periods.

In addition to the numerical evaluation metrics, it is important to analyze the time series behavior of each model's predictions compared to the actual USD/BRL exchange rate. Figures 4 and 5 display the forecasted and actual levels over time for the Lasso and Random Forest models, respectively.

4.5. Result Analysis

The performance summary shown in Table 6 highlights the overall results of the two modeling strategies. Although both models arrive at a remarkably similar final prediction—closely aligned with the actual USD/BRL level—their learning behaviors differ significantly.

The Lasso model slightly outperforms the Random Forest in terms of both RMSE and MAE, reflecting a more stable and parsimonious structure that avoids overfitting.

Table 4. Most frequent Lasso hyperparameters by period

Period	Most Frequent Hyperparams
2008–2013	`alpha: 0.0001`
2013–2019	`alpha: 0.0001`
2019–2025	`alpha: 0.01`

Table 5. Most frequent Random Forest hyperparameters by period

Period	Most Frequent Hyperparams
2008–2013	`n_estimators: 100, min_samples_split: 2, min_samples_leaf: 2`
2013–2019	`n_estimators: 100, min_samples_split: 2, min_samples_leaf: 2`

Figure 4. Lasso forecast vs. actual USD/BRL level

However, this comes at the cost of predictive variance, with Lasso often failing to track short-term dynamics (see Figure 4). In contrast, the Random Forest captures local variations more effectively, generating forecasts with greater volatility. While its error metrics are marginally worse, its flexibility may offer advantages in forecasting highly nonlinear or regime-shifting macroeconomic environments.

Overall, the execution of both models demonstrates how BACK2SERIES enables robust comparative evaluation through a valid backtesting workflow and customizable model configurations.

4.6. Evaluation Using LLMs

To further evaluate the extensibility of BACK2SERIES, an additional experiment was conducted to assess how LLMs interact with the API. The objective was to test whether LLMs could autonomously generate valid requests for the `/backtests` endpoint and correctly handle temporal and structural parameters within the request body.

Three levels of prompt specificity were tested:

1. **Generic prompt** – A short and intuitive instruction without technical context: *"Generate a JSON to forecast BRL using MXN, CLP, COP, and ZAR with the Lasso model and a rolling window of four years."* This version omitted the API schema and parameter details, simulating a user with limited technical knowledge.
2. **Specific prompt** – A structured instruction listing parameters and their values: *"Generate a JSON for the* `/backtests` *endpoint with dataset_id =* `ems_ccys_resampled`*, target_column = BRL, features = [MXN, CLP, COP,*

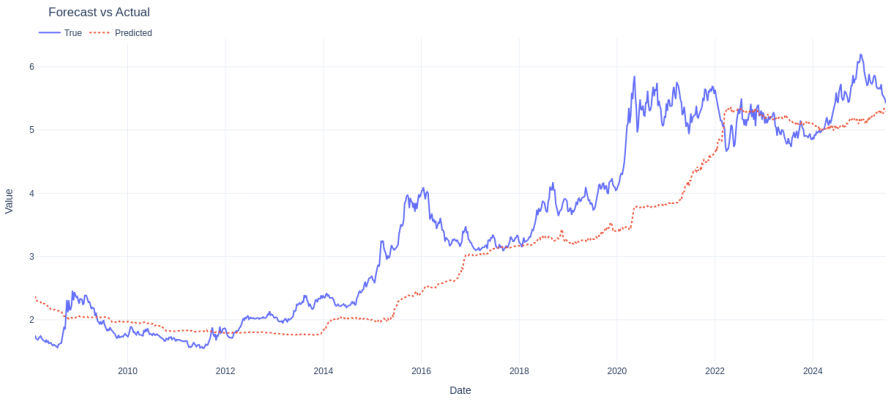

Figure 5. Random Forest forecast vs. actual USD/BRL level

Table 6. Performance metrics and final predictions for each model

Model	RMSE	MAE	Final Predicted Level	Final True Level
Lasso	0.6579	0.4965	5.427	5.432
Random Forest	0.6795	0.5004	5.429	5.432

ZAR], model = Lasso, rolling window = 4 years, tuning frequency = 4 years, horizon = 1, metrics = RMSE and MAE. The dataset has weekly frequency so please take that into account." This version described all necessary fields but did not provide the full schema.

3. **Highly specific prompt** – A detailed instruction including the entire API schema before requesting the JSON. The LLM was explicitly told the field names and types defined in the `BacktestRequest` class, ensuring full context for accurate request generation.

All these prompts were used to test if GPT-5 (OpenAI, 2025), the LLM in question, would be able to produce the correct request JSON. Then, the resulting JSON was tested to see if it yielded a correct response from the API.

The *generic prompt* produced an invalid JSON, with incorrect field names (`target`, `features`, `model`) and an erroneous tuning configuration (`tuning=4` instead of `tuning_frequency=208`), reflecting the lack of schema awareness and time-frequency understanding. In contrast, both the *specific* and *highly specific* prompts yielded valid and executable requests, correctly formatting all parameters.

These results demonstrate that prompt specificity is crucial for accurate LLM–API interaction. Without structured guidance, the LLM misinterpreted field names and temporal parameters. When provided with partial or complete schema information, however, it achieved 100% accuracy. This confirms that by exposing the backtesting process through a well-documented REST API, BACK2SERIES enables GPT-5 and similar models to act as reliable intermediaries for executing modeling workflows—automating both configuration and interpretation in a reproducible, user-friendly manner. Prompts and results are available in the file `LLM_prompts.txt`.

In the future, additional endpoints are envisioned to support analytical tasks. This

would allow LLMs to create complete dashboards with charts and tables, further assisting users in their data science workflows.

5. Conclusions

Time series forecasting remains a critical task in data science, especially in domains where anticipating future movements plays a strategic role. Areas such as economics, logistics, and health strongly benefit from predictive modeling, but evaluating regression models in time series data introduces specific challenges. Standard machine learning workflows tend to ignore temporal structure, often introducing methodological issues such as look-ahead bias and improper validation schemes. These flaws compromise the reliability of conclusions drawn from models and limit their practical usefulness.

Existing Python libraries like `scikit-learn`, `skforecast`, and `sktime` provide valuable tools for regression modeling and time series tasks, but fall short in implementing complete and rigorous backtesting protocols. Important aspects such as periodic hyperparameter tuning, appropriate data scaling, and configurable validation windows are often missing or require heavy customization. Moreover, many of these libraries lack intuitive interfaces for experimentation, limiting their usability for non-specialists.

To address these issues, this work proposed BACK2SERIES, a modular RESTful API designed specifically for time series regression backtesting. The architecture ensures that each step of the backtesting pipeline is configurable and well-structured. BACK2SERIES provides support for expanding and rolling windows, standardization without data leakage, periodic hyperparameter tuning via GSCV, and prediction for multiple horizons. It was built with accessibility and reproducibility in mind, offering a clean interface that abstracts away the technical complexity typically involved in modeling workflows. In organizational contexts, BACK2SERIES can be integrated into corporate analytics pipelines, supporting auditing, compliance, and decision traceability. By exposing forecasting workflows as APIs, the framework aligns with contemporary concerns related to process virtualization, governance, and socio-technical integration.

To demonstrate the capabilities of the BACK2SERIES, a forecasting experiment was conducted using the USD/BRL exchange rate as the target variable and the exchange rates of MXN, CLP, COP, and ZAR as predictors. The test illustrates how the API allows for streamlined backtesting routines, rolling window configuration, and hyperparameter optimization across different model types. Beyond model evaluation, the automatic storage of hyperparameter configurations throughout the time series enabled a broader analysis of model dynamics and their evolution. This case study exemplifies how the tool facilitates structured experimentation and supports model interpretability. Furthermore, as it was also shown, non-technical users can use LLMs as intelligent intermediaries to perform sophisticated modeling tasks through intuitive, conversational interfaces. The API structure facilitates the understanding of endpoints, and through complete and elaborate prompts, LLMs can deliver powerful results.

BACK2SERIES was designed as part of a broader software architecture, prepared for future integration with other tools. As the next step, we plan to develop a web-based application designed to improve the workflow of data scientists when conducting modelling experiments. The application will enable users to register, upload datasets,

configure their experiments visually, and save both models and results for later analysis. Additionally, the BACK2SERIES will be deployed on a remote server, allowing users to interact with the service programmatically without running any local infrastructure. A systematic quantitative comparison with existing libraries such as skforecast and sktime is left as future work, as it would require aligning evaluation objectives beyond the workflow-oriented scope of this study.

Moreover, we also plan to extend BACK2SERIES to support more model families beyond those available in `scikit-learn`, including ensemble learners, neural networks, and probabilistic models. It could also incorporate alternative validation strategies, such as nested cross-validation or even pipelines for systematic trading strategies that depend on forecast signals. These enhancements would strengthen the BACK2SERIES's capabilities and broaden its range of applications.

6. Artifact Availability

BACK2SERIES and all artifacts from our case study are available at: `https://github.com/aisepucrio/backtoseries`.

References

Amat Rodrigo, J. and Escobar Ortiz, J. (2025). skforecast. `https://skforecast.org/`. Accessed: 16 Jul 2025.

Bergmeir, C., Hyndman, R. J., and Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83.

Berthold, Michael R., Cebron, Nicolas, Dill, Fabian, Gabriel, Thomas R., Kötter, Tobias, Meinl, Thorsten, Ohl, Peter, Sieb, Christoph, Thiel, Kilian, and Wiswedel, Bernd (2025). Knime – konstanz information miner. `https://www.knime.com/`. Accessed: 16 Jul 2025.

Demšar, Janez, Curk, Tomislav, Erjavec, Ales, Gorup, Črt, Hočevar, Tomaž, Milutinovič, Matija, Možina, Martin, Polajnar, Ana, Toplak, Marko, Starič, Andrej, Šikonja, Marko, and Zupan, Blaž (2025). Orange data mining - data mining fruitful and fun. `https://orangedatamining.com/`. Accessed: 16 Jul 2025.

European Central Bank (2019). Focus on the euro area economic outlook and cross-border financial resilience. `https://www.ecb.europa.eu/press/economic-bulletin/focus/2019/html/ecb.ebbox201903_02~29b4722819.en.html`. Published: Mar 2019; Accessed: 16 Jul 2025.

Gupta, V. and Hewett, R. (2019). Adaptive normalization in streaming data. In *Proceedings of the 3rd International Conference on Big Data Research*, pages 12–17.

Hyndman, R. J. and Athanasopoulos, G. (2021). *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 3rd edition.

James, G. M., Witten, D., Hastie, T., and Tibshirani, R. (2023). *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer, New York, 3rd edition.

Kenton, W. and Kvilhaug, S. (2023). Backtesting: What it is, how it works, and example. https://www.investopedia.com/terms/b/backtesting.asp. Published: 6 May 2023; Accessed: 16 Jul 2025.

Löning, M., Bagnall, A., Ganesh, S., Kazakov, V., Lines, J., and Király, F. J. (2019). sktime: A unified interface for machine learning with time series. https://www.sktime.net/en/stable/. Accessed: 16 Jul 2025.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, (2025). Scikit-learn user guide. https://scikit-learn.org/stable/user_guide.html. Accessed: 16 Jul 2025.

Ramírez, S. (2025). Fastapi documentation. https://fastapi.tiangolo.com/. Accessed: 16 Jul 2025.

Song, Y., Xiong, W., Zhu, D., Wu, W., Qian, H., Song, M., Huang, H., Li, C., Wang, K., Yao, R., et al. (2023). Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

Zhan, H., Gomes, G., Li, X. S., Madduri, K., and Wu, K. (2018). Efficient online hyperparameter optimization for kernel ridge regression with applications to traffic time series prediction. *arXiv preprint arXiv:1811.00620*.