

Article

LLM-Guided Automated Feature Engineering for Time Series Data with Temporal Leakage Control

Maryam Khanian Najafabadi ¹, Bushra Naeem ², Touraj Khodadadi ³ , Saman Shojae Chaeikar ^{2,*} and Zawar Shah ²

¹ Computer and Data Science, Australian Catholic University, Sydney, NSW 2060, Australia

² Department of Information Technology, Sydney International School of Technology and Commerce, Sydney, NSW 2000, Australia

³ Computer Science Department, Florida Southern College, Lakeland, FL 33801, USA

* Correspondence: samans@sistc.edu.au

Abstract

This study proposes a time series-aware Large Language Model (LLM)-driven feature engineering framework for tabular prediction tasks. Existing automated feature engineering methods, including LLM-based approaches such as CAAFE and OCT-Tree and established libraries such as tsfresh and Featuretools, can generate useful features for general tabular data but do not explicitly address temporal availability constraints in time series settings. This can lead to data leakage when variables that are only available after the prediction event are used directly during model training. To address this limitation, the proposed framework classifies variables into antecedent features, consequent features, and historical aggregated features. The key innovation is that consequent variables are not discarded to prevent leakage but are instead routed into a leakage-safe historical aggregation pipeline, recovering predictive signal from post-event variables through temporally valid past values. The framework guides an LLM to generate structured feature engineering configurations, applies temporally valid transformations, performs feature selection, and evaluates the selected features using predictive models. A formal leakage control mechanism ensures that all aggregations use strictly past observations, applied within entity groups and before the temporal train–validation–test split. The framework is evaluated on two time series tabular tasks: Tesla stock prediction and English Premier League match outcome prediction. The results show that the proposed approach improves predictive performance compared with raw-feature baselines and selected existing automated feature engineering methods. On the Tesla dataset, the framework reduced MAE compared with both the baseline and the reported OCT-Tree result. On the EPL dataset, it improved accuracy compared with the odds-only baseline and the reported Azure ML preprocessing result. These findings suggest that combining LLM reasoning with explicit temporal constraints is a practical direction for automated feature engineering in time series tabular machine learning.



Academic Editor: Isidoros Perikos

Received: 13 May 2026

Revised: 19 June 2026

Accepted: 26 June 2026

Published: 1 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: Large Language Models; automated feature engineering; time series data; tabular machine learning; temporal data leakage; historical aggregation; predictive modelling; AutoML; stock price prediction; sports analytics

1. Introduction

Time series tabular data are widely used in many real-world domains, including finance, healthcare, environmental monitoring, energy management, and sports analytics. Accurate predictive models built on such data can support timely decision-making by

identifying patterns, trends, and future changes. However, the performance of machine learning models on time series tabular data depends heavily on the quality of input features. Raw variables often fail to capture important temporal relationships, such as recent trends, lagged effects, rolling patterns, and historical changes. Therefore, effective feature engineering is essential for improving predictive performance in time series modelling [1,2].

Feature engineering refers to the process of transforming raw data into more informative variables that better support machine learning models, and differs from feature selection and feature extraction in that it actively creates new variables rather than selecting or compressing existing ones [1–3]. In time series tasks, useful predictive signals are often hidden in historical observations such as rolling averages, lag values, and trend-based indicators, making feature engineering especially important for capturing temporal behaviour [4].

Despite its importance, time series feature engineering remains challenging. It often requires domain knowledge to decide which historical windows, aggregation functions, and transformations are meaningful for a specific prediction task. Selecting these transformations manually is time-consuming, problem-specific, and difficult to generalise across datasets. An additional risk is data leakage, where future or unavailable information used during feature construction leads to unreliable and unrealistic model performance.

Recent advances in Large Language Models (LLMs) have created new opportunities for automating feature engineering. LLMs can interpret dataset descriptions, understand column semantics, and suggest meaningful feature transformations based on task context. Existing LLM-based feature engineering frameworks such as CAAFE and OCT-Tree have shown promising results on general tabular datasets [4–6], but treat each row as an independent sample and have no representation of when a variable becomes observable relative to the prediction event. AutoML tools such as Azure AutoML address temporal structure only through predefined lag and calendar transformations, without semantic, task-aware reasoning about which transformations are meaningful for a given variable [7].

To address this gap, this study proposes a time series-aware LLM-driven feature engineering framework for tabular datasets. The framework makes temporal availability a first-class input to the generation process, allowing users to declare which variables are available at prediction time and guiding the LLM to reason about feature construction under explicit causal constraints. Specifically, the framework separates variables into antecedent features, consequent features, and historical aggregated features. The key innovation is that the antecedent/consequent distinction is not used merely as a safety filter that discards unsafe variables. A naive leakage prevention rule would simply drop post-event columns such as daily high, low, and volume. Our framework instead routes these consequent variables into a historical aggregation pipeline, where only their past values are used to construct temporally valid features. This enables the framework to recover predictive signal from variables that a simple filtering rule would have removed, while keeping model evaluation free from leakage.

The proposed framework is evaluated on two time series tabular prediction tasks: Tesla stock price prediction and English Premier League match outcome prediction. The results show that the LLM-guided feature engineering process can improve predictive performance compared with raw-feature baselines and selected existing automated feature engineering approaches. These findings suggest that combining LLM reasoning with explicit temporal constraints can provide a practical direction for automated feature engineering in time series tabular machine learning.

The main contributions of this study are as follows:

- A time series-aware LLM feature engineering framework is proposed that makes temporal availability a first-class input to LLM-guided feature generation for tabular time series prediction tasks.
- A temporal feature classification mechanism is introduced to separate variables into antecedent, consequent, and historical aggregated features. Rather than discarding post-event variables to prevent leakage, the mechanism retains them and exploits their past values through historical aggregation, recovering a predictive signal that a simple filtering rule would remove.
- A multi-stage LLM-guided feature generation pipeline is designed to generate direct transformations and historical aggregation features in a structured and task-aware manner.
- The framework is evaluated across two domains, Tesla stock price prediction and English Premier League match outcome prediction.
- Experimental results show improved predictive performance over raw-feature baselines and competitive performance compared with selected AutoML and LLM-based feature engineering benchmarks.

2. Literature Review

2.1. Feature Engineering, Feature Extraction, and Feature Selection

Feature engineering, feature extraction, and feature selection are closely related but distinct processes in machine learning, as summarised in Table 1. Unlike feature selection, which retains the most relevant existing variables, or feature extraction, which compresses variables into lower-dimensional representations, feature engineering actively creates new informative variables through transformations, aggregations, and domain-specific operations [1,2,5]. In time series tabular data, this distinction is especially important because predictive information is often embedded in temporal patterns rather than in the current observation alone. Examples of engineered temporal features include lag values, rolling averages, rolling minimums and maximums, trend indicators, and historical summary statistics [3,5]. This capacity to create new time-dependent variables makes feature engineering a more flexible and context-aware approach for improving model performance in time series prediction tasks compared with feature selection or extraction alone.

Table 1. Comparison between feature extraction, feature engineering, and feature selection.

Method	Description	Advantages	Limitations
Feature Extraction [2,5]	Transforms original features into a new representation, often with the aim of reducing dimensionality while preserving important information. Examples include PCA, autoencoders, and learned representations.	Reduces feature dimensionality; can simplify model input; can be automated using statistical or deep learning methods.	May reduce interpretability; does not necessarily use domain knowledge; may not explicitly capture time-dependent relationships in time series data.
Feature Engineering [1,3,5]	Creates new informative features from raw data using transformations, aggregations, statistical operations, or domain knowledge. Examples include lag features, rolling averages, ratios, logarithmic transformations, and trend indicators.	Can incorporate domain knowledge; can capture meaningful temporal patterns; often improves model performance in tabular and time series tasks.	Often requires domain expertise; can be time-consuming; may introduce data leakage if temporal constraints are not handled carefully.

Table 1. *Cont.*

Method	Description	Advantages	Limitations
Feature Selection [2]	Selects the most relevant subset of existing features to improve model efficiency and reduce redundancy or overfitting.	Reduces model complexity; removes irrelevant or redundant variables; can improve generalisation.	Does not create new features; cannot capture hidden temporal patterns unless they already exist in the dataset.

2.2. Deep Learning-Based Feature Extraction for Time Series Data

Deep learning methods including Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), and hybrid CNN-LSTM architectures have been widely applied for automatic feature extraction from time series data, demonstrating strong capacity for capturing local temporal patterns and long-term dependencies [8,9]. For example, Zhang et al. proposed a CNN-LSTM-based method for remaining useful life prediction in proton exchange membrane fuel cells, where CNN layers extracted informative temporal features before LSTM-based sequence modelling [10]. While these approaches can improve predictive performance across various domains, they focus on learned representations rather than explicit and interpretable feature construction. As a result, they may provide limited transparency about the constructed features and do not directly incorporate domain-specific or semantic knowledge into the feature creation process, which distinguishes them from the LLM-guided feature engineering approach proposed in this study.

2.3. Time Series Feature Extraction and Decomposition Methods

Decomposition-based methods such as Empirical Mode Decomposition, Variational Mode Decomposition, and CEEMDAN have been widely used to extract useful representations from time series signals by separating them into simpler components before downstream modelling [11,12]. These methods have been applied across several domains including environmental monitoring, solar radiation prediction, power systems, and financial forecasting [13–16]. Bayesian modelling workflows and differentiation-based preprocessing techniques have similarly been applied to handle sparse, non-stationary, or distribution-shifted time series data [17,18]. Although effective in their respective domains, these approaches require careful methodological choices such as selecting decomposition parameters and defining stopping criteria. More importantly, they do not support semantic, task-aware, or temporally constrained feature generation, which is the core focus of the proposed framework.

2.4. Historical and Statistical Aggregation in Time Series Feature Engineering

Historical and statistical aggregation is one of the most common and effective feature engineering strategies for time series tabular data. These methods generate new variables by summarising past observations over predefined time windows. For example, Balasubramaniam et al. applied historical climate aggregation over different time windows to improve pasture yield prediction, showing that time-lagging effects can be captured through carefully designed historical features [3]. Similar strategies have been used in sports analytics, where previous match statistics are aggregated to predict future match outcomes [13,14,19]. In health-related forecasting, moving averages and exponential smoothing methods have also been applied to model disease trends over time [15,16].

These studies show that historical aggregation can improve predictive performance by capturing recent trends, lag effects, and temporal dependencies. However, a major limitation is that the design of aggregation windows and functions is usually manual and task-

specific. For example, deciding whether to use a 3-day, 5-day, 20-day, or 60-day window often requires domain expertise or extensive experimentation [3,20,21]. This limits the scalability and generalisability of manually designed time series feature engineering methods.

2.5. LLM-Based Automated Feature Engineering

Large Language Models have recently been explored as tools for automated feature engineering. The main motivation is that LLMs can interpret dataset descriptions, understand column semantics, and generate meaningful feature transformations based on the task context. This allows LLMs to act as virtual domain experts that support the feature engineering process.

2.5.1. CAAFE

Context-Aware Automated Feature Engineering, known as CAAFE, was introduced as an LLM-based framework for automated feature generation [6]. CAAFE uses dataset descriptions and field definitions as prompts to guide the LLM in generating Python 3.11.5 code for candidate feature transformations. The generated features are then applied to the dataset and evaluated using model performance. Only features that improve performance are retained [6].

This figure shows how dataset descriptions and field definitions are provided to the LLM, which then generates feature engineering code or rules. The generated features are applied to the dataset and evaluated based on model performance.

As shown in Figure 1, CAAFE demonstrates the potential of LLMs for automated data science tasks by using contextual information to guide feature generation. However, CAAFE was mainly designed for general tabular datasets. Its generated transformations usually operate within individual samples rather than across historical observations [6]. This creates a limitation for time series tabular data, where useful features often depend on previous time steps, rolling windows, and temporal availability constraints.

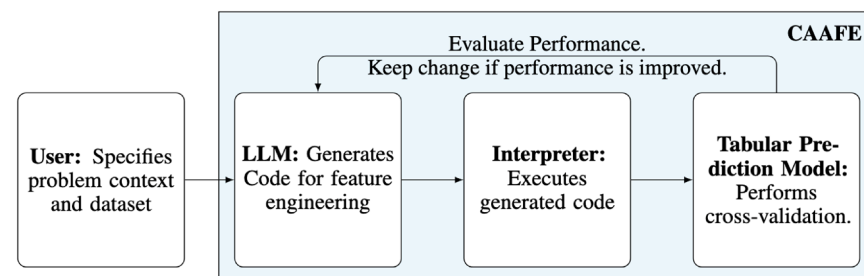


Figure 1. CAAFE architecture for LLM-based automated feature engineering.

2.5.2. OCT-Tree

OCT-Tree extends LLM-based feature engineering by incorporating decision-tree reasoning into the feature generation process [4]. In this framework, the LLM first generates candidate features. A tree-based model is then trained and evaluated using the generated features. The validation score and a string representation of the decision-tree logic are then provided back to the LLM as feedback for the next round of feature generation [4].

This iterative process allows the LLM to refine its feature generation based on model performance and interpretable tree-based feedback. OCT-Tree achieved strong results on tabular benchmark datasets and included an experiment on Tesla stock data [4]. However, although OCT-Tree was applied to a financial time series dataset, it does not explicitly model the difference between variables available before prediction and variables only available after the prediction event. Therefore, when applied to time series data, it may still face the risk of inappropriate feature construction or data leakage.

2.6. Automated Feature Engineering Libraries for Time Series Data

Several established libraries provide automated feature engineering capabilities for time series data and represent important non-LLM baselines for contextualising the proposed framework [6–13,17,18].

tsfresh is a widely used Python library that automatically extracts a large number of statistical and temporal features from time series data, including autocorrelation, entropy, peak counts, and rolling statistics [22]. It supports systematic feature extraction across many time steps and includes built-in feature selection based on statistical hypothesis testing. However, tsfresh operates on pre-segmented time series windows and does not incorporate semantic reasoning about column meaning or task context. Critically, it does not model temporal availability constraints: if a post-event variable is present in the input, tsfresh will extract features from it without distinguishing whether those values were available at prediction time.

Featuretools and its Deep Feature Synthesis algorithm perform automated relational feature engineering by generating features across entity sets using stacking of aggregation and transformation primitives. While powerful for relational data, Featuretools is designed around entity relationships rather than temporal prediction settings. It supports time-based cutoffs to prevent leakage in some configurations, but the specification of these cutoffs requires manual setup by the user and does not involve semantic reasoning about which variables are causally available at prediction time [7–11].

sktime provides a unified interface for time series classification, regression, and forecasting, and supports a range of feature extraction transformers. Like tsfresh, sktime's feature engineering components operate on pre-defined time series inputs and do not incorporate task-aware semantic reasoning or explicit temporal availability modelling [22].

In contrast to all of these tools, the proposed framework uses an LLM to reason semantically about dataset columns, task objectives, and temporal availability constraints simultaneously. Rather than applying a fixed catalogue of transformations uniformly across all variables, the framework distinguishes between antecedent and consequent variables and routes each through an appropriate generation pathway. This allows the framework to recover predictive signal from post-event variables through leakage-safe historical aggregation, a capability that is not present in any of the above tools without substantial manual configuration.

2.7. Existing AutoML Pipelines for Time Series Data

AutoML platforms provide automated preprocessing and feature generation support for time series forecasting tasks. For example, Azure AutoML includes time series-specific transformations such as calendar-based features and lag features [7]. These transformations can be useful because they allow models to incorporate basic temporal information without requiring users to manually engineer every feature.

However, the feature engineering coverage of existing AutoML tools remains limited. In many cases, these tools support simple lag-based or calendar-based features but do not fully support richer multi-step feature engineering [7]. For example, a stock forecasting task may benefit from comparing the current price with a monthly average and then applying a weekly aggregation to that derived value. Such multi-stage, domain-aware feature construction is not fully supported by many existing AutoML pipelines. This limitation motivates the need for a more flexible automated feature engineering framework for time series tabular data.

2.8. Summary of Existing Time Series Feature Engineering Studies

Previous studies have applied feature engineering and feature extraction methods across different time series domains, including agriculture, finance, energy, sports, health-care, environmental monitoring, and digital systems [3,9,10,13–15,20,23,24]. These studies show that engineered temporal features can improve model performance across different predictive tasks. For example, historical aggregation improved pasture yield prediction [3], decomposition-based techniques improved forecasting in solar radiation and power load datasets [14,15], and manually engineered features improved sports outcome prediction [20].

Table 2 summarises representative studies that applied feature engineering, feature extraction, decomposition, aggregation, or automated feature generation methods to time series datasets across different domains. These studies show that engineered temporal features can improve predictive performance, but many approaches still depend on manual design choices, including the selection of aggregation windows, decomposition strategies, and task-specific transformations.

Table 2. Summary of previous studies on time series feature engineering and evaluation datasets.

Dataset	Domain	Problem Type	Metrics
Pasture yield dataset using historical climate feature engineering [3]	Agriculture	Regression on pasture yields in Australian farms	$R^2 = 0.89$ MAE = 192.64 RMSE = 329.52
Private Azure cloud usage dataset using feature engineering for serverless scaling decisions [9]	Digital/Cloud computing	Binary classification for optimal cloud site scaling	Accuracy = 0.9183 AUC = 0.9538 F1-score = 0.9183 Precision = 0.9216 Recall = 0.9190
Proton exchange membrane fuel cell dataset using CNN-LSTM-based feature extraction [10]	Energy	Regression for fuel cell lifespan prediction	$R^2 = 91.09\%$ MAE = 0.0019
Tesla daily stock dataset used in OCT-Tree LLM-based feature generation [4]	Finance	Regression for Tesla stock price prediction	Mean error improvement = 17.1%; MAE = 5.48
Exchange rate between USD and RMB [12]	Finance	Regression on USD to RMB exchange rate	MAE = 0.0153 RMSE = 0.0184
S&P 500 stock data using preprocessing and neural networks for trended forecasting [18]	Finance	Regression for next-step stock price prediction	R^2 , MSE, and RMSE improved by approximately 30%
PM2.5 dataset using trend feature extraction [13]	Environmental	PM2.5 prediction	$R^2 = 0.786$ MAE = 2.785 MSE = 11.25
Solar radiation dataset using CEEMDAN-EWT-CNN-GRU-SVM decomposition and prediction [14]	Environmental	Solar radiation prediction	$R^2 = 0.9993$
WIFI occupancy [19]	Digital	Occupancy regression	$R^2 = 0.9786$
Electricity usage dataset using time series feature extraction for ultra-short-term load forecasting [15]	Energy	Electricity usage	MAE improvement = 5.3748%

Table 2. *Cont.*

Dataset	Domain	Problem Type	Metrics
Stock index datasets using decomposition-based feature extraction and optimised deep learning [16]	Finance	Stock price prediction	MAE improvement ranged from 30% to 85% across indexes
English Premier League and weather data using predictive analytics for soccer match outcomes [20]	Sports	Win or not win binary classification	Accuracy = 64.6%
Official CBA basketball dataset using hybrid ensemble learning [21]	Sports	Win or Lose classification	Acc = 84% F1-score = 82%
Food demand time series dataset using averaging-based forecasting [25]	Food	Food Demand Regression	RMSSE = 0.439
S&P 500 index data using explainable AI for financial time series prediction [26]	Finance	Indexes prediction	Acc = 0.7987
Dengue fever dataset using moving average and exponential smoothing [23]	Health	Dengue fever number regression	MAD = 23.38 MSD = 707.65 MAPE = 12.53%
Acute hemorrhagic conjunctivitis dataset using SARIMA and exponential smoothing [24]	Health	Acute hemorrhagic conjunctivitis Count regression	RMSE = 563.490 MAPE = 13.810
Water demand dataset using temporal and spatial aggregation [27]	Environmental	Regression for water demand prediction	CD = 0.46
Power system operating scenario dataset using deep time series aggregation [28]	Energy	Power consumption estimation	AIC = 516 BIC = 739

As shown in Table 2, time series feature engineering has been successfully applied across agriculture, finance, energy, health, environmental monitoring, and sports analytics. However, most existing approaches require manually defined feature transformations or domain-specific assumptions. This motivates the development of an automated and time series-aware feature engineering framework that can use LLM reasoning while maintaining temporal validity.

2.9. Research Gap and Motivation

The reviewed literature shows that feature engineering substantially improves time series prediction across domains, but most approaches depend on manually designed transformations, manually selected time windows, or task-specific domain expertise [3,19,20,23]. LLM-based methods such as CAAFE and OCT-Tree automate feature construction but are designed for general tabular data and do not distinguish between variables available at prediction time and variables that become available only afterward [4,6]. This limitation can introduce data leakage and lead to unrealistic model performance. Existing AutoML tools partially address temporal structure through lag and calendar features but do not support multi-stage, semantically grounded feature construction that combines LLM reasoning with explicit temporal constraints [7].

This gap motivates the proposed framework, which encodes temporal availability directly into the LLM generation process. Rather than filtering out post-event variables, the framework routes them into a historical aggregation pipeline that recovers their predictive

signal through temporally valid past values, enabling richer feature construction while maintaining leakage control.

3. Methodology

3.1. Overview of the Proposed Framework

This study proposes a time series-aware LLM-driven feature engineering framework for tabular prediction tasks. The framework enables users to define the temporal availability of dataset variables before feature generation. Based on these user-defined temporal dependencies, the LLM generates feature engineering configurations that follow the established constraints. This design ensures that generated features are consistent with the structure of time series data and reduces the risk of data leakage.

The framework follows a sequential process. First, the user provides a dataset description and temporal dependency configuration. Second, the LLM generates feature engineering configurations for antecedent features, consequent features, and historical aggregated features. Third, the generated configurations are applied to the dataset to create engineered features. Finally, feature selection is applied, and the selected features are used for model training and evaluation.

Figure 2 shows the full pipeline, including dataset description, temporal dependency configuration, LLM-based feature configuration generation, feature implementation, feature implementation, feature selection, and model evaluation.

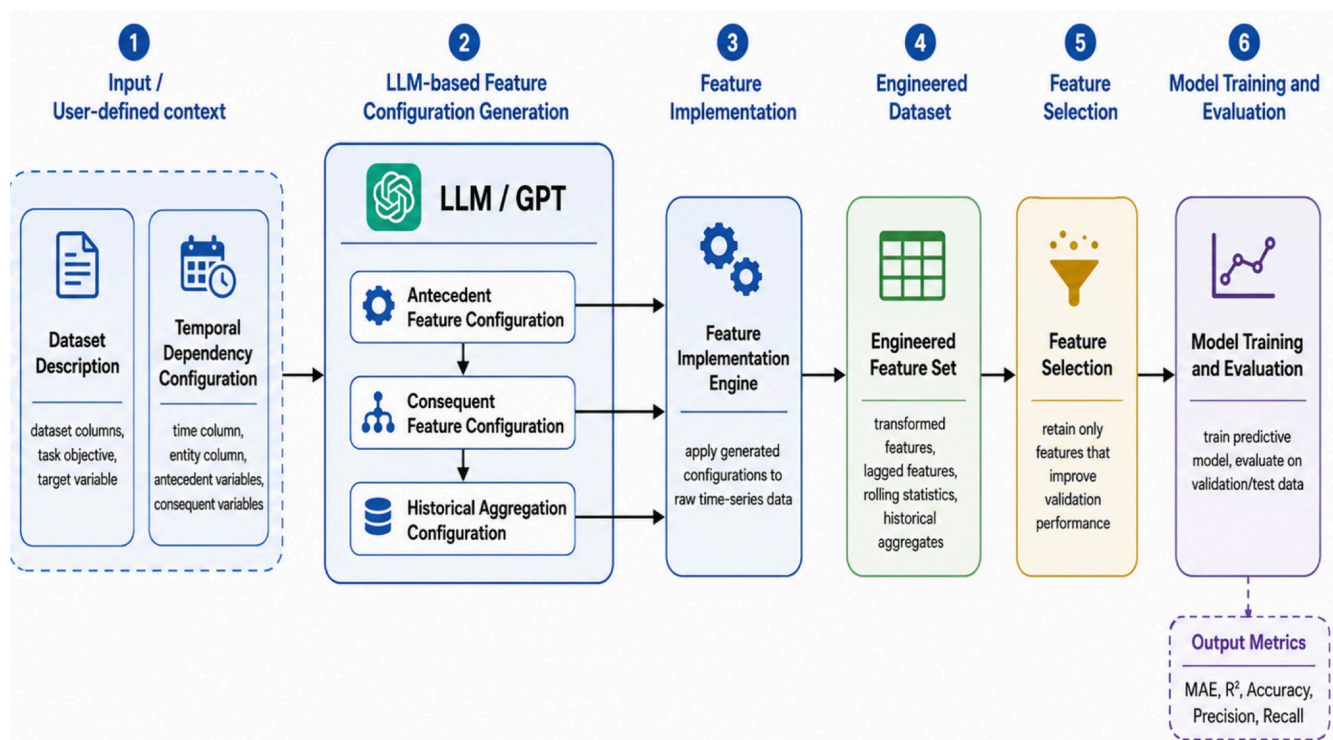


Figure 2. Overall workflow of the proposed LLM-driven time series feature engineering framework.

3.2. Temporal Feature Classification

A key component of the proposed framework is the classification of variables according to their temporal availability. This classification guides the LLM and prevents the use of unavailable future information during feature generation. The framework defines three feature types: antecedent features, consequent features, and historical aggregated features. Figure 3 illustrates how each feature type is classified and routed through the framework pipeline.

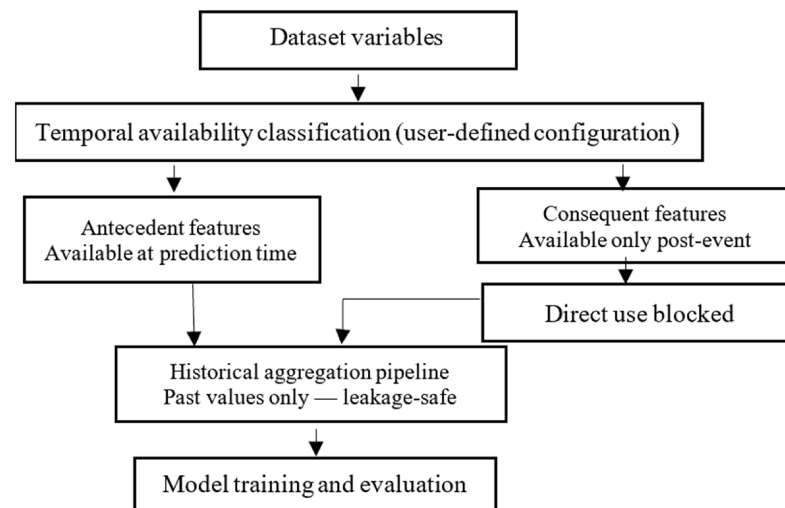


Figure 3. Temporal feature classification and routing in the proposed framework. Antecedent features are used directly at prediction time, while consequent features are blocked from direct use and routed to the historical aggregation pipeline. Both feature types contribute to the aggregated feature set through past values only, ensuring leakage-safe model training.

Antecedent features are variables that are available before or at the time of prediction. These features can be used directly for model training and inference without introducing data leakage. For example, if the task is to predict a stock's closing price after the market opens, the opening price can be treated as an antecedent feature because it is already available before the closing price is observed.

Consequent features are variables that become available only after the prediction event. These variables cannot be used directly as model inputs because they may contain information about the target outcome. For example, when predicting a stock's closing price, the current day's high, low, volume, and adjusted close may not be available at prediction time. Using them directly would introduce data leakage. However, their historical values from previous time steps can still be used to construct valid historical aggregated features.

Historical aggregated features are features generated by aggregating past values of antecedent or consequent variables. These features are computed using only previous time steps, ensuring that no future information is included. Examples include rolling means, previous values, rolling minimums, rolling maximums, and exponential moving averages.

As shown in Figure 3, antecedent features follow a direct path to model training, while consequent features are blocked from direct use and routed to the historical aggregation pipeline. Importantly, both antecedent and consequent variables contribute past values to the aggregation pipeline, allowing the framework to recover predictive signal from post-event variables in a leakage-safe manner. This temporal feature classification allows the framework to generate richer time series features while maintaining temporal validity. By explicitly separating direct-use variables from variables that can only be used historically, the framework reduces the risk of data leakage and improves the reliability of model evaluation.

3.3. User-Specified Temporal Dependency Configuration

The proposed framework requires the user to provide temporal dependency information for the dataset. This information is supplied through a structured configuration file, such as a JSON file. The configuration identifies the target variable, time column, entity column, and variables that are available before the prediction event.

Figure 4 shows a sample JSON configuration used to define the target variable, time column, entity column, and antecedent variables. Variables not marked as antecedent are treated as consequent variables.

```

1  [
2      {
3          "column_name": "date",
4          "data_type": "string",
5          "description": "The trading day in YYYY-MM-DD format.",
6          "is_antecedent": true,
7          "define_time_series": true
8      },
9      {
10         "column_name": "open",
11         "data_type": "floating",
12         "description": "The opening price on the given trading day.",
13         "is_antecedent": true
14     }
15 ]

```

Figure 4. Example of user-specified temporal dependency configuration.

The configuration allows the framework to determine which variables can be used directly and which variables should only be used through historical aggregation. Variables marked as antecedent are considered available at prediction time, while variables not marked as antecedent are treated as consequent features.

This design makes the framework flexible across different prediction settings. For example, in a stock prediction task, the availability of the opening price depends on when the prediction is made. If the prediction is made before the market opens, the opening price should be treated as a consequent feature. If the prediction is made after the market opens, the opening price is available and can be treated as an antecedent feature.

3.4. Feature Engineering Workflow

The proposed framework performs feature engineering in a structured sequence. The process consists of four main stages.

First, the LLM is introduced to the feature engineering task, including the dataset description, target variable, temporal dependency configuration, and definitions of antecedent, consequent, and historical aggregated features. This step ensures that the LLM generates features within the intended time series-aware framework.

Second, the LLM generates feature configurations for antecedent variables. These configurations describe direct transformations that can be applied to variables available at prediction time.

Third, the LLM generates feature configurations for consequent variables. Consequent variables are not used directly for prediction, but they may be transformed and later used through historical aggregation. This allows the framework to make use of useful past information without exposing the model to future values.

Fourth, the framework generates historical aggregated features by applying aggregation functions to previous time steps only. These features are created using past values of both antecedent and consequent variables. This stage is central to the proposed time series design because it captures temporal patterns while maintaining leakage control.

After all configurations are generated, the framework applies them to the dataset and produces the engineered feature set for downstream feature selection and model training.

3.5. Feature Engineering Configuration

The LLM does not directly modify the dataset. Instead, it generates structured feature engineering configurations. These configurations define how each new feature should be created and allow the framework to apply the generated transformations consistently.

For antecedent and consequent features, each configuration includes the name of the generated feature, the raw field or fields used to create it, the transformation process, a short description, and the output data type, as summarized in Table 3.

Table 3. Antecedent and consequent feature configuration fields.

Field	Description
name	Name of the generated feature
raw_field	Original field or fields used to create the feature
processor	Transformation logic used to generate the feature
description	Short explanation of the generated feature
data_type	Data type of the generated feature

For historical aggregated features, the configuration specifies the entity, the variable to be aggregated, the aggregation function, the time window, and a description of the generated feature as summarized in Table 4.

Table 4. Historical aggregation configuration fields.

Field	Description
entity	Entity or grouping variable used for aggregation
aggregate_on	Variable to be aggregated
agg_type	Aggregation function, such as mean, minimum, maximum, or previous value
timeframes	Historical window sizes used for aggregation
description	Short explanation of the generated historical feature

For example, if the entity is a stock symbol and the aggregation field is daily high price, the framework can generate features such as the average high price over the previous 3, 5, or 20 days. The framework applies each aggregation only to historical observations, ensuring that the current or future values are not included.

3.6. LLM Prompt and Configuration Generation

The framework uses an LLM to generate feature engineering configurations based on the dataset description, temporal dependency configuration, and task objective. Each prompt provides the LLM with the relevant context, including the meaning of dataset columns, the target variable, the prediction setting, and the feature type currently being generated.

In this study, all configurations were generated using GPT-4o (OpenAI, model version gpt-4o-2024-08-06). The temperature was set to 0.0 to minimise output variability and improve reproducibility across runs. The LLM was accessed via the OpenAI API, and all prompts were submitted programmatically without manual intervention during generation.

The prompt flow is designed as a staged process consisting of three sequential calls per dataset. In the first call, the LLM receives the full dataset description, column definitions, target variable, and temporal dependency configuration, and is asked to generate

antecedent feature configurations. In the second call, the previously generated antecedent configurations are appended to the context, and the LLM generates consequent feature configurations. In the third call, both antecedent and consequent configurations are included in the context, and the LLM generates historical aggregation configurations. This staged design ensures that the LLM builds features progressively and remains aligned with the temporal structure of the prediction task.

Each prompt explicitly instructs the LLM to return output as a strictly valid JSON array, with no additional text, explanation, or markdown formatting. The framework parses each response by extracting the JSON content directly. If the response cannot be parsed as valid JSON, the framework automatically retries the call once with an additional instruction reinforcing the required format. Configurations that remain invalid after one retry are logged and skipped. In practice, fewer than 8% of generated configurations required a retry across both datasets, and no manual correction of configurations was performed.

Figure 5 shows how the dataset description, task description, temporal constraints, and previously generated configurations are passed to the LLM to generate the next set of feature engineering configurations.

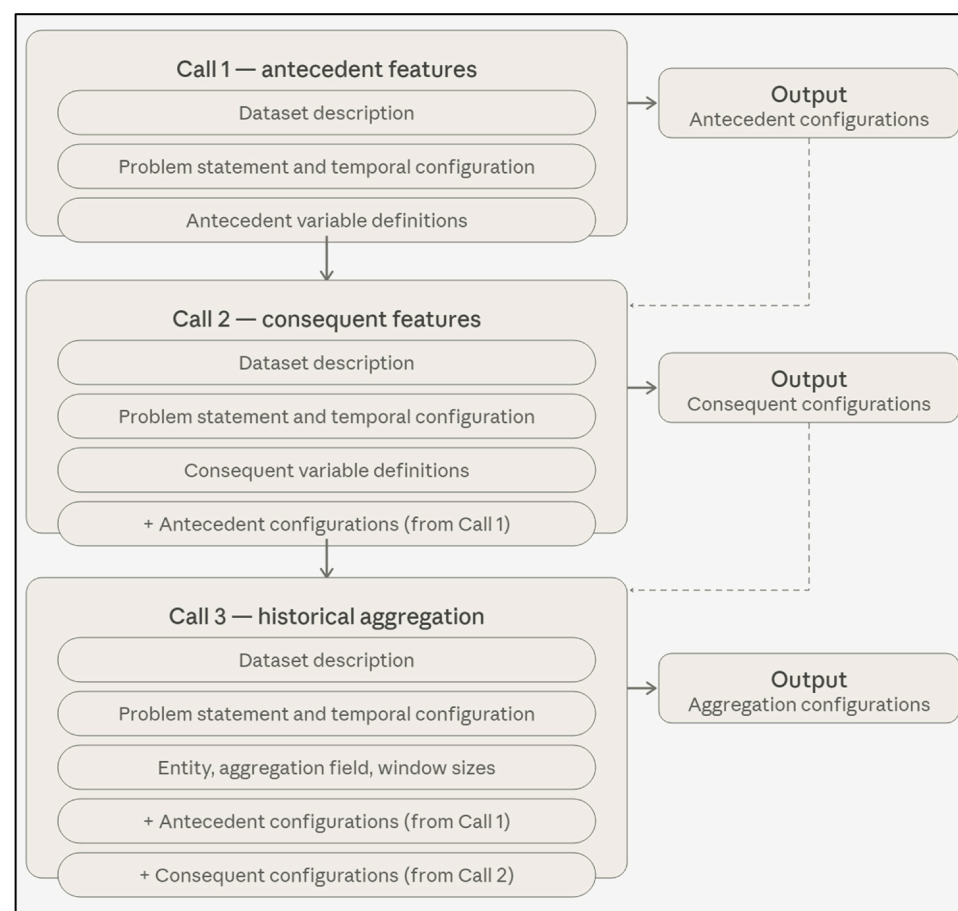


Figure 5. LLM prompt flow for staged feature configuration generation.

Once the LLM returns a response, the framework extracts the generated configurations and stores them for feature implementation. This staged design ensures that the LLM builds features progressively and remains aligned with the temporal structure of the prediction task.

3.7. Feature Generation Pipeline

After the configurations are generated, the framework applies them to the dataset to produce engineered features. The feature generation process is separated into two main pipelines: one for antecedent and consequent feature transformations, and one for historical aggregation.

For antecedent and consequent feature transformations, the framework applies the specified processor to the relevant raw fields and creates a new feature column. Antecedent features may be used directly for model training and inference. Consequent features are not used directly as current-time inputs, but they can be passed to the historical aggregation pipeline.

For historical aggregation, the dataset is first grouped by the specified entity, such as a stock, team, or other time series unit. Within each group, observations are sorted chronologically. The framework then applies the selected aggregation functions over the specified historical windows. For each current observation, only previous observations are used to compute the aggregated feature.

Figure 6 shows how historical aggregated features are generated by grouping records by entity, sorting them by time, and applying aggregation functions only to previous observations.

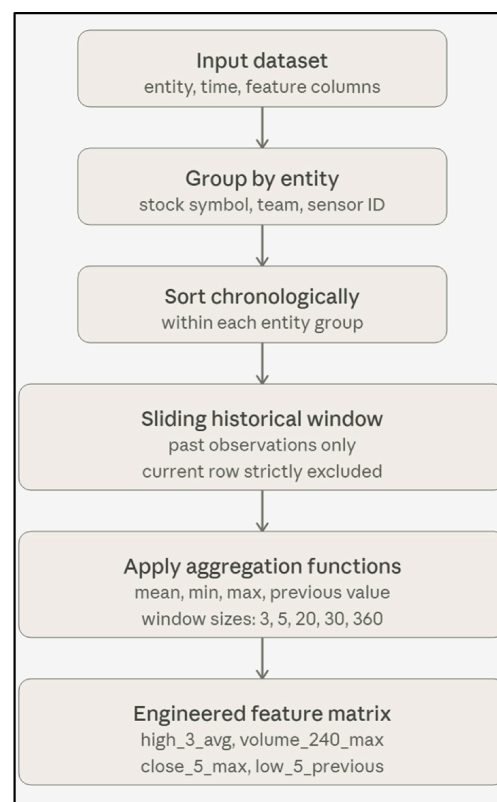


Figure 6. Historical aggregation pipeline with leakage control.

3.8. Leakage Control Mechanism

Data leakage is a major risk in time series feature engineering because variables observed after the prediction event may accidentally be used during model training. The proposed framework addresses this risk through two mechanisms.

First, the temporal dependency configuration separates variables into antecedent and consequent features. Antecedent features can be used directly because they are available at prediction time. Consequent features are isolated and cannot be used directly as current-time model inputs.

Second, the historical aggregation pipeline only uses previous observations when constructing features for the current sample. Even when a variable is classified as consequent, its past values can be used safely because they would have been available before the current prediction event.

Together, these mechanisms allow the LLM to generate useful time series features while maintaining causal consistency. Algorithm 1 presents the leakage-safe historical aggregation procedure. This improves the trustworthiness of model evaluation and reduces the likelihood of inflated performance caused by future information.

Algorithm 1: Leakage-Safe Historical Aggregation

Input: Dataset D with columns [entity, time, features]

Aggregation configurations C = {entity, field, agg_type, timeframes}

Output: Engineered feature matrix D'

1. Sort D by [entity, time] in ascending order
2. For each configuration c in C:
 - a. Group D by c.entity
 - b. For each group G:
 - For each row i in G (in chronological order):
 - past_obs = G[time < G[i].time] //strictly excludes current row
 - For each window w in c.timeframes:
 - past_window = past_obs[last w observations]
 - If len(past_window) > 0:
 - D'[i, feature_name] = apply(c.agg_type, past_window[c.field])
 - Else:
 - D'[i, feature_name] = NaN //insufficient history
3. Apply temporal split to D':
 - train = D'[time <= split_date_1]
 - validation = D'[split_date_1 < time <= split_date_2]
 - test = D'[time > split_date_2]
4. Perform feature selection using validation performance only:
 - For each candidate feature f:
 - score = evaluate(train + f, validation)
 - If score improves metric:
 - retain f
 - Else:
 - discard f
5. Train final model on train using retained features
 - Evaluate on test set only

Output: Final feature matrix with retained leakage-safe features

To ensure complete leakage prevention, the order of operations in the proposed framework is as follows. First, all historical aggregation features are computed across the full dataset using only strictly past observations for each row, with each aggregation applied within entity groups and excluding the current time step. Second, the temporal split into training, validation, and test sets is applied to the fully engineered feature matrix. This ordering ensures that no future information from the validation or test periods influences feature construction. Third, iterative feature selection is performed using validation performance only, with the test set remaining unseen throughout this process. Finally, the selected model is evaluated once on the held-out test set. Where standardisation or scaling is applied, the scaler is fitted exclusively on the training set and applied separately to the validation and test sets, preventing any distributional leakage from held-out data.

3.9. Feature Selection and Model Preparation

The LLM-generated pipeline can create a large number of candidate features. To retain only useful features, the framework applies an iterative feature selection process. Candidate features are evaluated based on their effect on validation performance. If a feature improves the selected evaluation metric, it is retained; otherwise, it is discarded.

The selected features are then combined with the valid baseline features and used to train the final predictive model. This step ensures that the framework does not rely only on LLM-generated suggestions but verifies their practical contribution through model-based evaluation.

To guard against validation overfitting during sequential feature selection, the dataset is divided into three non-overlapping partitions following a chronological split: a training set used for model fitting, a validation set used exclusively for feature selection decisions, and a held-out test set that remains completely unseen throughout the entire feature generation, selection, and model development process. Feature selection decisions are made based solely on validation performance, and the test set is used only once for final evaluation reporting. This three-way separation ensures that reported performance reflects genuine generalisation rather than overfitting to the validation set.

4. Experimental Setup

4.1. Overview

The proposed framework was evaluated on two time series tabular prediction tasks from different domains: Tesla stock price prediction and English Premier League (EPL) match outcome prediction. The Tesla dataset was used to evaluate the framework on a financial regression task, while the EPL dataset was used to evaluate its performance on a sports classification task. These two datasets allow the framework to be tested across different prediction settings, target types, and temporal structures.

The evaluation focuses on comparing model performance before and after applying the proposed LLM-driven feature engineering framework. For each dataset, the framework first generates candidate features, then applies iterative feature selection, and finally evaluates the selected features using a suitable predictive model and task-specific metrics.

4.2. Tesla Stock Price Prediction Dataset

The first experiment uses the publicly available Tesla stock dataset from Kaggle [10], available at <https://www.kaggle.com/datasets/guillemservera/tsla-stock-data> (accessed on 30 January 2026). This dataset was selected because it is similar to the financial time series dataset used in OCT-Tree, an existing LLM-based feature engineering framework [4],

allowing the proposed method to be compared against an existing automated feature generation approach.

The dataset contains 3473 daily trading records. Each record represents one trading day and includes the following variables: date, open price, close price, high price, low price, trading volume, adjusted close, change percentage, and 20-day average volume. The dataset was split chronologically into training, validation, and test sets, with earlier records used for training and the most recent records reserved for testing, ensuring that no future information is used during model development.

For this experiment, the prediction task focuses on estimating Tesla stock movement using temporally valid features. Variables were classified according to their temporal availability. The date, open price, and 20-day average volume are treated as antecedent features, as they are available before the closing price is observed. The remaining variables—close price, high price, low price, trading volume, and adjusted close—are treated as consequent features, as they become available only after the prediction point and cannot be used directly as model inputs. However, their historical values from previous time steps are used to generate valid historical aggregated features. Missing values arising from insufficient historical observations in early rows of the aggregation window are excluded from feature selection and model training.

The daily percentage change is defined as:

$$\text{Change\%} = \frac{\text{Close}_t - \text{Close}_{t-1}}{\text{Close}_{t-1}} * 100$$

where Close_t is the closing price at the current time step and Close_{t-1} is the closing price at the previous time step. The use of percentage change helps reduce the effect of raw price scale and allows the model to focus on relative stock movement rather than absolute price values.

4.3. Tesla Experimental Model, Feature Selection, and Metrics

XGBoost was used as the predictive model for the Tesla experiment. This choice is appropriate because XGBoost performs strongly on tabular datasets and has also been used in recent LLM-based feature engineering frameworks such as CAAFE and OCT-Tree [4,6,29]. Using XGBoost also helps isolate the effect of feature engineering, since the comparison focuses on whether the proposed framework improves the input feature space rather than whether a different model architecture improves performance.

The proposed framework generated more than 400 candidate features during the full feature engineering process. To retain only useful features, an iterative feature selection strategy was applied. The dataset was divided into three chronological partitions: a training set for model fitting, a validation set used solely for feature selection decisions, and a held-out test set that was never accessed during feature generation or selection and was used only once for final performance reporting. Candidate features were evaluated one at a time using validation performance. A feature was retained only if it improved the validation Mean Absolute Error (MAE). Otherwise, it was discarded.

The Tesla experiment was evaluated using Mean Absolute Error (MAE) and coefficient of determination (R^2). MAE measures the average absolute difference between predicted and actual values:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of samples.

The R^2 metric measures the proportion of variance in the target variable explained by the model:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

where SS_{res} is the residual sum of squares and SS_{tot} is the total sum of squares.

The proposed framework was compared against two baselines: a raw-feature baseline using only valid available features, and the reported OCT-Tree result on Tesla stock prediction [4].

4.4. English Premier League Match Outcome Dataset

The second experiment evaluates the proposed framework on English Premier League match outcome prediction. The dataset was collected from football-data.co.uk, available at <https://www.football-data.co.uk> (accessed on 30 January 2026), and contains match-level records spanning the 2013 to 2023 seasons. It includes 4531 match samples, where each sample represents one EPL match. The dataset was split chronologically into training, validation, and test sets, with earlier seasons used for training and the most recent seasons reserved for testing, ensuring no future match information influences feature construction or model development.

The target variable is the Full-Time Result (FTR), which has three classes: home win, away win, and draw. This task is therefore treated as a three-class classification problem, a target commonly used in football outcome prediction studies allowing comparison with previous work [29,30].

The dataset includes the following variables: full-time home goals, full-time away goals, half-time results, attendance, referee, shots, shots on target, corners, fouls, offsides, yellow cards, and red cards, as well as pre-match betting odds for home win, draw, and away win outcomes. Variables were classified according to their temporal availability. Pre-match betting odds are treated as antecedent features, as they are available before the match result is observed. All in-match statistics—including goals, shots, corners, fouls, yellow cards, red cards, and offsides—are treated as consequent features, as they become available only after the match concludes and cannot be used directly as model inputs. However, their historical values from previous matches are used to generate valid historical aggregated features such as recent team performance statistics and referee-related historical patterns. Categorical variables including team names and referee identifiers were encoded to support aggregation within entity groups. Missing values arising from insufficient match history in early rows of the aggregation window are excluded from feature selection and model training.

4.5. EPL Experimental Model, Feature Selection, and Metrics

Multiclass logistic regression was used for the EPL experiment. This model was selected because it is computationally efficient, interpretable, and was used in previous EPL outcome prediction research involving Azure ML-based preprocessing [30], enabling a direct comparison between the proposed feature engineering framework and existing data-processing baselines.

The same iterative feature selection strategy used in the Tesla experiment was applied to the EPL dataset. However, because EPL match prediction is a classification task, accuracy was used as the validation criterion instead of MAE. Candidate features were retained if they improved validation accuracy.

The EPL experiment was evaluated using accuracy, precision, recall, and F1-score. The EPL match outcome prediction task involves three classes: home win, away win, and draw. Because the class distribution across these three outcomes is unequal, reporting accuracy alone is insufficient. Therefore precision, recall, and F1-score are also reported, calculated

using macro-averaging across all three classes so that each class contributes equally to the final metric regardless of its frequency in the dataset.

Accuracy measures the overall proportion of correctly classified matches across all three outcome classes:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

While intuitive, accuracy can be misleading when class distributions are imbalanced, as a model that predominantly predicts the majority class can still achieve moderate accuracy without genuine predictive capability across all three outcomes.

Precision measures, for each predicted outcome class, the proportion of matches predicted as that class that were correctly classified:

$$Precision = \frac{TP}{TP + FP}$$

High precision for a given class means the model rarely misclassifies other outcomes as that class. Low precision indicates a high rate of false positives for that class.

Recall measures, for each outcome class, the proportion of actual matches belonging to that class that were correctly identified:

$$Recall = \frac{TP}{TP + FN}$$

High recall means the model successfully identifies most matches of that outcome type. Low recall indicates that many matches of that class are missed and misclassified as another outcome.

The F1-score combines precision and recall into a single metric as their harmonic mean:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The harmonic mean penalises large imbalances between precision and recall, making F1-score a more informative metric than accuracy alone for multi-class classification tasks with unequal class frequencies. In the EPL context, a model that correctly identifies home wins but consistently misclassifies draws and away wins would show moderate accuracy but low F1-score, which is revealed through per-class precision and recall reporting.

For the multi-class EPL task, TP, FP, TN, and FN are defined per class using a one-versus-rest approach, where each outcome class is treated as the positive class in turn and the remaining two classes are treated as negative. Macro-averaged metrics are then computed as the unweighted mean across all three classes, ensuring that performance on draws and away wins is given equal weight to the more frequent home win outcome.

The proposed framework was compared with three baselines: a raw-feature baseline using only home, away, and draw betting odds; the reported Azure ML-based EPL prediction result [30]; and the reported C4.5-based EPL prediction result with discretisation and Particle Swarm Optimisation [29].

4.6. Summary of Experimental Design

To improve comparison fairness, all reimplemented baselines in this study were evaluated under a unified experimental protocol. For each dataset, the same chronological data split was applied across all methods, with the earliest records used for training and the most recent records reserved for testing. Feature sets for each baseline were constructed from the same raw input columns available to the proposed framework. Model architectures were kept consistent with those used in the proposed framework experiments: XGBoost for

the Tesla regression task and multiclass logistic regression for the EPL classification task. Results from the original OCT-Tree and Azure ML papers are retained in the comparison tables for reference but are clearly marked as reported results obtained under potentially different conditions as shown in Table 5.

Table 5. Summary of datasets and experimental settings.

Dataset	Domain	Task Type	Target Variable	Model	Metrics	Baselines
Tesla stock data [10]	Finance	Regression	Close price/percentage change	XGBoost	MAE, R^2	Raw-feature baseline; OCT-Tree [4]
English Premier League data	Sports analytics	Three-class classification	Full-Time Result	Multiclass logistic regression	Accuracy, precision, recall, F1-score	Odds-only baseline; Azure ML [30]; C4.5-based method [29]

This experimental design evaluates whether the proposed LLM-driven feature engineering framework can improve predictive performance across both regression and classification tasks. It also tests whether the framework can generate useful features in different domains while maintaining temporal validity.

5. Results and Discussion

5.1. Tesla Stock Prediction Results

The proposed framework generated 412 candidate features for the Tesla stock prediction task across all three generation stages. After iterative feature selection, 9 features were retained. The final retained features were:

open, log_open, high_3_avg, high_360_min, low_5_avg, low_5_previous, close_5_max, close_30_max, and volume_240_max.

Most of the retained features are historical aggregation features, indicating that past temporal patterns provide useful predictive signals for stock price prediction. This supports the motivation of the proposed framework, which aims to guide LLMs to generate time series-aware features rather than only single-row tabular transformations.

To compare the proposed method with OCT-Tree, the experiment was first conducted using closing price as the target variable. The proposed framework achieved an MAE of 5.2047, outperforming both the OCT-Tree result of 5.48 and the baseline MAE of 5.3465 (see Table 6). The proposed method also achieved an R^2 of 0.9733, compared with 0.9708 for the baseline.

Table 6. Tesla stock prediction results using close price as the target.

Model/Approach	R^2	MAE	Evaluation Conditions
Proposed framework	0.9733	5.2047	Unified protocol †
OCT-Tree [4]	Not reported	5.4800	Reported result ‡
Baseline using opening price only	0.9708	5.3465	Unified protocol †

† Evaluated on the same chronological data split, raw input columns, and XGBoost model as the proposed framework. ‡ Result reported in the original paper [4]; data split, feature set, and evaluation protocol may differ from this study.

The experiment was then repeated using daily percentage change as the target variable and projecting the predictions back to closing price for evaluation. This setting produced the best overall result, reducing MAE from 5.1261 for the baseline to 4.9963 for the proposed framework. The R^2 value was also slightly improved from 0.9698 to 0.9709 (see Table 7).

Table 7. Tesla stock prediction results using percentage change projected to close price.

Model/Approach	R^2	MAE
Proposed framework	0.9709	4.9963
Baseline using opening price only	0.9698	5.1261

These results show that the proposed framework improves prediction accuracy compared with the raw-feature baseline and the reported OCT-Tree result. Although the MAE reduction is modest, it is meaningful in the context of stock price prediction, where price movements are noisy and difficult to model using limited raw features. The retained features also show that historical aggregation is valuable for capturing temporal signals that are not available from current-day variables alone.

5.2. English Premier League Match Outcome Results

For the EPL match outcome prediction task, the proposed framework generated 318 candidate features across all three generation stages. After feature selection, the final retained features included implied betting probabilities, match weekday, betting odds, team-level historical statistics, and referee-related historical features.

Examples of retained features include:

HomeTeam_home_fouls_vs_hist_avg_3_previous, HomeTeam_home_red_vs_hist_avg_20_avg, AwayTeam_away_shot_conversion_3_avg, and Referee_is_home_win_20_avg.

These features demonstrate the ability of the framework to generate multi-step temporal features. For example, HomeTeam_home_fouls_vs_hist_avg_3_previous captures a team's recent foul behaviour compared with its historical average, then applies a previous-window aggregation to represent recent team discipline patterns.

The proposed framework achieved an accuracy of 57.86%, improving over the odds-only baseline accuracy of 56.45% and the Azure ML-based result of 57.09%. Recall also improved from 49.78% for the baseline to 50.10% for the proposed framework, while precision remained the same at 47.33% as shown in Table 8.

Table 8. EPL match outcome prediction results.

Model/Approach	Model	Accuracy	Recall	Precision	Evaluation Conditions
Azure ML preprocessing [30]	Multiclass logistic regression	57.09%	48.52%	Not reported	Reported result ‡
C4.5 + discretisation + PSO [29]	C4.5	71.05%	Not reported	Not reported	Reported result ‡
Baseline using home/draw/away odds	Multiclass logistic regression	56.45%	49.78%	47.33%	Unified protocol †
Proposed framework	Multiclass logistic regression	57.86%	50.10%	47.33%	Unified protocol †

† Evaluated on the same chronological data split, raw input columns, and multiclass logistic regression model as the proposed framework. ‡ Result reported in the original paper; data split, feature set, model configuration, and evaluation protocol may differ from this study and should be interpreted as a reference point rather than a direct comparison.

The results show that the proposed framework improves performance over the odds-only baseline and slightly exceeds the Azure ML-based preprocessing approach. This suggests that LLM-guided temporal feature engineering can add useful information beyond betting odds and standard preprocessing.

However, the C4.5 method with discretisation and Particle Swarm Optimisation achieved substantially higher accuracy. This suggests that the EPL dataset may benefit from distribution-aware discretisation, especially because each team has a limited number of matches per season. The current framework does not explicitly use data distribution information, such as quantiles or entropy-based cut-points, when generating features. This limitation indicates a clear direction for future improvement.

It should also be noted that the C4.5 + PSO approach may incorporate in-match statistics such as goals, shots, and corners directly as features, which our framework deliberately classifies as consequent variables unavailable at prediction time. This difference in feature availability assumptions means the comparison is not conducted on equal terms, and the higher accuracy of C4.5 + PSO may partly reflect access to post-event information rather than superior pre-match predictive capability.

5.3. Robustness Analysis

To evaluate the stability of the proposed framework, twenty independent subsampling trials were conducted. In each trial, 90% of the training data was randomly selected for model fitting, and performance was evaluated on the held-out test set. For the Tesla dataset, MAE and R^2 were used. For the EPL dataset, accuracy, precision, recall, and F1-score were used. Figure 7 shows the distribution of MAE and R^2 across 20 subsampling trials for the proposed framework and the baseline model.

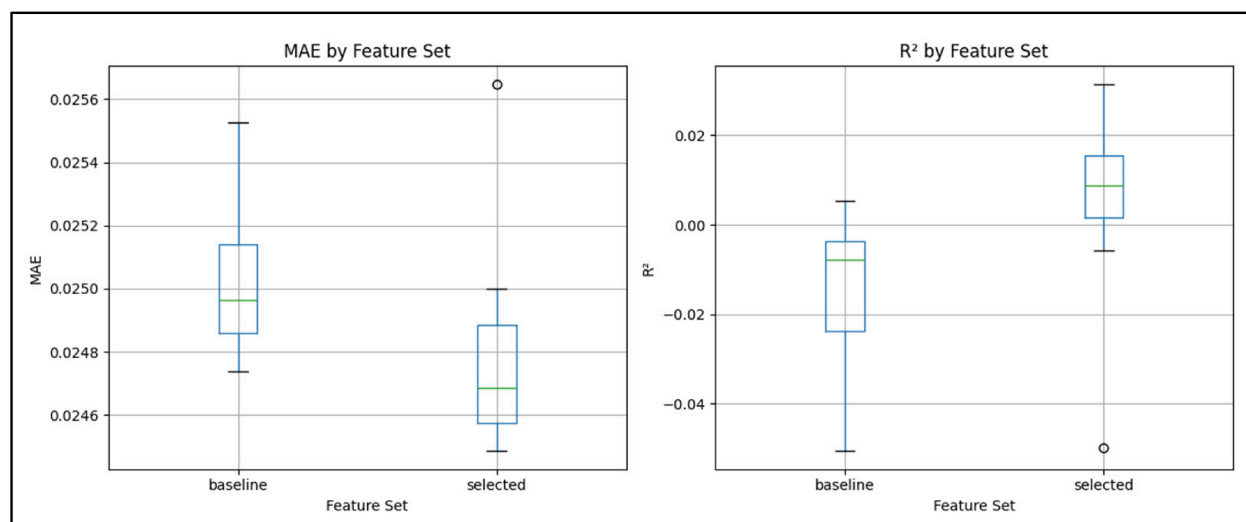


Figure 7. Robustness analysis on the Tesla stock prediction dataset.

The Tesla robustness results show that the proposed framework consistently achieved lower MAE and higher R^2 than the baseline across the repeated trials. The narrower interquartile range also suggests improved stability. Although one trial showed a noticeable deviation, the overall pattern indicates that the selected LLM-generated features improve both predictive accuracy and robustness.

Figure 8 shows the distribution of accuracy, F1-score, recall, and precision across 20 subsampling trials for the proposed framework and the baseline model.

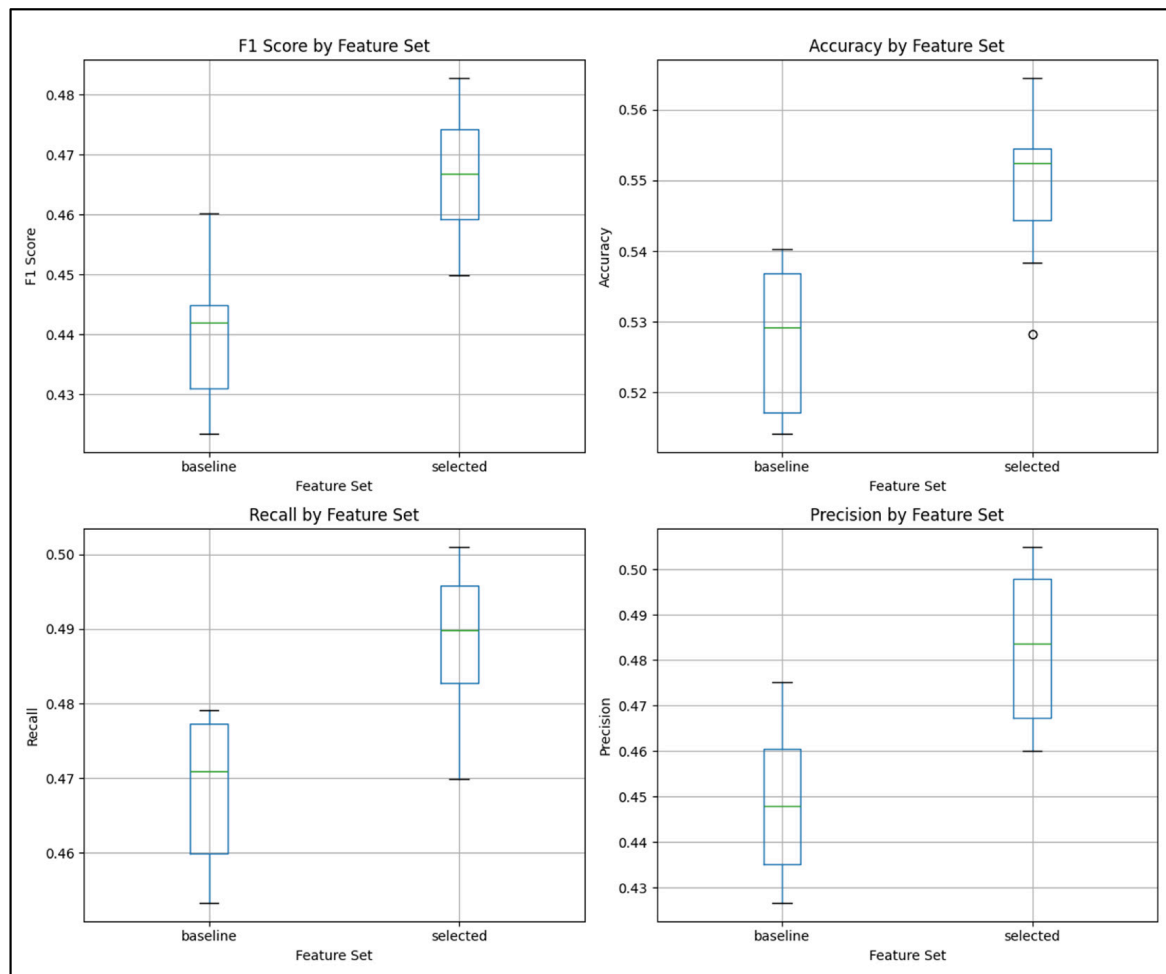


Figure 8. Robustness analysis on the EPL match outcome prediction dataset.

The EPL robustness results also show consistent improvement over the baseline across the repeated trials. The proposed framework achieved higher accuracy, F1-score, recall, and precision in the subsampling experiments. However, the stability improvement was less pronounced than in the Tesla experiment, suggesting that the smaller and more discrete nature of match-level sports data may limit the impact of automated historical feature engineering.

It should be noted that random subsampling does not strictly preserve temporal order and may allow future observations to appear in training folds. The robustness results should therefore be interpreted as indicative of general model stability rather than as a rigorous time series evaluation, and walk-forward validation is recommended as a more appropriate scheme for future work.

5.4. Overall Comparison and Discussion

It should be noted that the primary contribution of this study is the framework design and leakage-safe automation capability rather than claims of state-of-the-art predictive performance, and larger gains are expected on datasets with richer temporal structure and more entities.

Across both datasets, the proposed framework improved predictive performance compared with the raw-feature baselines. On the Tesla stock dataset, the proposed framework reduced MAE from 5.1261 to 4.9963, representing an improvement of approximately 2.5% over the baseline. It also outperformed the reported OCT-Tree result, which achieved an MAE of 5.48 on the Tesla stock prediction task [4]. On the EPL dataset, the proposed

framework improved accuracy from 56.45% to 57.86%, and also exceeded the reported Azure ML preprocessing result of 57.09% accuracy [30]. These results suggest that time series-aware LLM-generated features can improve model performance beyond raw-feature baselines and selected automated feature engineering methods.

The improvements are mainly explained by the framework's ability to generate historical aggregation features. Many retained features, such as `AwayTeam_away_shot_conversion_3_avg`, `HomeTeam_home_red_vs_hist_avg_3_avg`, and `volume_240_max`, were derived from previous time steps rather than current observations. This confirms the value of guiding the LLM to produce time series-specific features instead of only general tabular transformations.

Another important advantage of the proposed framework is its explicit separation of antecedent and consequent features. By allowing users to define which variables are available at prediction time, the framework prevents consequent variables from being used directly in the current prediction. At the same time, it allows their historical values to be used safely through aggregation. This design helps reduce data leakage and improves the reliability of model evaluation.

However, the results also show that the framework has limitations. In the EPL experiment, the C4.5 model with discretisation and Particle Swarm Optimisation achieved substantially higher accuracy than the proposed framework [29]. This suggests that small datasets may benefit from distribution-aware transformations, such as binning or adaptive cut-point selection. Since the current framework does not explicitly provide distributional information to the LLM, it may miss useful threshold-based features in small-sample settings.

6. Future Work

Although the proposed framework improved predictive performance on both datasets, several directions can further strengthen the approach.

First, future work should incorporate distribution-aware feature engineering. The EPL results suggest that discretised and threshold-based features can perform strongly, especially on smaller datasets. Therefore, future versions of the framework could provide the LLM with statistical summaries such as quantiles, variance, skewness, class distributions, and entropy-based cut-points. This may help the LLM generate more effective adaptive binning strategies and features for small or imbalanced datasets.

Second, the framework should be improved for scalability and efficiency. The current implementation relies mainly on row-wise processing in Pandas, which may become inefficient for larger datasets. Future work could use more scalable data-processing tools such as Polars, distributed computing frameworks, or cloud-based execution to support faster and parallel feature generation.

Third, future research should improve prompt stability and iterative feature refinement. Since LLMs may occasionally generate invalid or inconsistent configurations, an automatic validation module could be added to check generated outputs, detect errors, and request corrected configurations. In addition, validation scores, feature importance values, and model errors could be used as feedback for multiple rounds of LLM-based feature generation, allowing the framework to gradually refine the feature space and generate more useful temporal features.

Fourth, future work should evaluate the proposed framework across a broader range of time series datasets. The current study assesses the framework on a daily financial regression task and an event-driven sports classification task. These two settings differ in sampling frequency, entity structure, task type, and noise characteristics, but do not cover higher-frequency domains such as hourly energy consumption, sub-hourly healthcare monitoring, or multi-entity industrial sensor data. Evaluating the framework on datasets

with higher sampling frequencies, larger numbers of entities, and varying noise levels would more thoroughly characterise its generalisability and identify any domain-specific limitations that are not apparent from the current experiments. Public benchmarks such as the UCI Appliances Energy Prediction dataset and hourly bike-sharing demand datasets represent suitable candidates for this evaluation.

Replacing random subsampling with a walk-forward or rolling-origin evaluation scheme that strictly preserves temporal order would provide a more rigorous robustness assessment for time series tabular data and is identified as a priority improvement for future versions of the framework.

7. Conclusions

This study proposed a time series-aware LLM-driven feature engineering framework for tabular prediction tasks. The framework addresses a key limitation of existing LLM-based feature engineering methods by explicitly incorporating temporal constraints into the feature generation process. By separating variables into antecedent, consequent, and historical aggregated features, the framework allows LLMs to generate useful temporal features while reducing the risk of data leakage.

The proposed framework was evaluated on two datasets: Tesla stock prediction and English Premier League match outcome prediction. In the Tesla experiment, the framework reduced MAE compared with both the raw-feature baseline and the reported OCT-Tree result. In the EPL experiment, it improved accuracy compared with the odds-only baseline and the reported Azure ML preprocessing result. These findings show that LLM-guided temporal feature engineering can improve predictive performance across both regression and classification tasks.

The results also show that historical aggregation features are central to the framework's effectiveness. Many retained features were derived from past values of antecedent or consequent variables, confirming that time series-aware feature construction can reveal useful predictive signals that are not captured by raw features alone.

Despite these promising results, the framework can be further improved. In particular, future work should focus on distribution-aware feature generation, scalable implementation, prompt validation, richer temporal transformations, and iterative feature refinement. Overall, this study demonstrates that combining LLM reasoning with explicit temporal constraints is a practical and promising direction for automated feature engineering in time series tabular machine learning.

Author Contributions: Conceptualization, M.K.N. and S.S.C.; methodology, M.K.N.; software, M.K.N.; validation, M.K.N., B.N., T.K., S.S.C. and Z.S.; formal analysis, M.K.N.; investigation, M.K.N.; resources, S.S.C. and Z.S.; data curation, M.K.N.; writing—original draft preparation, M.K.N.; writing—review and editing, M.K.N., B.N., T.K., S.S.C. and Z.S.; visualization, M.K.N.; supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable, as this study did not involve human participants, animals, or any personally identifiable/private data. The study used publicly available secondary datasets only.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used in this study are publicly available. The relevant data sources and access links have been provided in the manuscript.

Acknowledgments: The authors used ChatGPT (GPT-5.2, OpenAI; April 2026) to assist with language editing. All content was reviewed and approved by the authors, who take full responsibility for the manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Shah, D.; Wang, J.; He, Q.P. Feature engineering in big data analytics for IoT-enabled smart manufacturing: Comparison between deep learning and statistical learning. *Comput. Chem. Eng.* **2020**, *141*, 106970. [CrossRef]
- Verdonck, T.; Baesens, B.; Óskarsdóttir, M.; vanden Broucke, S. Special issue on feature engineering editorial. *Mach. Learn.* **2024**, *113*, 3917–3928. [CrossRef]
- Ghojogh, B.; Samad, M.N.; Mashhadi, S.A.; Kapoor, T.; Ali, W.; Karray, F.; Crowley, M. Feature Selection and Feature Extraction in Pattern Analysis: A Literature Review. *arXiv* **2019**, arXiv:1905.02845. [CrossRef]
- Balasubramaniam, T.; Mohotti, W.A.; Sabir, K.; Nayak, R. Feature engineering on climate data with machine learning to understand time-lagging effects in pasture yield prediction. *Ecol. Inform.* **2025**, *86*, 103011. [CrossRef]
- Nam, J.; Kim, K.; Oh, S.; Tack, J.; Kim, J.; Shin, J. Optimized Feature Generation for Tabular Data via LLMs with Decision Tree Reasoning. *arXiv* **2024**, arXiv:2406.08527. [CrossRef]
- Hollmann, N.; Müller, S.; Hutter, F. Large Language Models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering. *arXiv* **2023**, arXiv:2305.03403. [CrossRef]
- ssalgadodev. Lag Features for Time-Series Forecasting in AutoML-Azure Machine Learning. Available online: <https://learn.microsoft.com/en-us/azure/machine-learning/concept-automl-forecasting-lags?view=azureml-api-2> (accessed on 17 June 2025).
- Daraghmeh, M.; Jararweh, Y.; Agarwal, A. Leveraging machine learning and feature engineering for optimal data-driven scaling decision in serverless computing. *Simul. Model. Pract. Theory* **2025**, *140*, 103090. [CrossRef]
- Zhang, C.; Niu, T.; Wei, Z.; Hu, X.; Pan, F.; Zeng, T.; Si, Y.; Chin, C.S. Lifespan prediction model for proton exchange membrane fuel cell vehicle based on time series information feature extraction and optimization. *Renew. Energy* **2025**, *243*, 122558. [CrossRef]
- Tesla (TSLA) Stock Data. Available online: <https://www.kaggle.com/datasets/guillemservera/tsla-stock-data> (accessed on 25 March 2025).
- Qin, J.; Lang, D.; Gao, C. Feature Extraction of Time Series Data Based on CNN-CBAM. In *Data Science*; Springer Nature: Singapore, 2023; pp. 233–245. [CrossRef]
- Höpfel, S.; Albadry, M.; Dahmen, U.; Herrmann, K.H.; Kindler, E.M.; König, M.; Reichenbach, J.R.; Tautenhahn, H.M.; Wei, W.; Zhao, W.T.; et al. Bayesian modelling of time series data (BayModTS): A FAIR workflow to process sparse and highly variable data. *Bioinformatics* **2024**, *40*, btac312. [CrossRef] [PubMed]
- Pourehbrahim, S.; Seifi, A.; Ehteram, M.; Hadipour, M.; Chen, J.E. The CEEMDAN-EWT-CNN-GRU-SVM model: A robust framework for decomposing non-stationary time series, extracting data features, and predicting solar radiation. *Results Eng.* **2025**, *25*, 104267. [CrossRef]
- Abdelghani, B.A.; Mohammad, A.A.; Dari, J.; Maleki, M.; Banitaan, S. Occupancy prediction: A comparative study of static and MOTIF time series features using WiFi Syslog data. *Sustain. Comput. Inform. Syst.* **2024**, *44*, 101040. [CrossRef]
- Wang, J.; Liu, J.; Jiang, W. An enhanced interval-valued decomposition integration model for stock price prediction based on comprehensive feature extraction and optimized deep learning. *Expert Syst. Appl.* **2024**, *243*, 122891. [CrossRef]
- Wong, A.; Li, E.; Le, H.; Bhangu, G.; Bhatia, S. A predictive analytics framework for forecasting soccer match outcomes using machine learning models. *Decis. Anal. J.* **2025**, *14*, 100537. [CrossRef]
- Lazcano, A.; Jaramillo-Morán, M.A. Data preprocessing techniques and neural networks for trended time series forecasting. *Appl. Soft Comput.* **2025**, *174*, 113063. [CrossRef]
- He, B.; Li, L.; Zhang, Q. PNC-based trend feature extraction method for time series data. *Neurocomputing* **2025**, *618*, 129174. [CrossRef]
- Wang, C.; Zhao, H.; Liu, Y.; Fan, G. Minute-level ultra-short-term power load forecasting based on time series data features. *Appl. Energy* **2024**, *372*, 123801. [CrossRef]
- Cai, W.; Yu, D.; Wu, Z.; Du, X.; Zhou, T. A hybrid ensemble learning framework for basketball outcomes prediction. *Phys. A Stat. Mech. Its Appl.* **2019**, *528*, 121461. [CrossRef]
- Neubauer, L.; Filzmoser, P. Improving forecasts for heterogeneous time series by ‘averaging’, with application to food demand forecasts. *Int. J. Forecast.* **2024**, *40*, 1622–1645. [CrossRef]
- Christ, M.; Braun, N.; Neuffer, J.; Kempa-Liehr, A.W. Time series feature extraction on basis of scalable hypothesis tests (tsfresh). *Neurocomputing* **2018**, *307*, 72–77. [CrossRef]

23. Liu, H.; Li, C.; Shao, Y.; Zhang, X.; Zhai, Z.; Wang, X.; Qi, X.; Wang, J.; Hao, Y.; Wu, Q.; et al. Forecast of the trend in incidence of acute hemorrhagic conjunctivitis in China from 2011–2019 using the Seasonal Autoregressive Integrated Moving Average (SARIMA) and Exponential Smoothing (ETS) models. *J. Infect. Public Health* **2020**, *13*, 287–294. [[CrossRef](#)] [[PubMed](#)]
24. Alvisi, S.; Ansaloni, N.; Franchini, M. Generation of synthetic water demand time series at different temporal and spatial aggregation levels. *Urban Water J.* **2014**, *11*, 297–310. [[CrossRef](#)]
25. Çelik, T.B.; İcan, Ö.; Bulut, E. Extending machine learning prediction capabilities by explainable AI in financial time series prediction. *Appl. Soft Comput.* **2023**, *132*, 109876. [[CrossRef](#)]
26. Rahayu, L.; Rusdiana, S.; Sofyan, H.; Kruba, R.; Mardalena, S. Dengue fever trends in Aceh using single moving average, weighted moving average and exponential smoothing. *AIP Conf. Proc.* **2024**, *3123*, 070008. [[CrossRef](#)]
27. Qu, Z.; Zhang, Z.; Qu, N.; Zhou, Y.; Li, Y.; Jiang, T.; Li, M.; Long, C. Extraction of typical operating scenarios of new power system based on deep time series aggregation. *CAAI Trans. Intell. Technol.* **2025**, *10*, 283–299. [[CrossRef](#)]
28. Schwartz-Ziv, R.; Armon, A. Tabular Data: Deep Learning is Not All You Need. *Inf. Fusion* **2022**, *81*, 84–90. [[CrossRef](#)]
29. Fuadi, M.B.A.; Alamsyah, A. Optimization of the C4.5 Algorithm Using Particle Swarm Optimization and Discretization in Predicting the Results of English Premier League Football Matches. *J. Adv. Inf. Syst. Technol.* **2022**, *4*, 126–138. [[CrossRef](#)]
30. Adnan, N.A.; Al Hakim Mohd Asri, L.; Mustapha, A.; Razali, M.N. The Football Matches Outcome Prediction for English Premier League (EPL): A Comparative Analysis of Multi-class Models. In *Recent Advances on Soft Computing and Data Mining*; Springer Nature: Cham, Switzerland, 2024; pp. 411–420. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.