FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

LLM-assisted AutoML for energy forecasting

Nuno Bazenga Martinho



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Dr. Ricardo Bessa

September 30, 2026

Resumo

O tema desta dissertação, desenvolvida com o apoio do INESC TEC tem como objetivo desenvolver um modelo de Machine Learning capaz de se adaptar aos diferentes casos de uso no domínio de Energia, consoante os parâmetros especificados pelo utilizador. Este processo será automático e aplicável a uma grande variedade de cenários propostos.

Com base nos avanços recentes na integração das LLM e na automação dos modelos de Machine Learning (AutoML), torna-se possível criar uma abordagem flexível e adaptável para o tratamento de séries temporais, independentemente do tipo de informação envolvida em cada caso de uso. Esta abordagem permite automatizar tarefas tradicionalmente complexas, como a seleção de modelos, a engenharia de variáveis e a otimização de hiperparâmetros.

Este avanço tecnológico reduz significativamente a necessidade de elevada experiência técnica na área de Machine Learning para o desenvolvimento de modelos preditivos de qualidade. Além disso, os engenheiros do setor da energia beneficiam de uma simplificação substancial do seu trabalho, uma vez que deixam de ter de investir grandes quantidades de tempo na investigação e implementação manual de modelos sempre que surge um novo pedido ou cenário de análise.

Este documento tem como objetivo explorar os avanços mais relevantes relativos ao estado da arte em LLM e AutoML, assim como determinar de que forma é que estes podem ser aplicados a metodologias tradicionais de previsão, a partir de vários testes realizados para os casos de uso de de energia solar e eólica produzida, assim como previsão de preços de mercado.

Abstract

The topic of this dissertation, developed with the support of INESC TEC, aims to develop a Machine Learning model capable of adapting to different use cases within the energy domain, according to user-specified parameters. This process will be fully automated and applicable to a wide variety of proposed scenarios.

Based on recent advances in the integration of Large Language Models (LLMs) and the automation of Machine Learning pipelines through AutoML, it becomes possible to create a flexible and adaptive approach for handling time series data, regardless of the type of information involved in each use case. This approach enables the automation of traditionally complex tasks, such as model selection, feature engineering, and hyperparameter optimization.

This technological advancement significantly reduces the need for high levels of technical expertise in Machine Learning to develop high-quality predictive models. Furthermore, engineers in the energy sector benefit from a substantial simplification of their work, as they no longer need to invest large amounts of time in the manual investigation and implementation of models whenever a new request or analysis scenario arises.

This dissertation document aims to explore the most relevant advances in the state of the art in LLMs and AutoML, and how this knowledge can be applied to improve traditional forecasting methodologies, by testing various forecasting scenarios with the different use cases considered : solar and wind power generation, and market price forecasting.

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisor, Ricardo Bessa , for his guidance, support, and availability throughout this dissertation project. His expert insights and constructive feedback were invaluable in improving the quality of this research, while his encouragement and willingness to discuss ideas helped me overcome the challenges encountered along the way. I am truly grateful for his time, patience, and dedication throughout this journey.

I would also like to express my sincere gratitude to my supervisor at INESC TEC, José Ricardo Andrade, for his unwavering commitment to this dissertation project from the very beginning of its development. His continuous support, valuable suggestions, and ability to propose better approaches to the challenges I encountered were crucial throughout this journey. I am especially grateful for his constant availability and willingness to help whenever I needed it, which remained unwavering until the very end of this project.

I want to thank INESC TEC, in particular to CPES (Center of Power and Energy Systems), that provided me the experience of working in such a good and healthy environment during these last months. This dissertation work wouldn't be possible to be presented without their scholarship opportunity, supported by Component 5 - Capitalization and Business Innovation, integrated in the Resilience Dimension of the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021 - 2026, within project ATE, with reference 56. For this I'm extremely grateful, since the whole experience was a distinct and valuable challenge for my future.

I would also like to express my deepest gratitude to my family for their unwavering support throughout my academic journey. Their constant belief in my abilities, encouragement, and the comfortable environment they provided enabled me to reach this final stage. Without their understanding, patience, and support, this journey would have been considerably more challenging.

Lastly, I would like to thank my friends for all the moments we shared throughout this period of my life. Their companionship greatly enriched my university experience, from the everyday moments of laughter to the unforgettable memories we created together. I am also deeply grateful for their constant encouragement and for always motivating me to pursue my goals.

UN Sustainable Development Goals

The UN Sustainable Development Goals aim to end poverty, protect the planet, and ensure peace and prosperity for all people by 2030, through 17 interconnected goals adopted by all UN member states in 2015. This dissertation contributes to these Development Goals by advancing forecasting technologies for energy systems, enabling more accurate predictions of solar power output, wind power generation, and electricity market prices, which directly supports the transition to sustainable and efficiently managed energy infrastructures. With this, the following Development Goals are covered by the work developed:

- SDG7: Affordable and Clean Energy
- SDG11: Sustainable Cities and Communities
- SDG12: Responsible Consumption and Production

In Table 1 it is possible to see how these goals are fulfilled by the presented dissertation project.

Table 1: Mapping of Sustainable Development Goals (SDGs) to Dissertation Contributions

SDG	**Goal**	**Dissertation contribution**
SDG7 - Affordable and Clean Energy	Ensure access to affordable, reliable and modern energy for all	The proposed pipeline improves the accuracy of solar and wind power forecasting by automatically extracting domain knowledge from research literature via LLMs and combining it with AutoML-driven feature and model selection. More accurate renewable energy forecasts reduce forecasting error in grid planning, supporting more reliable integration of clean energy sources and contributing to the reduction of dependence on fossil fuel-based reserves.
SDG11 - Sustainable Cities and Communities	Make cities and human settlements inclusive, safe, resilient and sustainable	The domain-agnostic design of the proposed approach (requiring no structural changes to operate across solar, wind, and electricity market forecasting) allows it to be deployed across diverse urban and community energy contexts without specialized reengineering. This adaptability makes accurate energy forecasting accessible to municipalities and communities that lack dedicated data science expertise, lowering the barrier to data-driven sustainable energy management.
SDG12 - Responsible Consumption and Production	Ensure sustainable consumption and production patterns	By forecasting day-ahead electricity market prices alongside renewable generation, the proposed system provides actionable signals for balancing energy consumption and production in real time. The automation of knowledge extraction and feature engineering reduces the human effort required to maintain and update forecasting models as energy systems evolve, supporting more responsive and responsible management of energy resources over time.

Contents

1 Introduction **1**
- 1.1 Objectives . . . 2
- 1.2 Research Questions . . . 2
- 1.3 Document Structure . . . 3

2 Background and State-of-the Art **4**
- 2.1 Time-Series Forecasting for the Energy Sector . . . 4
- 2.2 Forecasting Methods . . . 7
  - 2.2.1 Probabilistic and deterministic forecasting . . . 7
  - 2.2.2 Commonly used models . . . 8
- 2.3 Feature Engineering and Domain Knowledge . . . 10
  - 2.3.1 Common data sources in energy forecasting . . . 10
  - 2.3.2 Advanced Feature Engineering Techniques . . . 11
  - 2.3.3 Domain Knowledge . . . 12
- 2.4 AutoML for Time Series Forecasting . . . 13
  - 2.4.1 The AutoML Pipeline . . . 13
  - 2.4.2 AutoML frameworks applied to the energy domain . . . 14
  - 2.4.3 Challenges when applying AutoML for Time Series . . . 17
- 2.5 Extraction and Operationalization of Scientific Knowledge using LLMs . . . 18
  - 2.5.1 Large Language Models . . . 18
  - 2.5.2 Information Extraction . . . 19
  - 2.5.3 Code generation . . . 20
  - 2.5.4 Feature Engineering . . . 22
- 2.6 Hybrid LLM and AutoML Systems . . . 22
  - 2.6.1 Transforming Extracted Knowledge into AutoML Priors . . . 22
  - 2.6.2 LLM-based Agents in AutoML Systems . . . 23
  - 2.6.3 Limitations of current approaches . . . 26
- 2.7 Critical analysis . . . 26

3 Methodology **28**
- 3.1 Overview . . . 28
- 3.2 LLM-based knowledge extraction and validation . . . 29
  - 3.2.1 Information extraction agent . . . 31
  - 3.2.2 Development and validation agent . . . 34
  - 3.2.3 Integrity validation . . . 38
- 3.3 Time-series forecasting . . . 39
  - 3.3.1 AutoML Framework . . . 39
  - 3.3.2 From Knowledge Base to AutoML Priors . . . 40

3.3.3 Data Preparation and Feature Application . . . 40
3.3.4 Evaluation . . . 41

4 Experimental Setup & Results **44**
4.1 Overview . . . 44
4.2 Forecasting use cases & datasets . . . 44
4.2.1 Wind energy (Sotavento experimental wind farm) . . . 45
4.2.2 Solar energy (INESC TEC) . . . 45
4.2.3 Electricity Prices (MIBEL) . . . 46
4.3 Knowledge base . . . 47
4.3.1 Papers per use case . . . 47
4.4 LLM providers . . . 47
4.4.1 Local models . . . 48
4.4.2 Groq . . . 48
4.5 Time-series forecasting setup . . . 48
4.5.1 Baseline selection . . . 48
4.5.2 Regression models selection . . . 48
4.6 Prototype . . . 49
4.7 Results - Research Question 1 . . . 49
4.7.1 Quality of results . . . 49
4.7.2 Response Integrity Assessment . . . 52
4.7.3 Adaptability . . . 53
4.7.4 Conclusions . . . 54
4.8 Results - Research Question 2 . . . 55
4.8.1 Evaluation baselines . . . 55
4.8.2 Solar power forecasting . . . 56
4.8.3 Wind Power Forecasting . . . 59
4.8.4 Electricity Market Prices Forecasting . . . 63
4.8.5 Competitiveness . . . 67
4.8.6 Computational effort . . . 68
4.8.7 Conclusions . . . 70

5 Conclusions & Future Work **72**
5.1 Summary of findings . . . 72
5.2 Limitations . . . 73
5.3 Future work . . . 74

References **76**

A Code execution and validation **83**

B Used Prompts for the LLM **87**
B.1 Feature-Function Extraction Prompt . . . 87
B.2 Regression-Model Extraction Prompt . . . 91
B.3 Feature-Function Adaptation Prompt . . . 93
B.4 Cross-Domain Applicability Prompt . . . 94

C Considered research documents for each use case **95**
C.1 Solar Power Forecasting 95
C.2 Electricity Market Prices Forecasting 97
C.3 Wind Power Forecasting 98

D Pipeline of all system performed tasks **99**

List of Figures

2.1 Different forecasting horizons for electrical load prediction [2]. 4
2.2 Forecasting horizons considered for market price forecasting [3] 6
2.3 Structure of probabilistic forecasting [6] 7
2.4 Structure of deterministic forecasting [6] 7
2.5 Classification Framework for Energy Forecasting Models [8] 8
2.6 Integration of domain knowledge throughout the Machine Learning workflow [23] 12
2.7 Typical pipeline of an AutoML process, adapted from [28] 14
2.8 Flowchart of the AutoML-based building energy load prediction method [30] . . 14
2.9 Auto-sklearn workflow [31] 15
2.10 TPOT workflow [31] 15
2.11 Testing error on energy use appliance consumption data with different training time [31] 16
2.12 Overall workflow of the LLM extraction, including the Data preparation and Data extraction phases [42] 20
2.13 Comparison of different approaches for constructing a deep learning model for subsurface flow [53]. 23
2.14 Example of a system prompt used to condition the LLM-based Model Agent for model selection and specification [54]. 24
2.15 General overview of the *AutoML-Agent* framework, including all the steps from the RAG phase to extract information from external sources until the final validation with LLM [54]. 25

3.1 Flowchart of all performed processed steps for Methodology 28
3.2 Extraction and validation workflow 29
3.3 Structured JSON output formats exchanged between agents, in a form of a dictionary: model schema (left) and function schema (right). 30
3.4 LLM-Agent workflow considered for the LLM tasks performed for extraction and validation 31
3.5 Steps performed by the validation agent to adapt, execute, and screen each candidate function 35
3.6 Example of a generated python implementation for a lagged feature 38
3.7 Workflow of the data integrity assessment 39
3.8 TPOT tree-based pipeline. Each circle corresponds to a machine learning operator, and the arrows indicate the direction of the data flow [67] 40
3.9 Time-series cross-validation logic used for the baselines considered 41
3.10 Proposed approach and available baselines for comparison 42

4.1 Communication operations with the LLM server 47

4.2 Example of an extracted function output, based on the information available in [71] 49
4.3 Example of an extracted model output, based on the information available in [71] 50
4.4 Predicted values (in red) vs actual values (in black) for generated power (in W), in the last fold of evaluation for cross-validation 58
4.5 Predicted values (in red) vs actual values (in black) for generated power (in W), in the last fold of evaluation for cross-validation 59
4.6 Predicted values (in red) vs actual values (in black) for power generation (in kW) in the last fold of evaluation for cross-validation 61
4.7 Predicted values (in red) vs actual values (in black) for generated power (in kW) for the last fold of evaluation in cross-validation 62
4.8 Predicted values (in red) vs actual values(in black) for market price (in €/MWh) for the last fold of evaluation in cross-validation 66
4.9 Predicted values (in red) vs actual values(in black) for market price (in €/MWh) for the last fold of evaluation in cross-validation 67

D.1 Diagram that represents the overall workflow from information extraction and validation until the final evaluation with AutoML. 99

List of Tables

1 Mapping of Sustainable Development Goals (SDGs) to Dissertation Contributions v

2.1 Comparison of basic functions among six representative AutoML frameworks, adapted from [30] . . . 15
2.2 Summary of the main hallucination causes, detection methods, and mitigation strategies, adapted from [39]. . . . 19
2.3 Comparison of ML and Software Engineering Benchmarks across AI Models and Scaffolds . . . 21
2.4 Comparison of LLM-based AutoML agent architectures . . . 25
2.5 Comparison Matrix of Documents and Sources Across Domains 1–8 . . . 27

3.1 Function attributes extracted by the **information extraction agent** . . . 33
3.2 Model attributes extracted by the **information extraction agent** . . . 34
3.3 Function attributes filled directly in the feature validation phase . . . 37

4.1 Raw variables available for the wind energy generation dataset . . . 45
4.2 Raw variables available for the solar energy production dataset considered . . . 46
4.3 Raw variables available in the MIBEL dataset considered . . . 46
4.4 Research literature read by each LLM and generated features considered for testing 51
4.5 Valid model and hyperparameter configurations found in all the models extracted from the research documents . . . 51
4.6 Description-implementation gap: among artefacts with a Perfect or Good description, the share whose code implementation was rated Inaccurate . . . 53
4.7 Baseline Scenario Performance Comparison Across Multi-Domain Metrics . . . 55
4.8 RMSE and MAE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 . . . 57
4.9 Computational time, MAE and RMSE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 results, and computational time of the whole pipeline (s) . . . 58
4.10 RMSE and MAE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 . . . 61
4.11 Computational time, MAE and RMSE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 results, and computational time of the whole pipeline (s) . . . 62
4.12 Model Performance Evaluation with different sets of input features and respective improvement, in percentage, in comparison to the baseline results . . . 65
4.13 Computational time, MAE and RMSE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 results, and computational time of the whole pipeline (s) . . . 66

4.14 Computational effort of the proposed approach across use cases and search-space conditions, compared with the baseline scenarios 69

C.1 Literature Matrix for Solar Power Forecasting 95
C.2 Literature Matrix for Electricity Market Price Forecasting 97
C.3 Literature Matrix for Wind Power Forecasting 98

List of Acronyms

ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
AutoML	Automated Machine Learning
CAAFE	Context-Aware Automated Feature Engineering
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DER	Distributed Energy Resource
DIML	Domain-Informed Machine Learning
FFT	Fast Fourier Transform
GEFCom	Global Energy Forecasting Competition
GP	Genetic Programming
HPO	Hyperparameter Optimization
JSON	JavaScript Object Notation
LLM	Large Language Model
LR	Linear Regression
LSTM	Long Short-Term Memory
LTF	Long-Term Forecasting
MAE	Mean Absolute Error
MIBEL	Mercado Ibérico de Electricidade
ML	Machine Learning
MLP	Multi-Layer Perceptron
MTF	Medium-Term Forecasting
NER	Named Entity Recognition
NWP	Numerical Weather Prediction
PV	Photovoltaic
RE	Relation Extraction
RF	Random Forest
RLHF	Reinforcement Learning from Human Feedback
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SCADA	Supervisory Control and Data Acquisition
SFT	Supervised Fine-Tuning
SMAC	Sequential Model-based Algorithm Configuration
STF	Short-Term Forecasting
SVM	Support Vector Machine
TPOT	Tree-based Pipeline Optimization Tool
VPN	Virtual Private Network
VSTF	Very-Short-Term Forecasting
WRF	Weather Research and Forecasting

Chapter 1

Introduction

Modern power systems no longer exhibit a single one-way flow from the grid to the consumer. The inclusion of renewable sources (e.g., wind and solar) and distributed energy resources (DERs) (e.g., rooftop solar panels, electric vehicles and batteries) has transformed the energy landscape. This shift toward decentralization and intermittency has made power systems significantly more complex, with increasingly dynamic and unpredictable behavior. The variability of renewable generation, when combined with fluctuating demand patterns and market dynamics, has introduced new challenges for system operators and energy planners who must maintain grid stability, optimize resources, and minimize operational costs in real time. To address these challenges, time series forecasting has become a core tool for supporting operational decision-making under uncertainty. As highlighted in [1], forecasting methods are predominantly engineered and maintained through manual processes, relying extensively on expert knowledge and human intervention, with practices that vary across application domains. Traditional methods are prone to errors, time-consuming and require detailed data pre-processing, manual engineering of features, and domain-specific expertise. One of the identified disadvantages of these models for forecasting tasks was the wasted time and the lack of scalability when applied to large datasets, particularly in energy portfolios that aggregate multiple time series for the final scheduling to the energy market. Network Operators, suppliers and aggregators demand reliable and adaptive forecasts for power consumption, renewable generation and energy market prices. To overcome these limitations, this dissertation proposes an autonomous pipeline that combines Large Language Models (LLMs) with Automated Machine Learning (AutoML). This pipeline first extracts and structures knowledge from state-of-the-art technical literature such as research papers or technical reports. This information is then used to configure an AutoML framework with literature informed priors, reducing the search space for the model optimization processes. This integration will enable an automatic discovery and adjustment of base models and feature engineering processes to specific use cases, enhancing both scalability and adaptability of these tools.

1.1 Objectives

The main objective of this dissertation is to develop an autonomous forecasting framework for the energy sector that combines the capability of information extraction associated with Large Language Models (LLMs) with further evaluation, considering an Automated Machine Learning (AutoML) framework.

LLM agents are initially used to extract and structure domain knowledge from technical literature and convert it into machine-readable priors that guide AutoML. Then, given a user-defined forecasting specification (e.g., target variable, horizon, and available data), the framework aims to automatically produce a task-appropriate predictive model by combining AutoML optimization with domain knowledge extracted by the LLM.

The workflow of this autonomous system is organized into specialized LLM-agents, each responsible for a distinct function within the pipeline. The specific tasks assigned to these agents are as follows:

- **Read and process domain-specific literature** to extract information on features and regression models used in relevant research studies.
- **Generate code for feature engineering techniques** based on the information collected from the literature.
- **Test the extracted features and models**, evaluating whether, using this information, it is possible to create robust supervised learning models for energy forecasting.

This structured knowledge extracted by the LLM agents is integrated as priors into a given AutoML framework. By constraining the candidate feature sets, learning algorithms and hyperparameter ranges, these priors will reduce the search space of the AutoML optimization processes, focusing on configurations that are plausible to work for the specified forecasting use cases. This ultimately can lower computational overhead and development effort, and improve the consistency of model selection, particularly for use cases where domain expertise is limited or not available.

In the end, this autonomous forecasting framework produces a forecasting pipeline grounded in state-of-the-art information and then specializes it to the target datasets (e.g., of specific wind farms) through AutoML optimization routines, ultimately producing a model tailored to the characteristics of the specific energy system, site, or portfolio, built with minimal domain expertise.

1.2 Research Questions

Associated with this topic, two Research Questions are raised, to check if the proposed system can really solve the problems that researchers of the domain face nowadays.

- **Research Question 1 (RQ1)**: The first Research Question that can be asked for this developed work is: "An approach based in LLMs and AutoML can minimize human intervention, inferring domain knowledge and experience from existing literature automatically?"

- **Research Question 2 (RQ2)**: "Will that approach be competitive, in implementation time and forecasting quality, compared to traditional methods?"

In chapter 4, these Research Questions will be addressed, with the obtained results from this dissertation work.

1.3 Document Structure

In addition to this introductory chapter, this dissertation is organized into four main chapters. Chapter 2 presents the background and the state of the art relevant to the dissertation topic. Chapter 3 describes the methodology adopted throughout the development of this work. Chapter 4 details the experimental setup and discusses the results obtained. Finally, Chapter 5 presents the main conclusions of this research and outlines directions for future work. An appendix is also included to provide supplementary material referenced throughout the document.

Chapter 2

Background and State-of-the Art

2.1 Time-Series Forecasting for the Energy Sector

Time-series forecasting plays a central role in modern energy systems, particularly with the increasing penetration of intermittent renewable energy sources and the growing complexity of electricity markets. By leveraging historical observations, such as electricity demand, generation profiles, market prices, and meteorological variables, energy forecasting enables the anticipation of future system states. Accurate forecasts are essential for maintaining grid stability, supporting efficient market operation, optimizing asset utilization, and reducing overall operational and investment costs.

Energy forecasting problems are commonly categorized according to their prediction horizon, as different time scales correspond to distinct decision-making processes within power systems. These horizons range from very-short-term forecasts used for real-time control to long-term forecasts supporting strategic planning.

Figure 2.1, adapted from [2], provides an overview of the different forecasting horizons commonly considered for electrical load prediction.

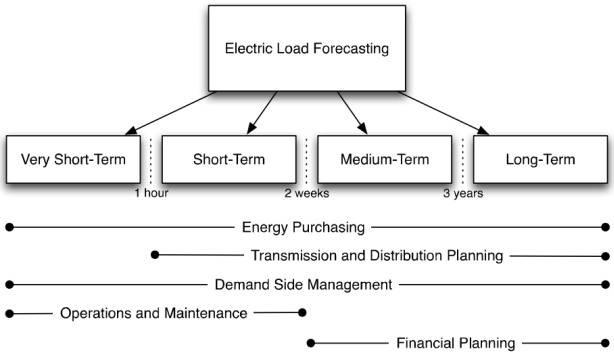


Figure 2.1: Different forecasting horizons for electrical load prediction [2].

Very-short-term forecasting (VSTF) typically covers time horizons ranging from a few minutes up to one hour ahead. These forecasts are primarily used for real-time grid control, frequency

regulation, voltage control, and balancing services. In renewable-rich systems, very-short-term forecasts are particularly important for tracking rapid fluctuations in wind and solar generation caused by sudden changes in wind speed or cloud cover. High-resolution data and fast-updating models are essential at this horizon.

Short-term forecasting (STF) generally spans from one hour ahead to several days, with day-ahead forecasting being especially critical in energy systems. Short-term forecasts support operational decisions such as unit commitment, economic dispatch, reserve allocation, and electricity market clearing. Inaccurate short-term load or generation forecasts can lead to inefficient dispatch, increased reserve requirements, or system reliability issues. For this reason, short-term forecasting has been one of the most extensively studied problems in energy analytics.

Medium-term forecasting (MTF) typically ranges from several weeks to a few months ahead and is mainly used for maintenance planning, fuel procurement, hydro reservoir management, and assessment of system adequacy under seasonal demand variations. At this horizon, forecasts must capture broader consumption trends, seasonal patterns, and expected weather conditions, while tolerating a higher degree of uncertainty than short-term predictions.

Long-term forecasting (LTF) extends from several months to years ahead and plays a strategic role in power system planning. Long-term forecasts inform decisions related to generation and transmission expansion, integration of renewable energy resources, policy evaluation, and long-term demand-side management. Unlike short-term forecasts, long-term predictions often rely on aggregated data, scenario analysis, and socio-economic indicators rather than high-frequency measurements.

Across these forecasting horizons, the complexity of energy data (characterized by non-stationarity, strong seasonality, weather dependency, and human behavioral influences) poses significant modeling challenges. In particular, renewable energy generation exhibits high volatility due to atmospheric dynamics, while electricity demand is influenced by socio-economic activity, calendar effects, and consumption behavior.

To address these challenges, a wide range of artificial intelligence and hybrid modeling approaches have been widely adopted, especially for short- and very-short-term forecasting. Common techniques include Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and Adaptive Neuro-Fuzzy Inference Systems (ANFIS), which are capable of capturing nonlinear relationships and complex temporal dependencies [2]. More recently, automated modeling approaches have further enhanced forecasting performance across multiple horizons.

As for market price forecasting, a study exploring the potential of extending data-driven models to different forecasting horizons by [3] classifies predictions into short, mid, and long-term horizons. This classification is used to explain how data-driven approaches, such as machine learning and artificial intelligence models, make use of historical data for electricity price forecasting. In contrast, fundamental models incorporate market mechanisms that directly influence price formation. For long-term forecasting, it is necessary to consider the expected market conditions and the composition of the power plant fleet within a given geographic zone. In this context,

electricity prices are derived either from the intersection of supply and demand curves or from the shadow prices associated with demand constraints.

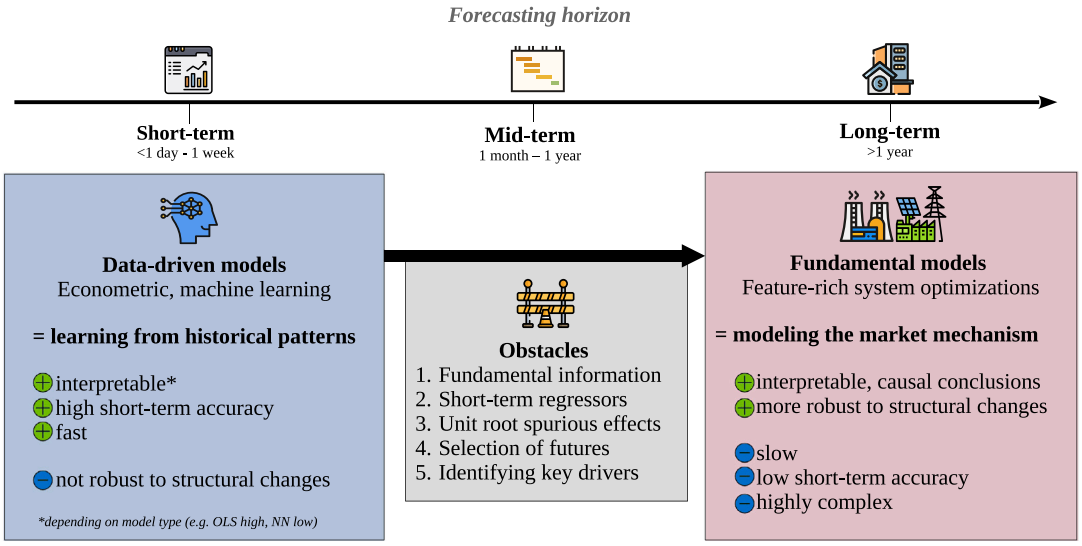


Figure 2.2: Forecasting horizons considered for market price forecasting [3]

In the domain of renewable energy forecasting, a study on deep learning for solar nowcasting by [4] defines forecasting horizons in a distinct way through the inclusion of nowcasting, which enables the prediction of power production within a maximum horizon of 6 hours. This is possible due to the use of sky images, ground sensors, satellites, and other real-time measurements. Unlike short-term (6–72 hours) and day-ahead forecasting (>72 hours), these predictions do not rely on Numerical Weather Prediction (NWP) models.

For wind forecasting, [5] presents a multi-horizon comparison, dividing forecasting horizons into ultra-short, short, medium, and long-term. The ultra-short-term horizon is particularly important for the management of spinning reserves, as well as for anticipating power output and turbine rotational speed. Short-term forecasting (STF), unlike what is commonly stated for load forecasting, is usually considered useful only up to 24 hours ahead. These short-term studies typically rely on high-resolution SCADA measurements, with records collected every 1 to 10 minutes, complemented by local weather observations, similarly to solar forecasting.

For medium-term forecasting, the authors consider prediction horizons ranging from 2 hours to one week ahead. At this stage, the use of NWP models is introduced. Long-term forecasting supports strategic grid operation management, maintenance scheduling, and cost–benefit analysis. This characterization is also common in solar forecasting studies.

Although the definition of forecasting horizons may vary slightly across research papers, especially in the renewable energy domain, many underlying concepts and operational objectives remain consistent.

2.2 Forecasting Methods

2.2.1 Probabilistic and deterministic forecasting

Energy forecasting techniques fall into two broad categories: deterministic and probabilistic forecasting.

Deterministic forecasting aims to predict a single expected value of the target variable for a given forecasting horizon, while probabilistic forecasting estimates the uncertainty associated with future values by producing prediction intervals or probability distributions. Figure 2.3 and Figure 2.4 illustrate the general structure of both forecasting paradigms, using wind forecasting as an example [6].

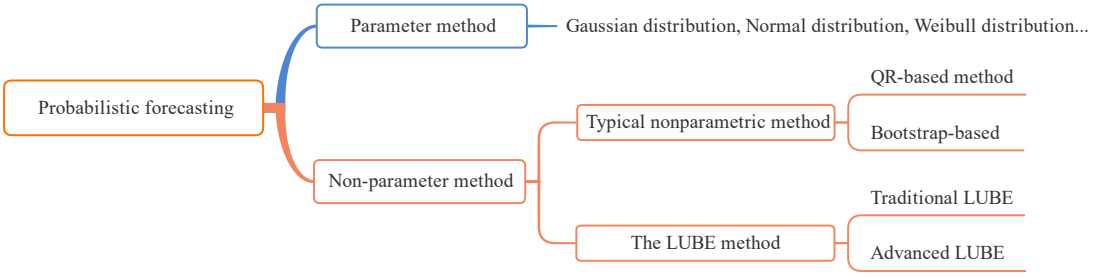


Figure 2.3: Structure of probabilistic forecasting [6]

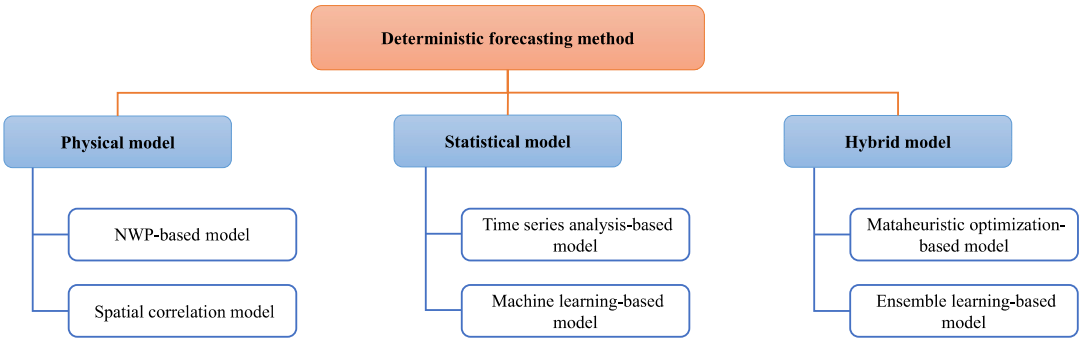


Figure 2.4: Structure of deterministic forecasting [6]

Deterministic forecasting methods generate point forecasts by learning the relationship between historical observations and future values. According to [6], these methods can be broadly categorized into physical, statistical, and hybrid models. Since this dissertation focuses on deterministic forecasting, these categories are discussed in greater detail in subsection 2.2.2.

Probabilistic forecasting grant additional information that enables the quantification of forecast uncertainty, making probabilistic approaches particularly useful in scenarios characterized by high variability, such as renewable energy generation and electricity demand [7].

Despite both forecasting paradigms being widely used in energy applications, this dissertation focuses on deterministic forecasting because its objective is to predict exact values of the target variables rather than to estimate the uncertainty associated with those predictions.

2.2.2 Commonly used models

Energy forecasting methodologies have evolved considerably over the past decades, progressing from statistical approaches to physics-based models, machine learning methods, transformer architectures, and, more recently, foundation models. Figure 2.5 summarizes this evolution together with the taxonomy of energy forecasting proposed by [8]. In this picture it is also possible to see the relations between different forecasting horizons, learning paradigms, data types, and model architectures, highlighting how foundation models build upon these concepts to provide a unified forecasting framework.

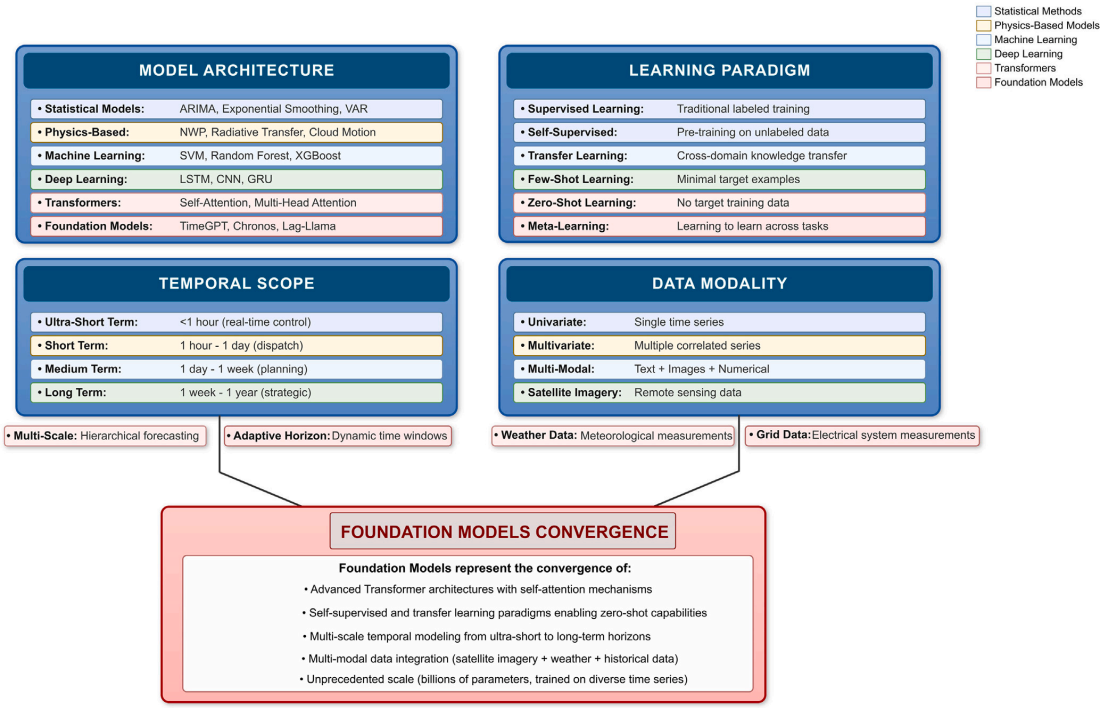


Figure 2.5: Classification Framework for Energy Forecasting Models [8]

As depicted in Figure 2.5, a wide range of approaches can be used for forecasting tasks, from traditional statistical models, to advanced modern hybrid architectures. These models can then be split into different main categories:

- **Statistical models**: traditional forecasting approaches, such as ARIMA (Autoregressive Integrated Moving Average) and persistence models, rely on historical observations and statistical relationships to predict future values. While computationally efficient and easy to interpret, they often struggle to capture the nonlinear and non-stationary behaviour inherent

in modern energy systems, resulting in degraded performance for complex forecasting tasks [9].

- **Physical models**: estimate the target variable by explicitly modelling the physical processes governing the energy system. Instead of relying on historical observations, they use physical performance parameters and environmental conditions (e.g., turbine characteristics and atmospheric conditions in wind power forecasting) to estimate the expected output. According to [10], these models can be divided into:
  - **Analytical diagnostic models**: based on boundary layer parameterization schemes and simplified flow assumptions, making them suitable for flat or homogeneous terrain but less effective in representing complex dynamic flow variations.
  - **Numerical simulation models**: employ Computational Fluid Dynamics (CFD) together with Numerical Weather Prediction (NWP) outputs to simulate dynamic flow evolution, providing higher forecasting accuracy under complex terrain and atmospheric conditions.

  Although physical models do not require historical generation data, their computational cost is typically high, limiting their application to localized forecasting problems such as individual wind or solar farms.

- **Machine learning models**: methods such as Random Forests (RF), Support Vector Machines (SVM) and Linear Regression (LR) learn forecasting relationships directly from historical data. These models provide robust baseline performance, relatively low computational complexity and good predictive capability when sufficient engineered features are available [11].

- **Deep learning models**: neural network architectures, including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs) and Transformer-based models, automatically learn hierarchical representations from data, enabling them to capture nonlinear relationships, temporal dependencies and long-range interactions. Hybrid architectures, such as the Sustainable Energy Forecasting Network (SEF-Net), combine these models to further improve forecasting performance, particularly for medium- and long-term forecasting tasks [12, 13].

- **Foundation models**: represent the latest generation of forecasting approaches, consisting of large-scale pre-trained models capable of supporting multiple forecasting horizons and energy domains [8]. By transferring knowledge learned from diverse datasets, they enable capabilities such as zero-shot forecasting for newly deployed renewable energy installations with limited historical data. Despite their impressive generalization capabilities, foundation models do not replace specialized forecasting approaches. Their training and deployment require substantial computational resources, and their performance may deteriorate when the target domain differs significantly from the data used during pre-training. Moreover,

when sufficient domain-specific data are available, specialized forecasting models can often achieve comparable or superior performance while being considerably more computationally efficient and easier to adapt to application-specific requirements [14].

Overall, the evolution of energy forecasting reflects a gradual transition from simple statistical techniques towards increasingly data-driven and generalizable approaches. While modern deep learning and foundation models have significantly advanced forecasting accuracy and scalability, no single model family is universally optimal.

For this dissertation, Machine Learning (ML) models were selected for the forecasting tasks due to their adaptability and suitability across the different use cases evaluated, while providing deterministic predictions. Although deep learning and foundation models could also be applied to the proposed forecasting tasks, they generally require significantly greater computational resources and training time, particularly when training a dedicated model for each scenario considered. In addition, for the purposes of this work, conventional ML models have demonstrated competitive performance when combined with relevant engineered features [11], making them a suitable choice for the forecasting tasks addressed.

Physical models were also considered indirectly, as the forecast values available in the datasets considered (see section 4.2) were generated using such models.

2.3 Feature Engineering and Domain Knowledge

2.3.1 Common data sources in energy forecasting

As stated in a study conducted by [15], with the aim of creating a Global Energy Forecasting Competition, energy forecasting relies on different data sources, depending on the specific use case considered, forecasting horizon and energy subsystem under consideration (e.g load demand, wind power, or solar generation). Generally, the most important inputs include historical measurement data, such as past electricity demands, generation output, or system states, which capture temporal patterns including trends, seasonality and autocorrelation structures.

In addition to historical measurements, numerical weather prediction (NWP) also plays a central role, particularly for renewable energies forecasting. In recent research [16], the forecasting impact of the inclusion of meteorological variables such as the temperature, relative humidity, visibility, air pressure, wind speed was explored. For example, in some specific applications, such as wind power forecasting, it's expected that the wind speed has more impact that other inputs such as the irradiance or temperature, which are more relevant in solar and load forecasting.

The inclusion of standard calendar variables is also important for improving the forecasting accuracy in energy applications, since they reveal patterns in the data. In a study of the energy market, the authors of [17] confirmed this by using variables such as day-of-the-week, Hour-of-the-day, Month-of-the-year, Day-of-the-month and season.

2.3.2 Advanced Feature Engineering Techniques

2.3.2.1 Feature Engineering

Feature engineering refers to the process of transforming raw data into informative input variables (features) that better represent the underlying problem for machine learning or statistical models. The main goal of feature engineering is to learn meaningful patterns and relationships from data.

In the context of time series forecasting, feature engineering plays a crucial role, as raw data often needs to be restructured into representations that capture temporal dynamics, seasonality, and underlying system behaviour. Without domain-specific knowledge of the target application, several general-purpose feature engineering techniques can be applied, as discussed by [18]. These include:

- **Statistical features**: basic descriptive statistics computed over the data, such as mean, standard deviation, minimum, maximum, or higher-order moments. These features provide a compact summary of the local or global distribution of the signal.
- **Lag and lead features**: temporal shift-based transformations where past observations (e.g., $t-1, t-2, ..., t-n$) are used as predictors for future values (e.g., $t+1$). In some applications, future exogenous variables (e.g., meteorological forecasts) can also be incorporated as lead features.
- **Cyclical encodings**: transformations used to represent periodic variables such as hour of day, day of week, or month. Sine and cosine transformations are typically applied to preserve the circular nature of these variables. This approach can also be extended to other signals such as wind direction [18].
- **Differencing**: transformation that computes the change between consecutive or lagged observations, often used to reduce trend and improve stationarity in time series data.
- **Rolling window features**: statistics computed over a moving window (e.g., rolling mean, minimum, maximum, or variance), which capture local temporal dynamics and short-term trends in the data.

Beyond these classical approaches, more advanced feature engineering techniques have also been explored in the literature. For example, [19] employ Fast Fourier Transform (FFT)-based features to extract spectral information from meteorological time series, improving the prediction of solar radiation by capturing periodic frequency components in the data.

2.3.2.2 Feature Selection

Following feature engineering, feature selection is a crucial step to identify the most relevant variables and reduce the dimensionality of the input space considered. According to [20], feature selection methods can be broadly categorized into three main groups: filter, wrapper, and embedded methods. Filter methods rank features based on statistical or information-theoretic criteria

independently of any learning algorithm, while wrapper methods evaluate subsets of features by directly optimizing the performance of a predictive model. Embedded methods, in contrast, integrate feature selection into the model training process, often through regularization techniques or model-specific mechanisms that promote sparsity.

In addition to these traditional approaches, feature selection can also be automated within machine learning pipelines. For instance, [21] propose a Context-Aware Automated Feature Engineering (CAAFE) framework, which leverages automated machine learning techniques for tabular data. Beyond generating new features through code synthesis, the approach also provides explanations regarding the relevance and usefulness of the selected features, contributing to both automation and interpretability in the feature engineering process.

2.3.3 Domain Knowledge

Domain knowledge refers to structured prior information about a problem domain that is not directly observable from data alone. Such knowledge may include physical laws, empirical heuristics, causal relationships, or design constraints.

The integration of domain knowledge into forecasting models is essential for improving accuracy beyond purely data-driven approaches. Domain-informed machine learning (DIML) leverages physical principles, operational constraints, and meteorological insights to refine feature selection and model architecture [22].

The following graph from [23], illustrates the workflow that is associated with Domain Knowledge in Machine Learning.

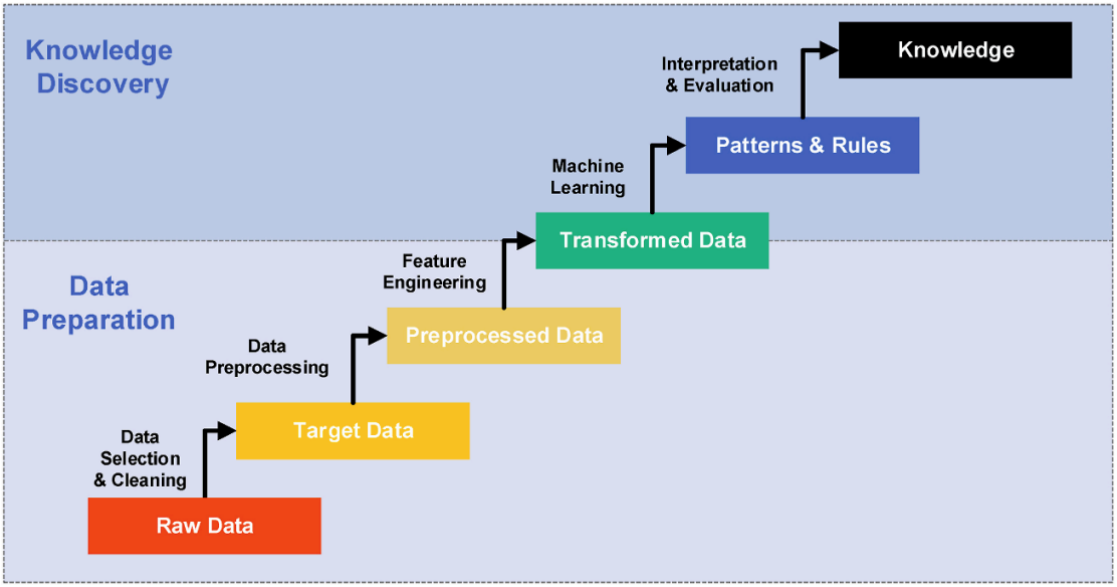


Figure 2.6: Integration of domain knowledge throughout the Machine Learning workflow [23]

As discussed by [23], domain expertise plays a decisive role throughout the feature engineering process, guiding the selection of the initial variables, the construction of application-specific

features and the choice of the most relevant predictors. Consequently, effective feature engineering is not only a data-driven process but also one that incorporates knowledge of the underlying energy system.

Examples of this integration include the incorporation of operational constraints, such as maximum generation capacity or ramp-rate limits, to ensure physically feasible predictions [24], as well as anomaly detection through data cleaning and normalization to distinguish meaningful patterns from sensor faults [22]. Domain knowledge can also guide the construction of specialized features, such as frequency-domain representations extracted through Fast Fourier Transforms (FFT) to capture daily and seasonal periodicities in meteorological data, or the selection of independent variables, including atmospheric variables such as solar radiation, cloud cover and humidity, to improve forecasting performance in data-constrained scenarios [25].

2.4 AutoML for Time Series Forecasting

2.4.1 The AutoML Pipeline

Automated Machine Learning (AutoML) refers to a class of methods and tools designed to automate key components of the machine learning workflow, including model selection, hyperparameter optimization, and, in some implementations, data preprocessing.

The AutoML pipeline consists in more than one processing step, from raw data preparation to final model evaluation and deployment [26]:

- **Preprocessing**: prepares and cleans raw data, handling missing values, normalization, outlier detection, and data formatting to ensure consistency for downstream learning tasks.
- **Feature Engineering**: constructs, transforms, and selects informative features from the preprocessed data, aiming to capture relevant patterns while reducing noise and redundancy.
- **Model Selection**: evaluates multiple candidate algorithms or model families to identify those best suited to the characteristics of the task (e.g., classification, regression, or forecasting).
- **Hyperparameter Optimization**: systematically tunes model hyperparameters using automated search strategies such as Bayesian optimization or evolutionary algorithms. Notably, [27] propose an LLM agent-based approach (see subsection 2.6.2), *AgentHPO*, focused exclusively on automating and improving this stage.
- **Final Model Evaluation**: assesses the best model obtained after all preceding steps, using appropriate performance metrics and validation strategies to estimate its generalization capability before deployment.

In work developed by [28], a scheme was developed to connect these processing steps (see Figure 2.7).

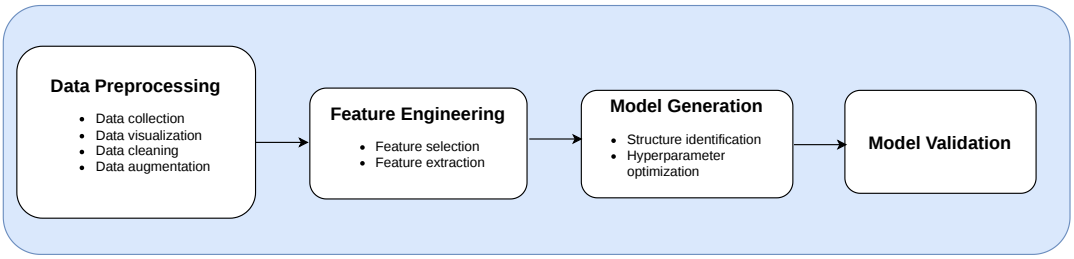


Figure 2.7: Typical pipeline of an AutoML process, adapted from [28]

It is this full-pipeline formulation that determines how large the search space actually is, and consequently how expensive the AutoML process is to run. In subsection 2.4.2, several AutoML frameworks are evaluated, considering all the steps performed by their pipelines.

2.4.2 AutoML frameworks applied to the energy domain

In the energy domain, AutoML has been explored for short-term load forecasting, electricity price forecasting, and building energy consumption prediction [29–31], with frameworks such as Auto-sklearn, Auto-WEKA, TPOT, AutoGluon, FLAML, AutoKeras and H2O applied across these tasks. In Figure 2.8, it is possible to see how some of these frameworks integrate in the pipeline used for predictions of energy load.

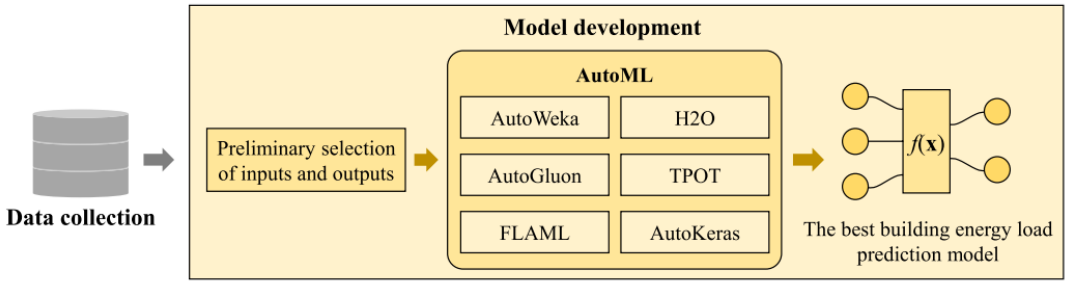


Figure 2.8: Flowchart of the AutoML-based building energy load prediction method [30]

In more detail, the research conducted in [31] described some of the most relevant classic AutoML systems (Auto-sklearn, Auto-WEKA and TPOT):

- **Auto-sklearn** is a Bayesian optimization–based AutoML framework built on top of scikit-learn that automatically constructs machine learning pipelines by jointly optimizing preprocessing steps, model selection, and hyperparameters. It relies on the SMAC algorithm with random-forest surrogate models and further enhances performance through an automated ensemble construction stage that combines multiple high-performing models discovered during optimization.
- **Auto-WEKA** is an AutoML system that also employs SMAC for Bayesian optimization but operates within the WEKA ecosystem. Unlike auto-sklearn, Auto-WEKA focuses on

selecting a single optimal pipeline without ensemble construction or meta-learning-based initialization, resulting in a simpler but less adaptive optimization process.

- **TPOT** (Tree-based Pipeline Optimization Tool) is an evolutionary AutoML framework that represents machine learning pipelines as tree structures and optimizes them using genetic programming. Instead of Bayesian optimization, TPOT evolves pipeline components, including preprocessing, feature selection, decomposition, and model choices, by applying genetic operators to maximize predictive performance, making it particularly flexible in discovering novel pipeline configurations.

For a deeper comparison in detail, workflows of both Auto-sklearn and TPOT were also shown by [31], in Figures 2.9 and 2.10.

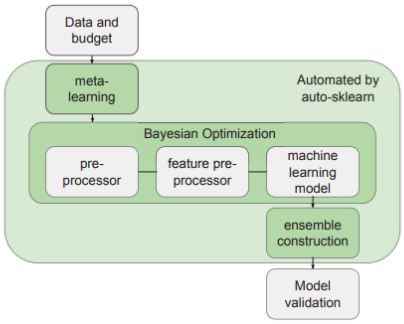


Figure 2.9: Auto-sklearn workflow [31]

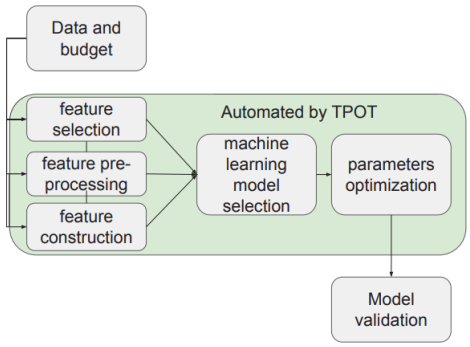


Figure 2.10: TPOT workflow [31]

Zhang et al. [30] compared six representative AutoML frameworks directly (Table 2.1), evaluating them on the task of forecasting one-hour-ahead heating, cooling, and electrical loads for three real-world buildings.

Table 2.1: Comparison of basic functions among six representative AutoML frameworks, adapted from [30]

AutoML framework	**Data preprocessing**	**Feature engineering**	**Model selection**	**Hyperparameter optimization**	**Model evaluation**
AutoWeka	No	Yes	Yes	Yes	Yes
H2O	No	No	Yes	Yes	Yes
TPOT	Yes	Yes	Yes	Yes	Yes
AutoGluon	No	No	Yes	No	Yes
FLAML	No	No	Yes	Yes	Yes
AutoKeras	No	No	No	Yes	Yes

As Table 2.1 shows, TPOT is the only framework among the six that covers every stage of the pipeline, including data preprocessing and feature engineering; the remaining frameworks each omit at least one stage, most commonly preprocessing and feature engineering, and instead restrict

their search to model selection and hyperparameter optimization. This important distinction in the scope of search is the basis for selecting TPOT in this dissertation work.

Regarding forecasting accuracy, the research conducted in [30] also compared some AutoML frameworks' performances against manually configured models. Most AutoML frameworks matched or exceeded manual modeling, with accuracy gains over manual modeling ranging from 1.10% to 18.66% across MAE, RMSE, CV-RMSE and R^2, depending on the framework and considered scenario, but the ranking was not consistent: AutoKeras consistently underperformed manual modeling across all the considered scenarios, and AutoWeka's relative performance varied between buildings. TPOT outperformed manual modeling in two of the three scenarios but was never the single best-performing framework in any of them.

Furthermore, the authors of [31] compared the performance of Auto-sklearn and TPOT on energy consumption prediction. TPOT generally achieved lower testing error than Auto-sklearn across all training time budgets tested (Figure 2.11). However, this advantage was conditional on the validation protocol: under 10-fold cross-validation, TPOT's results became considerably less stable than Auto-sklearn's, with both higher mean error and higher variance. Moreover, on a second, harder real-world benchmark in the same study, neither Auto-sklearn nor TPOT was able to outperform a baseline developed by a human expert over a six-month project, although both came close within just a few hours of automated search.

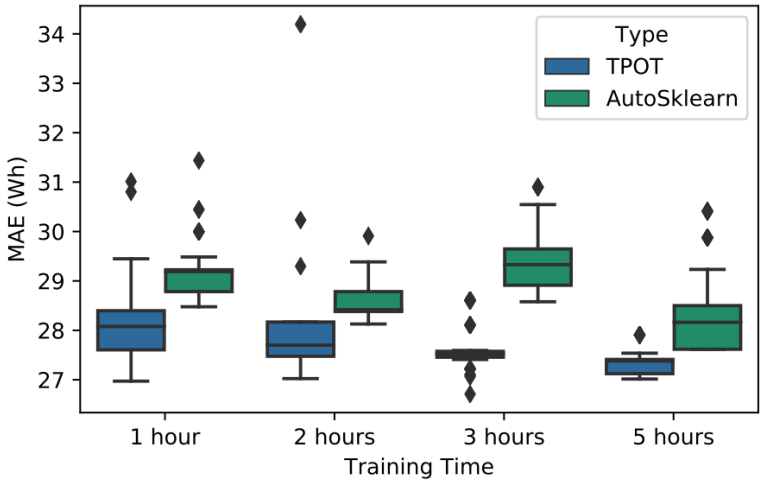


Figure 2.11: Testing error on energy use appliance consumption data with different training time [31]

Overall, despite its performance limitations in some scenarios reported in the literature, TPOT demonstrated promising results and remained competitive with other AutoML frameworks, while offering the core functionalities required for the AutoML process. These characteristics make it a suitable framework for the objectives of this dissertation.

2.4.3 Challenges when applying AutoML for Time Series

Unlike static datasets, time series data exhibit temporal dependencies, non-stationarity, and evolving patterns, which represent a challenge for applying suitable AutoML pipelines.

One of the most critical challenges is ensuring data integrity and preventing data leakage. As highlighted by [32], data leakage in time series forecasting often arises from the inadvertent use of future information during training, unrealistic assumptions about real-time data availability, or overlapping training and validation windows. Such issues can lead to overly optimistic performance estimates and poor generalization. To mitigate these risks, AutoML systems must incorporate time-aware cross-validation strategies and enforce strict temporal constraints during feature engineering and model evaluation.

As discussed in section 2.1, time series forecasting is further complicated by the inherent volatility of many real-world processes, particularly in energy-related applications. These sources of non-stationarity introduce significant uncertainty and irregular patterns into the data, which can substantially degrade forecasting accuracy [33]. For AutoML frameworks, capturing such complex temporal dynamics in an automated manner remains a significant challenge.

Another commonly encountered issue is feature mismatch, which occurs when selected features fail to align with the temporal resolution, lag structure, or dynamic behavior of the target variable. As discussed in [32], inappropriate feature selection can lead to misleading correlations and unreliable forecasts. This challenge is particularly pronounced in AutoML settings, where feature engineering and selection are often performed automatically without explicit domain-level validation.

Besides the challenges discussed above, the computational cost of the AutoML model search represents an issue for some developed approaches, being one of the topics of improvement in this dissertation. While traditional machine learning already requires extensive experimentation to identify suitable models and hyperparameter configurations [34], full-pipeline AutoML further expands this search by optimizing preprocessing operations, feature engineering, model selection, and hyperparameters. As a result, AutoML frameworks such as TPOT often require several hours of computation for a single dataset, with reported search times ranging from approximately 13 to 19 hours depending on the application and search space, for large datasets [35]. Since the AutoML search is generally performed independently for each forecasting task, the computational cost scales with the number of time series, making exhaustive pipeline optimization increasingly impractical in real-world deployments.

This thesis addresses precisely this limitation by reducing the AutoML search space before optimization begins. Rather than allowing TPOT to exhaustively explore all candidate configurations, prior knowledge extracted from a Large Language Model (LLM) is used to guide the search toward promising model families and hyperparameter configurations. By constraining the search space, the proposed approach aims to reduce computational time while preserving the predictive performance of the resulting forecasting pipelines. The following sections describe how these LLM-derived priors are obtained and incorporated into the AutoML process.

2.5 Extraction and Operationalization of Scientific Knowledge using LLMs

2.5.1 Large Language Models

Large Language Models (LLMs) are AI systems trained on large amounts of text, allowing them to understand, generate, summarize, translate, and predict natural language, producing human-like responses. Recent research shows their potential for finding, extracting, and combining information from unstructured sources, including identifying key entities and relationships and turning them into structured formats that support further analysis [36]. This makes LLMs especially useful for speeding up literature reviews, discovering new knowledge, and building features from data, including turning tabular and textual scientific data into knowledge representations for different systems [37].

Within the energy sector, [38] provides a systematic review of how LLMs are being applied to improve decision support, situational awareness, and operational efficiency in smart energy systems. The review finds that LLMs can process both structured and unstructured data (time-series measurements, weather information, operational logs, technical documentation, and human-generated reports) to enable more contextualized reasoning over complex energy management tasks. Reported applications cover three main areas:

- **Forecasting and monitoring:** electricity demand, wind and photovoltaic generation forecasting, industrial carbon emissions prediction, and electrical safety incident detection through analysis of operational data and reports;
- **Decision support and optimization:** smart grid balancing and operational planning, energy efficiency optimization across heterogeneous data sources, and Building Energy Modelling (BEM), including model generation, debugging, retrofit recommendations, and urban-scale energy analysis;
- **Domain-specific assistants:** LLMs fine-tuned on energy-related documentation, smart meter data, maintenance logs, and technical reports to directly support engineers and grid operators.

Overall, [38] concludes that LLMs extend the capabilities of traditional machine learning by integrating heterogeneous information sources and reasoning together over numerical and textual data, making them particularly well suited to next-generation smart grids and intelligent energy management systems.

Despite these advantages, LLMs carry limitations that are especially consequential in scientific and operational contexts where factual accuracy is critical. The survey by [39] identifies hallucination (the generation of fluent, plausible-sounding but factually incorrect output) as a primary concern, alongside contextual inconsistency and sensitivity to decoding strategies. Table 2.2 summarizes the main causes, detection methods, and mitigation strategies associated with this issue.

Table 2.2: Summary of the main hallucination causes, detection methods, and mitigation strategies, adapted from [39].

Category	Main aspects
Hallucination Causes	• **Data-related:** biased, incomplete or low-quality training data. • **Training-related:** limitations during pre-training, supervised fine-tuning (SFT), or reinforcement learning from human feedback (RLHF). • **Inference-related:** decoding strategies, model overconfidence, and reasoning failures.
Hallucination Detection	• **Factuality verification:** checking whether generated information is supported by reliable evidence. • **Faithfulness verification:** assessing whether the output is consistent with the provided input or retrieved context. • **Benchmark-based evaluation:** standardized datasets and benchmarks for measuring hallucination rates.
Hallucination Mitigation	• **Data-level:** improved data filtering and curation. • **Model-level:** model editing, fine-tuning, and Retrieval-Augmented Generation (RAG). • **Inference-level:** factuality-aware and faithfulness-aware decoding strategies.

2.5.2 Information Extraction

Large Language Models have demonstrated significant potential for automated information extraction from unstructured scientific and technical texts, enabling the transformation of raw research content into structured knowledge that supports downstream modeling workflows.

In the energy sector specifically, [40] developed an LLM-based framework to extract hydropower regulatory information from research documents, evaluating the performance of eight LLMs with different capabilities. The pipeline proposed by the authors covers several stages, from raw information extraction to the construction of the final knowledge base:

- **Document Collection:** Hydropower licensing and regulatory documents are collected as the input data source. These documents contain unstructured textual information describing project characteristics, operational constraints, and environmental requirements.
- **Text Splitting:** Documents are parsed and divided into smaller text segments that fit within the LLM's context window.
- **LLM-based Information Extraction:** Each text segment is provided to an LLM together with a predefined extraction schema, which the model uses to identify and extract the relevant fields.

- **Output Validation:** An optional step in which the extracted values are reviewed, with the LLM verifying whether each extracted value is explicitly supported by the source text.
- **Performance Evaluation:** Relevant metrics (precision, recall, and F1-score) are computed to assess the quality of the extracted information.

A broader survey by Xu et al. highlights a shift towards generative information extraction, in which a wide range of extraction subtasks such as entity and relation extraction are reformulated as a single text-generation problem, allowing one prompted LLM to replace task-specific architectures that were previously required [41].

The overall pipeline (Figure 2.12) illustrates distinct data preparation and data extraction phases, highlighting how domain knowledge from research content can be operationalized into structured inputs. This approach exemplifies how LLM-supported extraction can accelerate knowledge discovery and automate aspects of literature review that were previously manual, whilst making the connection with further model evaluation & interpretation with AutoML.

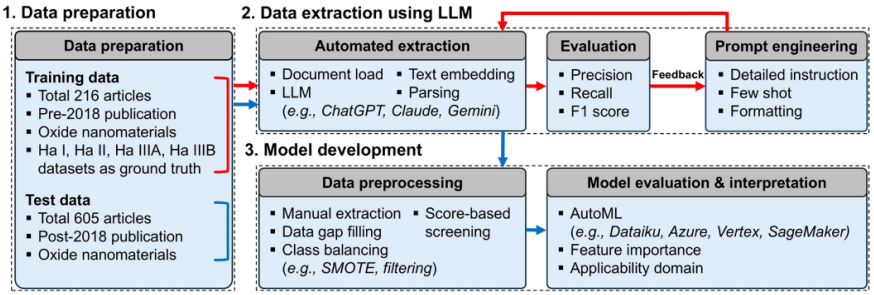


Figure 2.12: Overall workflow of the LLM extraction, including the Data preparation and Data extraction phases [42]

2.5.3 Code generation

As stated in the work developed in [43], Large Language Models have also demonstrated a significant capability in code generation, enabling the automatic translation of natural language descriptions into executable programming constructs. Beyond their use in text-based reasoning tasks, LLMs trained on large-scale code repositories can generate syntactically correct and semantically meaningful code, supporting tasks such as data preprocessing, feature extraction, and model implementation.

Building on this general code generation capability, an emerging line of work applies agentic coding tools directly to machine learning and data science workflows, where the LLM is used to plan, write, execute, and iteratively refine code within the modeling pipeline itself, rather than producing a single static output from a prompt [44]. CAAFE [45], for example, uses an LLM to iteratively propose and implement new features for tabular datasets, generating executable feature-construction code that is evaluated against downstream model performance and refined based on that feedback, effectively closing the loop between contextual domain knowledge and executable

feature engineering. More broadly, tools such as Data Interpreter and DS-Agent extend this idea to multi-stage data science pipelines, combining planning, retrieval, and code execution to automate tasks including data preprocessing, feature engineering, model selection, and hyperparameter optimization with limited human intervention [46, 47].

An inherent question that follows from this line of work is how reliable LLM-driven code generation actually is, and whether that reliability depends on the extent of the task being delegated to the model. To answer this, Table 2.3 summarizes recently reported results across two categories of benchmarks: those that evaluate LLMs on narrow, single-objective code-generation tasks, such as fixing a bug in an existing repository (SWE-bench) or generating new code against a repository's API (ML-Bench), and those that evaluate LLMs on open-ended, autonomous construction of a full ML pipeline from scratch (MLE-bench, MLE-STAR).

Table 2.3: Comparison of ML and Software Engineering Benchmarks across AI Models and Scaffolds

Benchmark	**Task**	**Model / Scaffold**	**Score**	**Source**
MLE-bench	Full ML pipeline	o1-preview + AIDE	16.9%[*1]	[48]
MLE-bench	Full ML pipeline	o1-preview + AIDE (8 attempts per task)	34.1%[*2]	[48]
MLE-STAR	Full ML pipeline	AIDE + Gemini-2.0-Flash (baseline)	25.8%[*1]	[49]
MLE-STAR	Full ML pipeline	MLE-STAR + Gemini-2.0-Flash (retrieval + refinement)	43.9%[*3]	[49]
MLE-STAR	Full ML pipeline	MLE-STAR + Gemini-2.5-Pro	>60%[*3]	[49]
ML-Bench	Repository-level code generation	GPT-4o (within 5 attempts)	>50%[*4]	[50]
SWE-bench	Repository-level bug fixing	Claude Sonnet 4	70.4%[*5]	[51]
SWE-bench	Repository-level bug fixing	o3	69.1%[*5]	[51]
SWE-bench Verified	Repository-level bug fixing	Claude Opus 4	67.8%[*5]	[51]
SWE-bench	Repository-level bug fixing	GPT-4.1	48.6%[*5]	[51]

[*1] % of Kaggle competitions where the agent's autonomously trained model matched or exceeded the human bronze-medal threshold on the competition leaderboard.
[*2] Same criterion as [*1], using the best result out of 8 independent attempts per competition.
[*3] Same criterion as [*1], achieved using MLE-STAR's retrieval-augmented, targeted refinement loop instead of the standard AIDE scaffold.
[*4] % of repository-level tasks where at least one of 5 generated code attempts correctly used the target repository's API to fulfill the given instruction, verified by execution.
[*5] % of real, human-verified GitHub issues for which the generated patch made the associated test suite pass.
Note: these percentages are not directly comparable across benchmarks, as each one reflects a different success criterion, task type, and attempt budget.

As Table 2.3 shows, narrow, single-objective code-generation tasks achieve substantially higher

success rates (50–70%) than open-ended, autonomous full-pipeline construction, which drops below 45% even for the strongest reported configurations. This large gap exists, since the full-pipeline benchmarks evaluate if the whole resulting model is competitively accurate, rather than only evaluating if the generated code itself is correct.

This distinction directly motivates the methodological choice adopted in this dissertation. Rather than delegating the entire ML pipeline to an LLM-driven agent, as is done in MLE-bench or MLE-STAR-style systems, the pipeline proposed in this work restricts LLM usage to the simpler, better-supported task of generating code from knowledge extracted from scientific literature.

2.5.4 Feature Engineering

Large Language Models (LLMs) can also be leveraged to improve the process of feature engineering. By analyzing the content of research papers, LLMs can identify relevant variables, relationships, and patterns, effectively generating insights that inform the creation of new features. These features can then be applied and tested on a target dataset, enabling more informed and data-driven modeling.

The work presented in [42] leverages the LangChain [52] framework to implement retrieval-augmented generation (RAG), enabling the direct extraction of relevant information from research papers stored in an external knowledge base. To strengthen the outputs of the large language model (LLM), the authors employed a response refinement strategy, in which the model's outputs were compared against actual dataset values, and feedback was used to iteratively improve the responses. This approach is particularly relevant to feature engineering, as it not only accelerates the process but also improves the quality of features by refining prompts based on feedback from empirical data.

While such approaches use the LLM to actively propose and refine new features through an iterative generation loop, this dissertation work only uses the LLM to identify and extract features that have already been engineered and reported by the authors of the considered research literature, rather than to generate new ones.

2.6 Hybrid LLM and AutoML Systems

2.6.1 Transforming Extracted Knowledge into AutoML Priors

One of the main advantages of integrating LLMs into AutoML systems is their ability to transform unstructured knowledge into actionable priors that guide downstream optimization tasks. The authors of [53] developed a framework named *AutoSurrogate* for physics-governed simulation tasks, where a dedicated model-selection agent leverages domain knowledge to pre-filter the space of candidate architectures before hyperparameter optimization is performed. Instead of treating architectures and hyperparameters as a single, undifferentiated combinatorial search space, the framework first eliminates architectures that are unlikely to satisfy the problem's physical constraints. According to the authors, this domain-informed pre-filtering not only improves the

quality of the final surrogate models but also substantially reduces the computational cost of the AutoML process by concentrating the search on architectures that are more likely to be suitable for the task.

Figure 2.13 illustrates the transition from a conventional machine learning workflow to the proposed LLM-driven multi-agent pipeline. In this architecture, multiple LLM-based agents are responsible for interpreting user requirements, extracting domain knowledge, selecting candidate model families, and orchestrating subsequent AutoML stages. Rather than replacing the optimization process, the agents inject structured prior knowledge that constrains and guides the search, allowing AutoML to operate more efficiently.

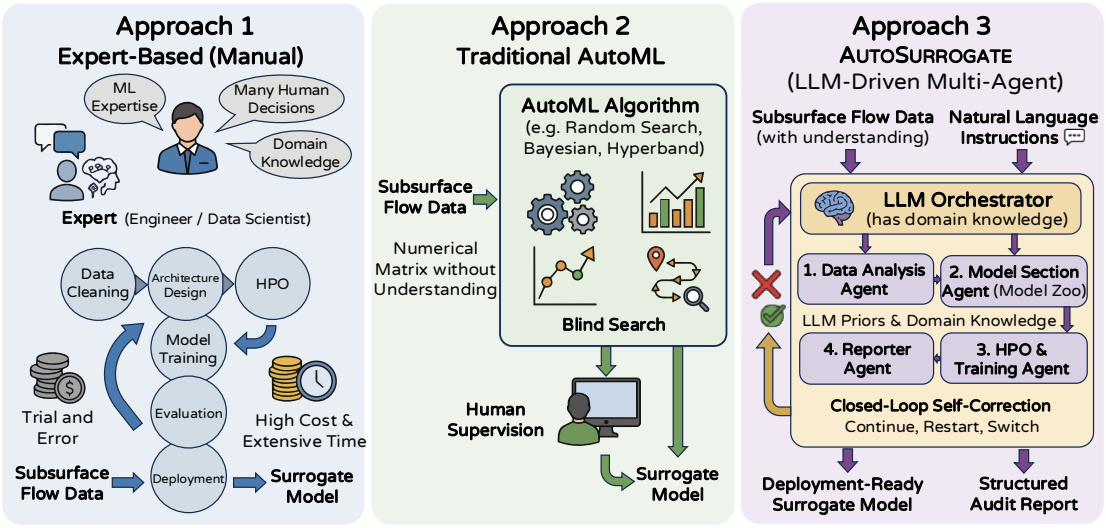


Figure 2.13: Comparison of different approaches for constructing a deep learning model for subsurface flow [53].

The results reported by the authors demonstrate that incorporating LLM-derived prior knowledge can significantly improve the efficiency of the AutoML pipeline by reducing the search space while maintaining or improving predictive performance. This observation provides a strong motivation for the approach adopted in this dissertation.

2.6.2 LLM-based Agents in AutoML Systems

An *LLM-based Agent* refers to a system in which one or more LLM instances are embedded as active components within a broader pipeline, interpreting user requests, coordinating pipeline stages, or constraining the search space, rather than an LLM that is trained or fine-tuned to directly produce predictions in place of the underlying process it supports. In other words, the LLM does not replace the system it is embedded in; it operates alongside or on top of it, guiding decisions rather than performing them directly. Applied to AutoML, this means the LLM does not replace

the AutoML system itself, but instead guides decisions such as data characterization, model selection, and hyperparameter configuration, while traditional model training and evaluation remain the responsibility of the underlying AutoML components.

A representative example of this approach is presented in [54], where the authors propose a multi-agent AutoML architecture composed of the following LLM-based agents:

- **Agent Manager**: Coordinates and distributes tasks among the other agents, ensuring that specialized components operate coherently within the overall pipeline.
- **Prompt Agent**: Parses user inputs into standardized JSON representations that are shared across agents, ensuring consistency during planning, search, and verification phases.
- **Data Agent**: Performs data-related tasks, including preprocessing, manipulation, and analysis. Its outputs characterize the dataset and inform downstream model selection and hyperparameter tuning.
- **Model Agent**: Handles model search, hyperparameter optimization (HPO), profiling, and candidate ranking, forwarding its recommendations to the Agent Manager for validation.
- **Optimization Agent**: Implements the validated solution by generating executable code, performing runtime execution, and recording experimental results prior to returning the final model to the user.

Each agent is guided by a dedicated system prompt that conditions the LLM to act as a specialist within its assigned role. For instance, the Model Agent is conditioned through a carefully designed prompt to behave as an expert in model selection and specification. This prompt guides the LLM to reason about suitable model families, hyperparameter configurations, and training strategies based on dataset properties and learning objectives. An example of such a system prompt is shown in Figure 2.14. Importantly, within this architecture, the LLM does not directly train models; instead, it provides high-level guidance that informs and constrains downstream AutoML processes.

System Message for Model Agent Specification

```
You are the world's best machine learning research engineer of an automated machine learning project (AutoML)
    that can find the optimal candidate machine learning models and artificial intelligence algorithms for the
    given dataset(s), run hyperparameter tuning to opimize the models, and perform metadata extraction and
    profiling to comprehensively understand the candidate models or algorithms based on the user requirements.
    You have the following main responsibilities to complete.
1. Retrieve a list of well-performing candidate ML models and AI algorithms for the given dataset based on the
    user's requirement and instruction.
2. Perform hyperparameter optimization for those candidate models or algorithms.
3. Extract useful information and underlying characteristics of the candidate models or algorithms using
    metadata extraction and profiling techniques.
4. Select the top-k ('k' will be given) well-performing models or algorithms based on the hyperparameter
    optimization and profiling results.
```

Figure 2.14: Example of a system prompt used to condition the LLM-based Model Agent for model selection and specification [54].

Figure 2.15 presents a general overview of the proposed *AutoML-Agent* framework and the interactions between its different agents.

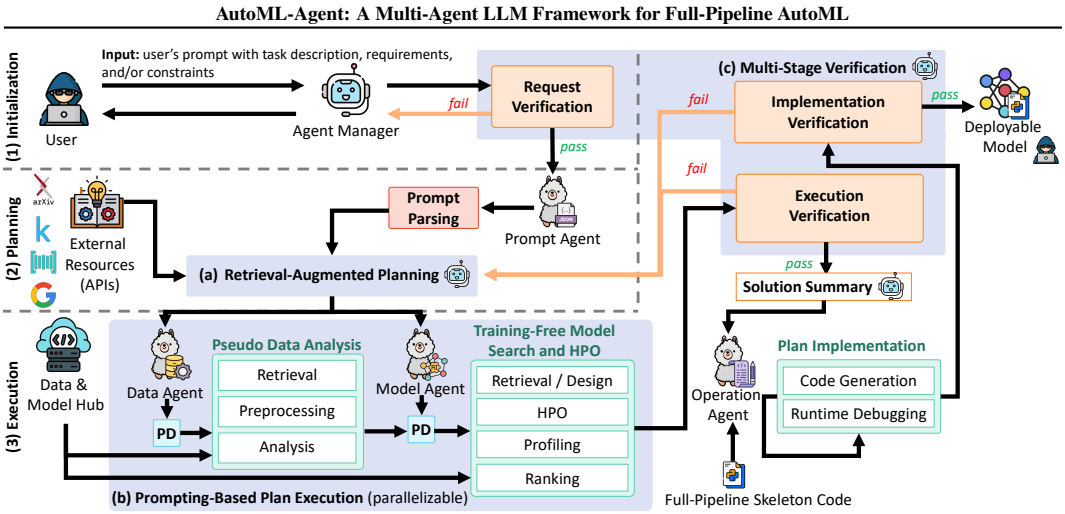


Figure 2.15: General overview of the *AutoML-Agent* framework, including all the steps from the RAG phase to extract information from external sources until the final validation with LLM [54].

Beyond this multi-agent design, several alternative architectures have been proposed in the literature, differing mainly in the number of agents used, the roles assigned to them, and the target domain. Table 2.4 summarizes these systems.

Table 2.4: Comparison of LLM-based AutoML agent architectures

Framework	# Agents	Agent Roles	Time Series	Domain
AutoML-Agent [54]	5	Agent Manager, Prompt, Data, Model, Optimization	No	General-purpose
AutoML-GPT [55]	2	Reasoning Agent, Coding Agent	No	General-purpose
TS-Agent [56]	2	Refinement Agent, Verification Agent	Yes	Time series
AgentHPO [27]	2	Creator, Executor	No	General-purpose (HPO-focused)
AutoM3L [57]	2	Modality Inference (MI-LLM), Feature Extraction/Refinement (AFE-LLM)	No	Multimodal

Across these systems, a common pattern emerges: as the number of agents decreases, roles tend to consolidate around a planning/reasoning function and an execution/coding function, while systems with more agents (e.g., AutoML-Agent) separate these responsibilities further into dataset characterization, model search, and optimization stages. This consistent separation between LLM-driven orchestration and traditional model execution motivates the design of the AutoML-agent-based pipeline developed in this dissertation, described in the following section.

2.6.3 Limitations of current approaches

Despite the growing adoption of LLM-based agents in AutoML systems, several challenges remain.

A recurring limitation across LLM-based agent systems more broadly is hallucination, where the model generates plausible but factually incorrect or unfounded outputs [58]. This risk is particularly relevant to AutoML agents, since erroneous outputs at any stage of the pipeline, such as an incorrectly characterized dataset property or a fabricated justification for a model choice, can silently propagate into downstream search decisions rather than being caught immediately. Closely related to this is the non-deterministic nature of LLM outputs: the same prompt can yield different responses across runs, undermining the reproducibility and stability expected from an AutoML pipeline [59].

Another important limitation concerns the dependence of many LLM-agent frameworks on carefully engineered prompts and predefined orchestration strategies. Although these mechanisms improve specialization and coordination between agents, they may also introduce additional complexity and sensitivity to prompt design choices, further compounding the reproducibility concerns raised by non-deterministic model behavior. In a broader review of LLM-based multi-agent systems, [60] similarly identify agent reliability, verification difficulty, and the lack of realistic, real-world evaluation benchmarks as key barriers preventing these systems from moving beyond prototype-level deployment.

Furthermore, the computational cost and latency associated with orchestrating multiple LLM-based agents may become significant in large-scale AutoML pipelines, particularly when iterative reasoning, code generation, and repeated validation steps are required [59, 61]. These constraints highlight the need for more efficient and scalable architectures capable of balancing reasoning capabilities with practical deployment requirements.

2.7 Critical analysis

The literature reviewed throughout this chapter demonstrates that significant progress has been achieved in the fields of energy forecasting, feature engineering, AutoML, and LLM-assisted automation. Modern forecasting models are increasingly capable of handling complex temporal dependencies, nonlinear relationships, and heterogeneous data sources, while AutoML frameworks reduce the amount of manual intervention required during model development. In parallel, recent advances in Large Language Models have introduced new possibilities for extracting and operationalizing domain knowledge from scientific literature, enabling more intelligent and context-aware machine learning pipelines.

In Table 2.5, it is shown an overview of performance by the different approaches considered across all domains covered by this dissertation work.

Table 2.5: Comparison Matrix of Documents and Sources Across Domains 1–8

Document (Source)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
STELLM [62]							X	X
Hydropower Extraction [40]	X		X					X
Building Energy AutoML [30]							X	X
CAAFE [45]		X					X	
ML-BENCH [50]		X	X			X	X	
DS-Agent [46]	X	X	X			X	X	
Data Interpreter [47]		X				X	X	
Text-to-ML [41]		X	X	X		X	X	
AutoML-Agent [54]	X	X	X	X	X	X	X	X
MLE-BENCH [48]							X	
MLE-STAR [49]		X	X			X	X	
AUTOSURROGATE [53]				X		X	X	X
Electricity Price Comparison [29]							X	X
Electric Load AutoML [31]							X	X
DARLI-AP-2 [21]		X					X	
Proposed approach	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

1 - Uses LLM for feature engineering techniques extraction from papers or other docs

2 - Uses LLM for feature engineering code generation

3 - Uses LLM for used regression models and parameters extraction from papers

4 - User specifies forecasting use case (forecast horizon, time resolution, model interpretability, etc.) in initial prompt

5 - LLM analyze which information the user has available (e.g. does the user have temperature forecasts?) and then uses this to filter which pipelines can be used

6 - LLM analyze extracted information and automate the forecasting pipeline creation (which features should I use/init AutoML with the best configs)

7 - Output validation with actual datasets and comparison with baseline approaches (AutoML without LLMs, Untuned ML models, etc.)

8 - Application to the energy domain

As it is possible to see from Table 2.5, only AutoML-Agent [54] satisfies all the evaluated domains across the registered examples of literature implementations. However, it does not integrate an established AutoML framework (e.g., TPOT, auto-sklearn); the only such framework mentioned, AutoGluon, is used solely as an external baseline in their experiments, not within AutoML-Agent's own pipeline. Instead, their Model Agent, an LLM, estimates the best models and hyperparameters from its own knowledge, a "training-free" search that bypasses actual training or empirical evaluation, with only the resulting single pipeline later implemented and run as code. The proposed approach in this dissertation work differs in this regard: the LLM is used only for information extraction of candidate features and models, while an actual fixed AutoML framework performs the empirical search and selection over this collected set. This lowers the impact of the LLM on the overall presented solution and reduces its associated reliability problems.

Chapter 3

Methodology

3.1 Overview

This chapter introduces the methodology used to find and extract domain knowledge from scientific documents and use this information to create priors to support time-series forecasting tasks, for the energy sector. The proposed workflow combines two main components (i) LLM-based agents, responsible for extracting and validating information from the literature (see Section 3.2) and (ii) a time-series forecasting framework, which leverages on the extracted knowledge and AutoML to build and evaluate forecasting pipelines (see Section 3.3). Figure 3.1 illustrates the complete methodology.

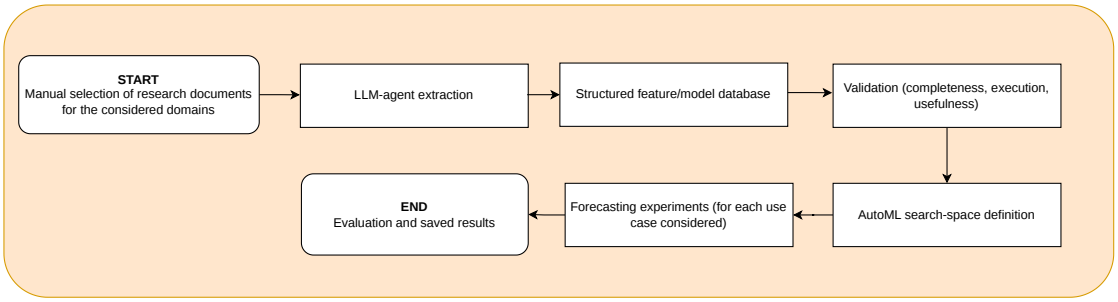


Figure 3.1: Flowchart of all performed processed steps for Methodology

As depicted by Figure 3.1, the process starts with a selection of scientific documents related to different forecasting use cases. In this work and given the complexity of this task by itself, we opted for a manual selection to ensure that the analyzed documents are relevant to the selected use cases. The selected documents are then processed by LLM-based agents that discover, extract and validate information about feature engineering techniques and forecasting models. This information is stored in a knowledge base (i.e., Excel workbook) and used to create priors to guide the AutoML process, which will use a refined search space to optimize the model search process, reducing its computational effort while maintaining (or improving) the forecast performance.

3.2 LLM-based knowledge extraction and validation

Developing accurate forecasting models for the energy sector often requires significant domain knowledge. This process goes beyond selecting a forecasting algorithm, which arguably is one of the most straightforward parts of the pipeline. In practice, data preprocessing and feature engineering techniques are key steps to ensure models are accurate given the specific characteristics of each use case, and remain robust over time. This knowledge already exists, but is typically dispersed across scientific papers, technical reports, presentations, industry publications and internal expertise. As a result, preparing a good forecasting pipeline can often be difficult and time-consuming, especially for newcomers or when addressing new use cases.

To automate knowledge discovery and preparation, an LLM-based workflow was developed. It processes technical and scientific documents, per forecasting use case (e.g., solar energy, electricity prices) and extracts structured information about commonly used feature engineering techniques, forecasting models, and implementation details (e.g., hyperparameters). A simplified diagram of this workflow is shown in Figure 3.2.

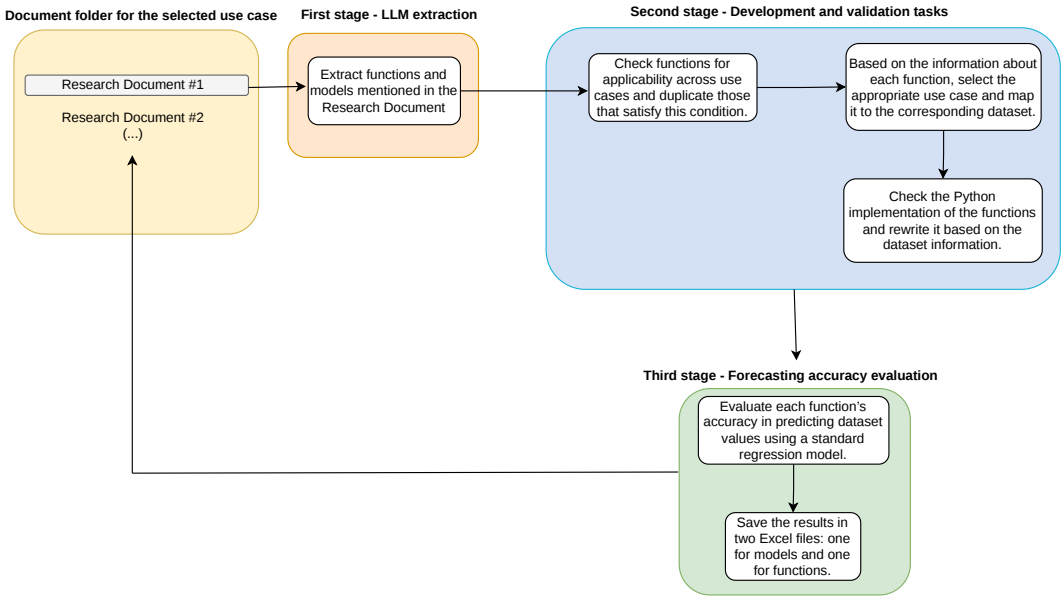


Figure 3.2: Extraction and validation workflow

The workflow is organized in three main stages.

- In the **first stage**, a dedicated LLM-based **information extraction agent** is responsible for retrieving and extracting relevant information from the source (in this case, scientific documents organized per forecasting use case). This includes feature engineering techniques, forecasting models, reported hyperparameters and input variables, per forecasting use case.
- In the **second stage**, the extracted content (represented in Figure 3.3 for functions and models) is passed to a **development and validation agent**. This agent generates Python code for

the candidate features, checks if the implementations can be executed, and verifies whether the candidate features are compatible with the raw variables available in the dataset that will be used later for training and testing forecasting models.

- Finally, on a **third stage**, the validated feature and model candidates are evaluated during forecasting experiments. This step ensures that candidate feature engineering functions (generated code) are not only executable, but also useful for forecasting performance.

Model output structure

```
{
"name": "<string>",
"horizon": "<string>",
"time_resolution": "<string>",
"description": "<string>",
"python_implementation":
"fn.get('python_implementation',
'')",
"hyperparameters": "<string>",
"sklearn_compatible":
  "<boolean>",
"case_of_use": "<string>",
"prediction\_variables":
  "<string>",
"complete": "<boolean>",
"search_space": "<string>",
"source": "{fn.get('source', '')}"
}
```

Function output structure

```
{
"name": "{fn['name']}",
"function_type":
  "{fn.get('function_type', '')}",
"short_description":
  "{fn.get('short_description', '')}",
"description":
  "{fn.get('description', '')}",
"python_implementation":
  "{fn.get('python_implementation', '')}",
"horizon": "{fn.get('horizon', '')}",
"case_of_use": "<one of wind, solar, load,
market price>",
"complete": "{fn.get('complete', True)}",
"compiles": "<boolean>",
"errors": "",
"improves_forecasting": "<boolean>",
"improves_intra\_day": "<boolean>",
"improves_day\_ahead": "<boolean>",
"source": "{fn.get('source', '')}",
"has_info_paper":
  "{fn.get('has_info_paper', False)}",
"has_info_dataset":
  "{fn.get('has_info_dataset', True)}",
"relies_on_lags":
  "{fn.get('relies_on_lags', False)}"
}
```

Figure 3.3: Structured JSON output formats exchanged between agents, in a form of a dictionary: model schema (left) and function schema (right).

To implement these ideas, a language model orchestration framework was used to connect LLMs with external documents for each use case. To separate extraction from validation, LangChain [52] and LangGraph [63] were selected as the most suitable frameworks, supporting both information extraction and subsequent validation of the extracted content. Figure 3.4 represents how these frameworks connect between agents.

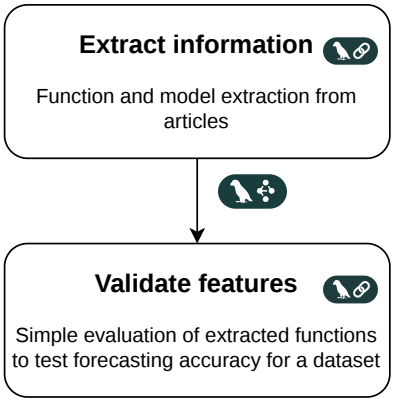


Figure 3.4: LLM-Agent workflow considered for the LLM tasks performed for extraction and validation

LangChain is commonly used to build flexible and modular LLM applications. It provides the foundational components for connecting user inputs (prompts, typically represented as human messages) to LLM-ready structured requests (model inputs), as well as for integrating external data sources and tools. LangGraph extends this capability by allowing the definition of agents that represent individual processing steps and conditional edges that control the flow between them, depending on intermediate results or decision criteria. This makes it particularly suitable for scenarios where the workflow is not strictly sequential, but may branch or loop based on validation outcomes. In this work, LangChain was used to connect the LLM to the required AI tasks, while LangGraph was used to orchestrate the interactions between agents, passing processed information (which was parsed to a JSON structured dictionary) from one agent to the next through an object provided by the LangGraph framework, better described in the LangGraph documentation [64].

The code that performed both the execution and consequent validation of all features is presented in Appendix A.

3.2.1 Information extraction agent

This subsection describes the first stage of the workflow, whose objective is to discover and extract, from scientific documents covering one or more forecasting use cases, information relevant to build forecasting pipelines for a list of use cases selected by an end-user. Specifically, information related to feature engineering techniques, forecasting models and respective hyperparameters.

For this purpose, an LLM-based agent was developed to process each scientific document, using two dedicated prompts: one to list and extract information for feature engineering techniques and one for regression models. Splitting the task in two reduces the complexity of each LLM call, allowing a clearer definition of expected constraints and desired output structures.

The full prompts are provided in the Appendix B.1 In this subsection, only the prompt objectives and constraints are summarized, to avoid duplicating the full prompt text.

3.2.1.1 Prompting strategy

Each scientific document is processed sequentially, by the two prompts. The first (see Appendix B.1) finds and extracts feature engineering functions such as calendar variables, lagged variables and rolling-window statistics, as well as raw variables that should be used *as is*. The second prompt (Appendix B.2) lists the regression models used, any reported hyper-parameters and whether each model can be represented through Python's `scikit-learn` API [65].

Both prompts require the model to return only valid JSON, making the output directly parsable and suitable for storage in the shared LangGraph state object, and allowing downstream agents to access individual fields (function name, implementation, use case, source, validation flags) without free-text interpretation. The two calls share conversation history via LangChain: when the model-extraction prompt is issued, it already has the context of the previously extracted features, so the reported model input variables can be aligned with those features.

3.2.1.2 Extraction constraints

Several constraints were included in the prompts to reduce invalid outputs and avoid unrealistic feature definitions:

- Each feature is classified into one of three categories. ***Raw functions*** correspond to direct input variables from the original dataset (e.g., wind speed or numerical weather predictions). ***Calendar functions*** correspond to timestamp-derived fields (e.g., hour, day of week, month). ***Feature functions*** correspond to engineered variables derived from one or more raw variables (e.g., lagged values, rolling means, sinusoidal encodings, or interaction terms).
- Each feature must be assigned to exactly one forecasting use case, except if the source document explicitly describes that it is used in multiple use cases.
- Any feature computed over observed variables must always be lagged by a minimum of N hours, which varies depending on the forecast horizon (e.g., if forecast horizon is 24 hours ahead, variables are first shifted in time by 24 hours), preventing access to information that would not be available at prediction time.
- If features are assigned to specific forecast horizons, these should also be explicitly declared. The model should select one of the following horizons: very-short-term ($\leq$ 24 h), short-term ($\geq$ 24 h), and medium-term (days to weeks).
- Models, metrics, and non-feature statistical descriptors should be excluded from the feature extraction step. For instance, correlation coefficients, standard deviation values used only as metrics, and complete forecasting models are not stored as feature functions.

3.2.1.3 Output format and parsing

Both prompts require the LLM to respond strictly with a predefined JSON structure. This particular format was selected to later facilitate the output parsing, storage and delivery to the following agents in our workflow. Tables 3.2 and 3.1 summarize the main attributes extracted by each prompt.

Because LLM outputs may contain formatting errors, each response is cleaned before parsing. Markdown fences, surrounding text, and minor markdown inconsistencies are removed before a JSON parser is applied. If parsing still fails, the document is skipped and a failure is logged for later review. If parsing succeeds, a final auxiliary function verifies that all required fields are present (see Tables 3.2 and 3.1), with expected types, before a new entry (i.e., feature engineering function or forecasting model) is added into the knowledge base.

Table 3.1: Function attributes extracted by the **information extraction agent**

Attribute	Description
Name	Name assigned to the function.
Function type	Category of the function: calendar function, raw function, or feature function.
Description	Detailed description of the function and its intended purpose.
Short description	Short description of the function.
Python implementation	Python code generated for the function.
Horizon	Forecasting horizon associated with the function: very-short-term, short-term, or medium-term.
Use case	Application domain associated with the function (e.g., wind, solar, market price forecasting).
Complete	(boolean) Indicates if the function has everything needed to compile.
Source	Research document from which the function was extracted.
Sufficient information	(boolean) Indicates whether the source document provides enough information to implement the function.
Uses lagged values	(boolean) Indicates whether the function uses lagged values of dataset variables.

Table 3.2: Model attributes extracted by the **information extraction agent**

Attribute	Description
Name	Name of the model, matching the class name used in the Python implementation.
Horizon	Forecasting horizon associated with the model: very-short-term, short-term, or medium-term.
Description	Description of the model and its intended forecasting application.
Python implementation	Python code generated for the model.
Hyperparameters	Hyperparameters reported in the source document.
Scikit-learn compatible	(boolean) Indicates whether the model is compatible with scikit-learn Python API.
Use case	Application domain associated with the model, such as wind, solar, or market price forecasting.
Input variables	Variables commonly used as inputs to the model.
Complete	(boolean) Indicates whether the model entry contains all information required for implementation.
Search space	Module path used to construct the TPOT search space, e.g., `sklearn.ensemble` for `GradientBoostingRegressor`.
Source	Research document from which the model was extracted.

3.2.2 Development and validation agent

After the information extraction phase (see Section 3.2.1), each candidate feature function is represented by a generic, dataset-agnostic, Python implementation, following the nomenclature (e.g., variable names) used by each source document. Because scientific documents often differ in variable naming and may reference variables that do not exist in local datasets, a validation agent adapts and verifies each implementation before it is used. Through a dedicated LLM call per feature engineering function, with the local dataset's actual column names injected in the prompt, the previously generic Python code is rewritten to use only existing variables. If the required columns to implement the function are not present in the dataset, the feature function is skipped.

Figure 3.5 shows the full sequence of steps, detailed below.

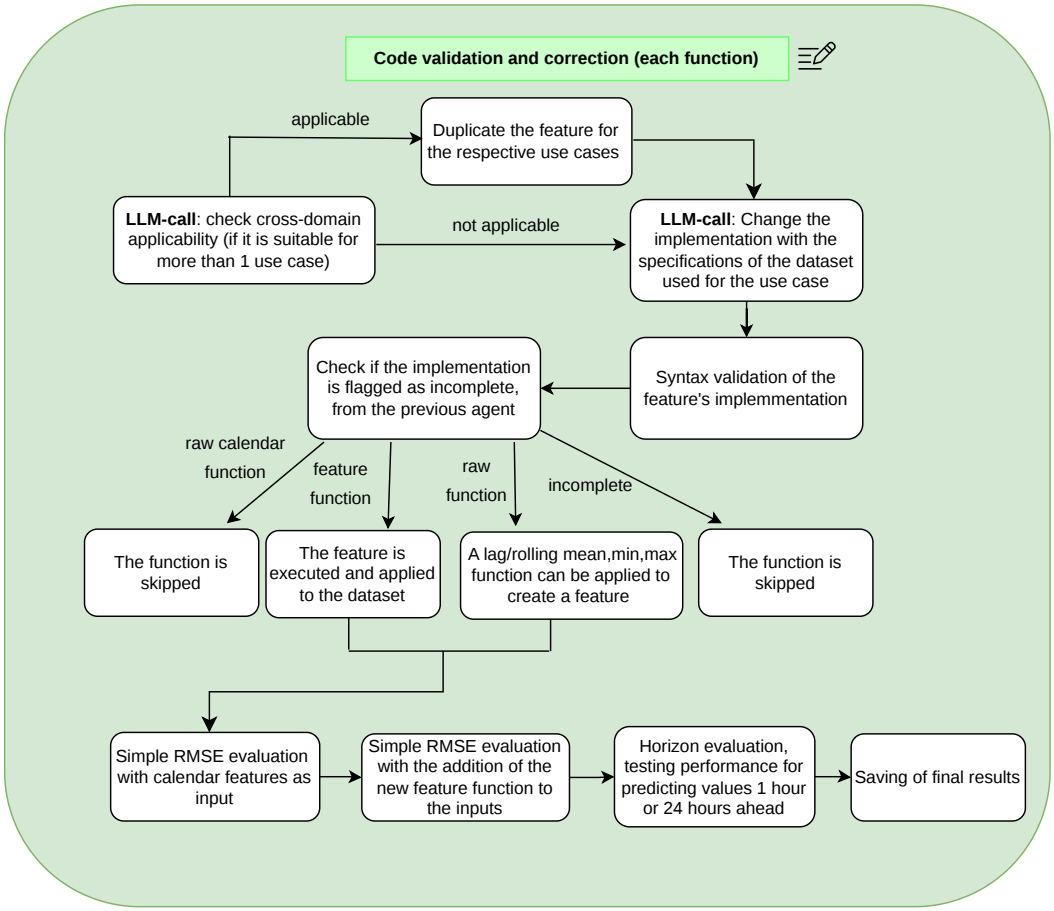


Figure 3.5: Steps performed by the validation agent to adapt, execute, and screen each candidate function

3.2.2.1 Syntax correction

The source code returned in the LLM responses is first passed through a local cleaning function that strips any markdown formatting introduced by the model, such as triple-backtick (“‘) fences or surrounding quotes. It then verifies if the expected function name is actually defined, rejecting any code with, for instance, missing `def` statements. Finally, it checks for syntax errors in the source code. Only if all these checks pass, the updated implementation is written back to the shared LangGraph object, otherwise a warning is issued and the original implementation is preserved.

3.2.2.2 Handling of raw features

A special handling path exists for functions classified as raw functions, meaning they represent direct input variables rather than engineered features. Since these functions do not inherently add a new derived column, an additional LLM call is made to identify which existing column in the dataset corresponds to the function. If the implementation is complete, the user is then asked, with

a short timeout of 5 seconds, to optionally apply a standard transformation to that column, choosing from a fixed set of options: 24-hour lag, 48-hour lag, 24-hour rolling mean, rolling standard deviation, rolling minimum, or rolling maximum. If no input is received within the timeout window, no transformation is applied. This timeout was applied to run the processing pipeline without interruptions, but it can be disabled at any time to give more time to the user to make the selection. If a transformation is selected, the derived column is created, the original duplicated raw column is dropped, and the function's data fields are updated accordingly, its 'function type' is changed to 'feature function', its 'name' is updated to reflect the transformation applied, and 'relies on lags' is set to 'True'.

In a similar path, functions classified as 'raw calendar function' (those that simply recompute standard calendar variables such as hour or day of week) are presented to the user ,with a timeout of 5 seconds for responding, asking whether they should be skipped, since these features are already present as baseline inputs in the evaluation framework.

3.2.2.3 Code execution

Once the code has been cleaned and validated syntactically, it is executed using Python's built-in function for executing code (`exec()`) within a local scope containing a copy of the target dataset. If the function raises any exception during execution, for instance due to missing columns, type errors, or incorrect return types, the 'compiles' field from the shared object is set to False, the error message is recorded in the 'errors' field, and the function is skipped for the remainder of the evaluation. If execution succeeds, the function object is called to be applied directly to the dataset. The return value is then inspected to ensure it is in the expected datatype . Functions that return tuples, booleans, or any other type of unexpected data are discarded at this point.

3.2.2.4 Accuracy test

To evaluate whether each extracted feature contributes positively to forecasting accuracy, a quick validation procedure is applied, using a HistGradientBoostingRegressor with a fixed random state for reproducibility. This is intended as a fast screening step, not the final evaluation, so a simple 80/20 train-test split is used rather than time series cross-validation, which is reserved for later, more rigorous evaluation.

- **Global accuracy check:** a baseline model using only calendar features (hour, day of week, month) is compared against a feature-augmented model that adds the new column. Both are evaluated on the same test split using RMSE; if the augmented model achieves lower error, the 'Improves_forecasting' flag is set to 'True' for that feature.
- **Horizon-specific check:** the feature is evaluated separately for intra-day (1-hour-ahead) and day-ahead (24-hour-ahead) forecasting. For each horizon, 20 forecast steps are run from randomly selected starting points, using only data strictly prior to each forecast time. Mean absolute error is compared between the baseline and augmented models; if the augmented

model performs better, the corresponding 'Improves intra-day' or 'Improves day-ahead' field is updated accordingly.

3.2.2.5 Final output

After processing all the previous validation steps, other fields from the shared object between agents are filled, as shown in Table 3.3

Table 3.3: Function attributes filled directly in the feature validation phase

Dataset information	True if it is possible to implement the function with the information available in the dataset; otherwise False
Compiles	True if the function execution is successful without errors
Errors	Errors obtained upon failing feature execution. If there are no errors in execution, this field is empty
Improves forecasting	True if the function improved forecasting as an input in the simple train-test split evaluation; otherwise False
Improves day ahead	True if the function improved forecasting as an input for a day-ahead prediction scenario; otherwise False
Improves intra-day	True if the function improved forecasting as an input for a intra-day prediction scenario; otherwise False

After being processed by both LLM-Agents, the generated implementation is ready to be executed. Figure 3.6 shows an example of a created code that applies lags for solar forecasting.

```python
import pandas as pd

def lag2(df):
    numerical_columns = ['power', 'cfh_85', 'cfl_85',
                         'cft_85', 'cfm_85',
                         'swflx_85', 'temp_85']

    lag_columns = [col for col in numerical_columns
                   if col not in ['power']]

    power_lag_column = ['power']

    df_lag2 = df.copy()

    df_lag2[['cfh_85_lag2', 'cfl_85_lag2',
             'cft_85_lag2', 'cfm_85_lag2',
             'swflx_85_lag2', 'temp_85_lag2']] = df_lag2[lag_columns].
                 shift(2)

    df_lag2[['power_lag24']] = df_lag2[power_lag_column].shift(24)

    return df_lag2
```

Figure 3.6: Example of a generated python implementation for a lagged feature

3.2.3 Integrity validation

To assess the integrity and consistency of the results obtained from the extraction pipeline, a qualitative evaluation approach was developed to analyze the outputs generated for each use case considered. To make this evaluation, an experiment was carried out by a separate LLM, acting as a “judge”, whose role was to assess each extracted item against its corresponding source paper.

Although the proposed integrity-verification approach is compatible with any LLM, a locally hosted model was selected to perform this evaluation. This choice was motivated by the substantial computational and resource requirements associated with the task, since each evaluation involves processing a large token context, combining the full source paper with the corresponding description and Python implementation for every extracted feature or model. Relying on a local model allowed this evaluation to be performed efficiently, while avoiding the costs associated with a large number of repeated calls to cloud-based APIs.

For each extracted feature and forecasting model, the "judge" model was provided with the source paper alongside the corresponding description and Python implementation, and was instructed to return a structured rating for each of these two components independently, according to a four-category scale (Perfect, Good, Partial, Inaccurate), further detailed in Subsection 4.7.2.

Among all the fields collected and stored for each function, only these two indicators, description accuracy and implementation accuracy, were considered for this integrity assessment experiment.

Figure 3.7 illustrates the complete workflow followed for this integrity assessment, from the initial configuration of the judge model to the aggregation and storage of the resulting accuracy scores.

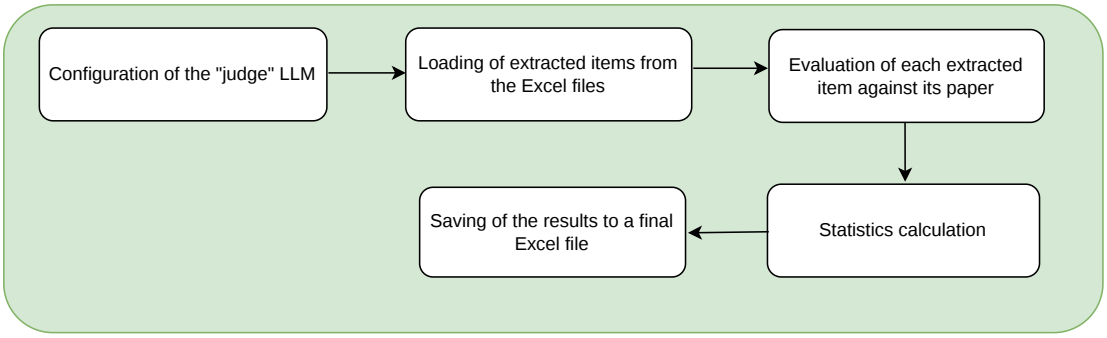


Figure 3.7: Workflow of the data integrity assessment

3.3 Time-series forecasting

This section describes the time-series forecasting framework used to evaluate the knowledge extracted by the LLM-based agents (see Section 3.2). The candidate feature functions and models are used to create priors to the AutoML process. The resulting pipelines are then evaluated against pre-defined forecasting benchmarks, ultimately to assess forecasting skill and computational efficiency of the model selection process.

3.3.1 AutoML Framework

The AutoML framework selected for this project was TPOT (ver 1.1.0) [66]. This is an open-source AutoML framework that uses genetic programming (GP) to automatically discover and optimize complete scikit-learn compatible machine learning pipelines. For better context, GP is an evolutionary algorithm, that has the fundamental principle to represent candidate solutions as individuals within a population and iteratively improve them through evolutionary mechanisms such as selection, crossover, and mutation. Figure 3.8 illustrates a TPOT pipeline representation.

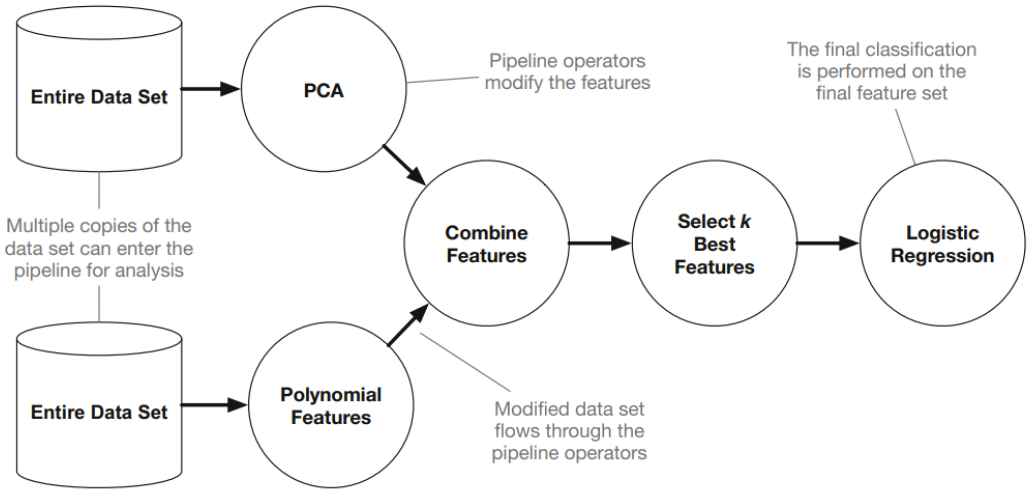


Figure 3.8: TPOT tree-based pipeline. Each circle corresponds to a machine learning operator, and the arrows indicate the direction of the data flow [67]

Among the available AutoML frameworks, TPOT was selected due to its straightforward integration and compatibility with scikit-learn API. TPOT focuses exclusively on scikit-learn-based models and pipelines, providing a simplified and efficient approach for automated model selection and optimization. Since the algorithms available within the scikit-learn framework were sufficient to evaluate the system's behavior, TPOT was considered an appropriate choice.

A limitation of TPOT (and other AutoML frameworks) is that, when used with a broad default search space, it explores a large number of candidate pipelines, not necessarily relevant to the forecasting problem under study, which consequently leads to long and time-consuming search processes for the best models for a specific forecasting use case.

3.3.2 From Knowledge Base to AutoML Priors

The knowledge base (defined in Section 4.3) is leveraged to guide the AutoML process, as introduced in the previous subsection. The implementation enables the selection of the regression models most frequently referenced in the scientific literature, which are then used to define TPOT's search space. Based on this knowledge-guided search space, TPOT performs an evolutionary search over candidate pipelines, jointly optimizing model selection, preprocessing operations and hyperparameters. Consequently, the search is constrained using evidence extracted from prior research, rather than relying on an unrestricted exploration of candidate algorithms.

3.3.3 Data Preparation and Feature Application

Regardless of the scenario selected (baseline or AutoML-driven, described later in subsection 3.3.4), a shared base dataset is constructed once from the original data. The datetime column is parsed and used to derive three standard calendar features: hour of day, day of week, and month. They are added to all feature sets as a fixed baseline.

The target variable and any columns that resulted from direct measurements are excluded from the considered input features. In a day-ahead forecasting setting, these quantities are not observable at prediction time (the forecast for day D+1 must be issued using only information available on day D) so they may enter the model only through lagged values, never at their concurrent timestamp. This distinguishes them from the remaining columns, which are themselves forecasts (predicted rather than measured values) and are therefore available upfront in a real deployment scenario. These are kept as raw input features and, together with the calendar columns, form the base columns used across all scenarios. Only functions marked as both 'complete' and 'compiles' in the stored knowledge base (see Section 4.3) are loaded for evaluation.

3.3.4 Evaluation

For the evaluation, an expanding-window time-series cross-validation approach was applied to test the model's predictive performance with the best hyperparameter configuration found by the AutoML framework. The full dataset was divided into 30 equal timesteps (folds). At each fold, the training set expands to include all preceding data, while the test set advances forward in time. Its primary purpose is to benchmark the different configurations considered on a common, dataset-wide evaluation scheme. For a more visual reference, Figure 3.9 shows how time-series cross-validation is applicable to a dataset.

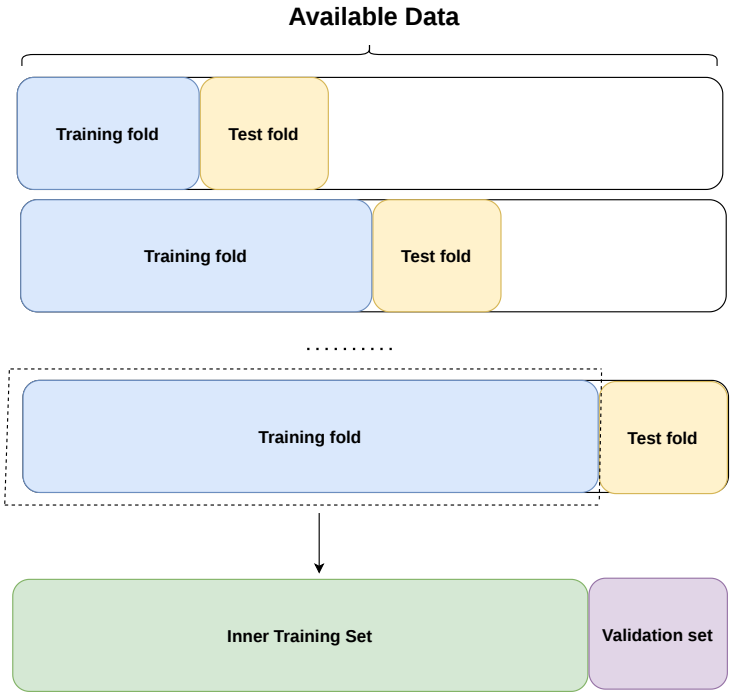


Figure 3.9: Time-series cross-validation logic used for the baselines considered

3.3.4.1 Forecasting baselines

Two baselines are defined for comparison against the proposed AutoML approach. The first applies a HistGradientBoostingRegressor [68] with default hyperparameters directly to the raw dataset columns, serving as a simple reference for what a standard model achieves without any feature engineering or tuning. The second follows the same setup but replaces the default configuration with hyperparameters optimized via Bayesian optimization using the Optuna library. Both baselines use the same regressor, whose suitability for this context is justified below. Figure 3.10 summarizes the proposed approach alongside these two baselines.

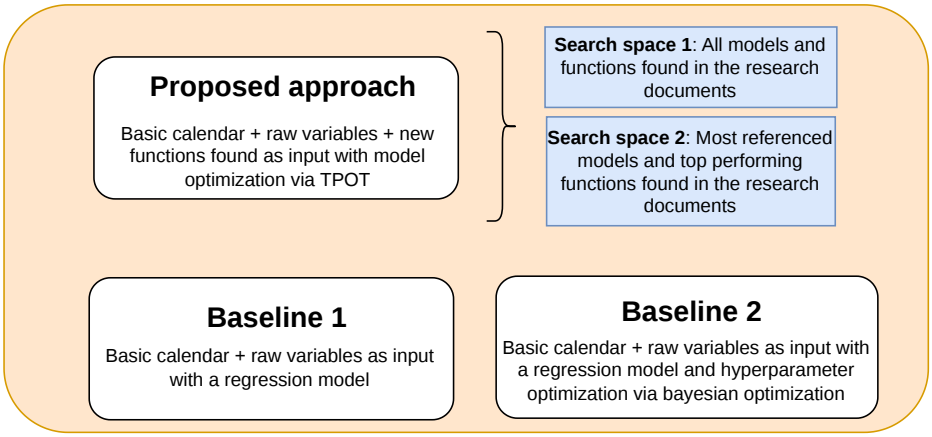


Figure 3.10: Proposed approach and available baselines for comparison

The HistGradientBoosting Regressor was selected for the baseline scenarios due to its simplicity and suitability for rapid prediction experiments. In contrast to the traditional Gradient Boosting Regressor, it can natively handle missing values (NaNs), making it more practical for most of the handled datasets.

According to the scikit-learn documentation [65], HistGradientBoosting builds an additive ensemble of decision trees by iteratively fitting new trees to the negative gradients (residuals) of the loss function. Prior to training, continuous features are discretized into a fixed number of integer-valued bins (histograms). This preprocessing step reduces the number of candidate split points evaluated during tree construction, resulting in significantly faster training while maintaining comparable predictive performance.

This process can be described step-by-step:

1. Start with an initial model, which can be the average of the target variable: $F_0(x) = \frac{1}{N}\sum_{i=1}^{N} y_i$
2. For each feature, compute histogram bins and then map every coninuous feature value to its corresponding bin index.
3. Iterate the model until reaching convergence, by computing the negative gradients for each training sample: $-g_i = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$. If squared-loss is considered, the formula becomes: $-g_i = y_i - F(x_i)$.

4. Construct histograms of gradients and Hessians for each feature bin rather than evaluating every possible feature value.

5. Fit a regression tree h(x;a) using the binned features and histogram statistics to determine the best splits, where a represents the tree hyperparameters.

6. Finally, update the model (τ is the learning rate): $F_m(x) = F_{m-1}(x) + \tau h(x;a)$

3.3.4.2 Metrics

For each evaluation fold of the dataset, the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) are computed, by comparing the results obtained with the expected target.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.1}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.2}$$

During testing, both evaluation metrics are valuable to understand the prediction performance: MAE measures the average magnitude of the deviations between the predicted values and the target values, while RMSE gives greater weight to larger deviations, highlighting the influence of larger prediction errors.

Chapter 4

Experimental Setup & Results

4.1 Overview

This chapter describes the experimental setup used to address the research questions introduced by this work (see Section 1.2) and reports the results of every experiment. The goal is, first, to demonstrate that it is possible to combine LLMs and AutoML into a workflow that minimizes human interaction when preparing models for energy time-series forecasting, while inferring domain knowledge from the available literature (RQ1); and, second, to evaluate whether this approach is competitive, both in implementation time and forecast quality, against predefined baselines (RQ2).

The chapter begins by presenting the forecasting use cases and their datasets (Section 4.2), which define the scope of our experiments. It then describes the knowledge base built from a collection of manually selected research papers for each use case (Section 4.3), followed by the LLM providers and models tested in the information extraction pipeline (Section 4.4). Next, it introduces the forecasting baselines and regression models considered (Section 4.5) and the resulting prototype (Section 4.6). Finally, the obtained forecasting results are reported and discussed per research question, namely RQ1 (Section 4.7) and RQ2 (Section 4.8).

4.2 Forecasting use cases & datasets

To evaluate the forecasting accuracy of the proposed framework, three representative energy forecasting domains were considered. Specifically:

- **Wind power forecasting**, for Sotavento Experimental Wind Farm (see Section 4.2.1)
- **Solar power forecasting**, for INESC TEC photovoltaic power plant (see Section 4.2.2)
- **Electricity market spot price forecasting**, for the Iberian Peninsula (MIBEL) (see Section 4.2.3)

These use cases will define the scope of the experiments, being useful to validate the feature engineering functions extracted from the scientific literature and, ultimately, to evaluate the performance of the AutoML based approach, enhanced with priors from our knowledge base.

4.2.1 Wind energy (Sotavento experimental wind farm)

To predict wind energy generation, data from an experimental wind farm located in Galicia, Spain, was used. The Sotavento Experimental Wind Farm [69] is widely employed for research purposes, and the availability of high-quality operational data makes it particularly suitable for evaluating wind power forecasting models. For this use case, the target variable is the hourly aggregated average power time-series generated by the installed wind turbines. The dataset spans from January 1st, 2014 until September 22nd, 2016.

Table 4.1 introduces the raw variables available for this dataset, which is a combination of both observed and forecasted variables, in the form of Numerical Weather Predictions (NWP). NWP data is essential for improving short-term forecast accuracy and mesoscale models such as the Weather Research and Forecasting (WRF) model generate high-resolution NWP grids suitable for renewable energy applications. All NWP-WRF data used in this work was retrieved from the MeteoGalicia THREDDS server, a publicly available service providing historical and daily forecasts of several weather variables.

Table 4.1: Raw variables available for the wind energy generation dataset

Variable	Type	Description	Units
m_speed	Observed	Wind speed	m/s
m_dir	Observed	Wind direction	[0°, 360°]
dir	Forecast	Wind direction	[0°, 360°]
dirlev1	Forecast	Wind direction at pressure level 1	[0°, 360°]
dirlev2	Forecast	Wind direction at pressure level 2	[0°, 360°]
dirlev3	Forecast	Wind direction at pressure level 3	[0°, 360°]
mod	Forecast	Wind speed module	m/s
modlev1	Forecast	Wind speed module at pressure level 1	m/s
modlev2	Forecast	Wind speed module at pressure level 2	m/s
modlev3	Forecast	Wind speed module at pressure level 3	m/s

It is important to note that both wind speed and wind direction variables correspond to observed measurements, available close to real-time. Therefore, when looking for feature engineering pipelines that consider these, one must be careful with potential data leakage (and unrealistically optimist results). To preserve a realistic operational scenario, these observed variables are therefore only used when lagged by at least 24 hours relative to the prediction time, ensuring that the model relies solely on information that would have been available when the forecast was generated.

4.2.2 Solar energy (INESC TEC)

For solar forecasting, a dataset composed of photovoltaic (PV) panel energy generation time-series from the INESC TEC facilities was used. The target variable is the hourly aggregated average power generated by all panels, which required some preliminary pre-processing, since each sample was originally recorded at a 5-minute sampling rate per PV string. The dataset spans from April

28th, 2013 to June 28th, 2016. Multi-year datasets are particularly relevant when building solar forecasting applications, since they enable a proper evaluation across different seasonalities.

Table 4.2 presents the raw variables available for this dataset which is here exclusively composed by forecasted variables, in the form of Numerical Weather Predictions (NWP) extracted from Meteogalicia THREDDS server.

Table 4.2: Raw variables available for the solar energy production dataset considered

Variable	Type	Description	Units
cfh	Forecast	Cloud cover at high levels	[0, 1]
cfl	Forecast	Cloud cover at low levels	[0, 1]
cfm	Forecast	Cloud cover at mid levels	[0, 1]
cft	Forecast	Cloud cover at mid and low levels	[0, 1]
swflx	Forecast	Surface downwelling shortwave flux	W/m^2
temp	Forecast	Temperature	K

The final set of input features used for forecasting aims to predict the power generated by the PV panels (target values).

4.2.3 Electricity Prices (MIBEL)

For electricity market price forecasting, available data from the MIBEL (Mercado Ibérico de Electricidade, or Iberian Electricity Market) was used. MIBEL is the joint wholesale electricity market operated by Portugal and Spain, in which day-ahead prices are determined through the intersection of aggregated supply and demand curves submitted by market participants for each hourly period of the following day.

The target variable for this use case is the hourly day-ahead electricity price for the Spanish bidding zone, expressed in €/MWh. Table 4.3 presents the raw variables considered for this dataset, all of which are forecast values published ahead of the market clearing session, since these are the only inputs realistically available at the time predictions must be issued.

Table 4.3: Raw variables available in the MIBEL dataset considered

Variable	Type	Description	Units
res_dmd_es_for	Forecast	Residual demand forecast for Spain	MW
solar_pv_es_for	Forecast	Solar photovoltaic generation forecast for Spain	MW
solar_thermal_es_for	Forecast	Solar thermal generation forecast for Spain	MW
wind_es_for	Forecast	Wind power generation forecast for Spain	MW

These four variables form the input feature set used for predicting the day-ahead electricity price, and further feature engineering steps applied to them are detailed in the next sections. The dataset is distributed with hourly timestamps, spanning from January 10th, 2022 until September 30th, 2025, a period that captures substantial variability in market conditions, including the extreme price volatility experienced across European electricity markets during the 2022 energy crisis, followed by a gradual stabilization in subsequent years. This makes the dataset particularly

valuable for evaluating the robustness of forecasting models under both turbulent and comparatively stable market regimes.

4.3 Knowledge base

As introduced in the Methodology Chapter (see Section 3.1), the knowledge base will be composed by a structured store of feature engineering techniques and forecasting models, extracted by LLM-based agents from a selection of scientific documents. This section introduces the document corpus from which the knowledge base was built for each forecasting use case described in Section 4.2.

4.3.1 Papers per use case

One of the first steps of this work was to manually curate a selection of scientific papers from conferences and peer-reviewed journals, covering a diverse range of use cases and spanning from reviews of forecasting applications to actual forecast competition reports (e.g., from the Global Energy Forecasting Competition, GEFCom).

Approximately 15 documents were assembled per forecasting use case, a number chosen to balance domain coverage with computational cost and LLM API usage, while remaining sufficient to evaluate the framework across different document types and formats. Appendix C lists these documents in tables, one per use case.

4.4 LLM providers

To support both local and closed-source LLM providers, the model name and API key are stored in a .env file. The main script (responsible for executing the LLM tasks) reads these values and uses them to configure an HTTP client, attaching the API key for authentication and resolving the target provider from the model name. Only then is an HTTP request made to call the LLM itself.

In Figure 4.1, it is possible to see how communication works during this process.

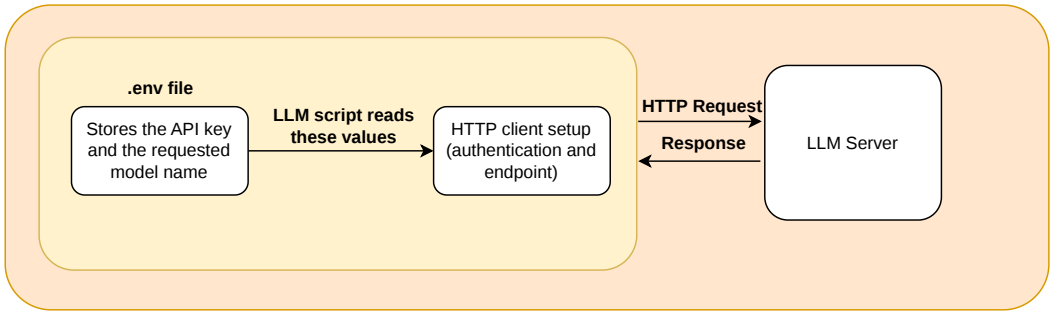


Figure 4.1: Communication operations with the LLM server

4.4.1 Local models

As described in Section 3.2.3, a local LLM was selected to perform the role of the "judge" during the integrity validation process. The model chosen for this purpose was *gemma4-31b*, based on the results of preliminary tests conducted during the LLM-agent development stage. The integrity assessment results obtained using this model are presented in Section 4.7.2.

4.4.2 Groq

Although several LLM providers were available, Groq was selected for testing due to the diversity of models offered through its platform. Using a dedicated API key, multiple Groq-hosted models were evaluated throughout the development of this work.

The results from the feature extraction and validation pipeline were obtained using the models: *meta-llama/llama-4-scout-17b-16e-instruct*, *llama-3.3-70b-versatile* and *openai/gpt-oss-120b*.

The final evaluation results with the information gathered by these 3 models are approached in section 4.8.

4.5 Time-series forecasting setup

4.5.1 Baseline selection

The two baselines described in sub-subsection 3.3.4.1 and the proposed approach share the same expanding-window cross-validation structure: 30 folds are performed, each using all data up to the test window for training, with the last 15% of that training portion reserved as a validation set. Performance is measured on the held-out test segment and reported as mean of RMSE and MAE across folds.

For Baseline 2, Bayesian optimisation is applied at each fold independently: 10 Optuna trials are run to minimize validation RMSE, after which the best configuration found is retrained on the combined training and validation data before evaluation on the test window.

4.5.2 Regression models selection

Regardless of the selection made by the user to filter the models considered for TPOT evaluation, the available models and their respective modules are listed in `sci-kit learn`'s website for supervised learning [70]. It is important to note that all selected models are compatible with scikit-learn, as TPOT only optimizes the hyperparameters of this class of models. Any other types of models reported in the literature are therefore excluded. Only the models registered in the website are considered and can be optimized within the framework.

4.6 Prototype

The prototype was implemented in Python, with dependencies managed through a dedicated virtual environment. The implementation is structured around two main scripts responsible for distinct phases of the pipeline. The first script handles information extraction and validation. Before execution, the user selects a folder corresponding to the target use case (market price, solar or wind). The papers within that folder are then processed by the LLM tasks script, which extracts regression models and feature engineering functions, applies validation checks, and writes the results to two Excel files: one for the collected models and their hyperparameters, and one for the extracted feature functions. The second script handles evaluation. It reads both Excel files, constructs the input set of features, and runs the selected baseline scenario under the cross-validation procedure defined above. All results, including mean RMSE, MAE, computational time, and the best pipeline when TPOT is used, are appended to a final Excel file at the end of execution. In Appendix D, a diagram is presented with all the tasks performed during the pipeline's execution.

4.7 Results - Research Question 1

The first Research Question that can be asked for this developed work is: "An approach based in LLMs and AutoML can minimize human intervention, infering domain knowledge and experience from existing literature automatically?" To answer this question, the following subsections will provide results gathered during the testing period.

4.7.1 Quality of results

For constructing the knowledge base, it was used a LLM-agent framework, as explained in detail in the Methodology chapter, to extract information from research documents related to the use case considered. Figure 4.2 shows an example of an extracted feature (using *meta-llama/llama-4-scout-17b-16e- instruct*) and its source from the paper.

2) *Lags and leads* $(t + k \pm z)$*:* The temporal dependency of a NWP variable in regression models can be modeled by including the preceding (lags, $t + k - z,\ z = 1, 2, \ldots$) and following (leads, $t + k + z,\ z = 1, 2, \ldots$) with respect to a specific lead-time. In previous works (c.f., [21], [24]), the inclusion of these variables resulted in accuracy improvement. In this work, the importance of this conventional approach is verified and used as benchmark in order to demonstrate the value of the new features.

functions[i]['python_implementation']

```
import pandas as pd
import numpy as np

def LagsAndLeads(df, z=24):
    df['power_lag'] = df['power'].shift(z)
    df['power_lead'] = df['power'].shift(-z)
    df['cfh_85_lag'] = df['cfh_85'].shift(z)
    df['cfh_85_lead'] = df['cfh_85'].shift(-z)
    df['cfl_85_lag'] = df['cfl_85'].shift(z)
    df['cfl_85_lead'] = df['cfl_85'].shift(-z)
    df['cft_85_lag'] = df['cft_85'].shift(z)
    df['cft_85_lead'] = df['cft_85'].shift(-z)
    df['cfm_85_lag'] = df['cfm_85'].shift(z)
    df['cfm_85_lead'] = df['cfm_85'].shift(-z)
    df['swflx_85_lag'] = df['swflx_85'].shift(z)
    df['swflx_85_lead'] = df['swflx_85'].shift(-z)
    df['temp_85_lag'] = df['temp_85'].shift(z)
    df['temp_85_lead'] = df['temp_85'].shift(-z)

    # Add lag and lead features for timestamp_utc (not applicable as it's datetime)
    return df
```

Figure 4.2: Example of an extracted function output, based on the information available in [71]

This implementation is an excellent example where the quality of results sometimes might vary, depending on the LLM. Notice that for this scenario, the evaluation prototype will not consider this implementation at all, since it uses leads in the target variable ('power'), that is uncertain beyond deployment time, despite the instructions given in Appendix B.1. This represents a challenge, since LLM doesn't always return the information as intended, despite being instructed correctly. Nevertheless, it is well implemented, based on the description provided by the research paper, making lags and leads operations in the raw variables of the dataset.

As for the models, a simple example is also given in Figure 4.3, with the extraction of the hyperparameters that are clearly described in the research document.

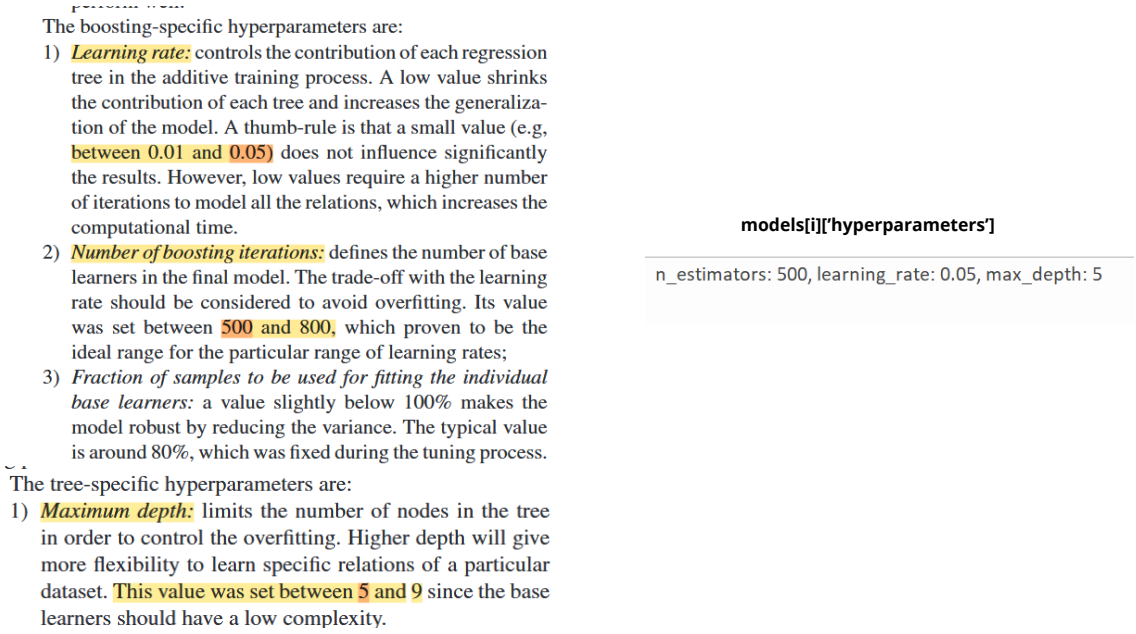


Figure 4.3: Example of an extracted model output, based on the information available in [71]

To address the problems that can be related with extraction quality, it was evaluated how many of the extracted features are actually considered at the evaluation stage. This shows how relevant the feature functions identified by each LLM are, considering the datasets available. Besides this, another performance indicator was considered, by evaluating how many documents were considered in the extraction phase, since some LLM struggle to return the JSON output structure that is requested by the issued prompt. This limits the final available functions and models for evaluation, since they don't get to be saved in the dedicated Excel file. In Table 4.4, it is possible to see the behavior of 3 knowledge bases, formed by different LLM.

Table 4.4: Research literature read by each LLM and generated features considered for testing

LLM model	Gen func solar	New input var solar	Gen func wind	New input var wind	Gen func market prices	New input var market prices	Docs considered func (%)
meta-llama/ llama-4-scout-17b-16e-instruct	152	33	155	38	152	34	95,12%
openai/gpt-oss-120b	83	8	59	20	57	14	51,22%
llama-3.3-70b-versatile	83	18	52	0	49	11	97,56%

Column definitions: Gen func solar — number of functions generated by the LLM for the solar forecasting use case. **New input var solar** — number of features among *Gen func solar* that were considered valid inputs for evaluation. **Gen func wind** — number of functions generated by the LLM for the wind forecasting use case. **New input var wind** — number of features among *Gen func wind* that were considered valid inputs for evaluation. **Gen func market prices** — number of functions generated by the LLM for the market price forecasting use case. **New input var market prices** — number of features among *Gen func market prices* that were considered valid inputs for evaluation. **Docs considered func (%)** — percentage of research documents whose generated functions were successfully parsed as JSON and therefore used to create features.

Some important observations are needed, regarding the results presented in Table 4.4. From the 3 LLM used (provided by *groq*) it was evidenced that *meta-llama-4scout-17b-16e* extracted the most feature functions, despite the low quality of implementation presented (only 22% of generated functions were evaluated in model training). On the other hand, *openai/gpt-oss-120-b* struggled in parsing functions and models into the pretended JSON format, making it difficult to take advantage from the information gathered in extraction. As for *llama-3.3-70b-versatile*, fewer functions were generated, comparing to *meta-llama-4scout-17b-16e*. For the wind use case, this LLM failed to generate code for functions, identifying mainly models only and creating most of the implementations with syntax errors.

As for the identified models, in Table 4.5 it is possible to see how each LLM performed in extracting valid models to be used in the AutoML pipeline (i.e compatible with `sci-kit learn` [70]). Despite extracting less models from literature, *openai/gpt-oss-120b* has proven to be the best in identifying valid pipelines, with valid hyperparameters, with 51% of the identified models ready to be passed as priors for the AutoML evaluation.

Table 4.5: Valid model and hyperparameter configurations found in all the models extracted from the research documents

LLM Model	Total models	Valid model & hyperparameters combinations	Valid model & hyperparameters (compatible with scikit-learn)	Docs considered models (%)
meta-llama/ llama-4-scout-17b-16e-instruct	149	94	37	87,80%
openai/gpt-oss-120b	112	91	57	75,61%
llama-3.3-70b-versatile	133	67	15	97,56%

4.7.2 Response Integrity Assessment

To assess the integrity of the responses produced by the extraction pipeline, a separate LLM was employed as a "judge": a locally hosted model, *gemma4-31b*, accessed through INESC-TEC's VPN. This judge model took no part in generating the features or the forecasting models; its role was restricted to comparing each generated item against the corresponding source paper and rating the accuracy of both its description and its Python implementation.

The "judge" model was instructed to rate the description and the implementation of every extracted feature and model according to the same four-point scale:

1) Accuracy of the description: PERFECT, GOOD, PARTIAL, INACCURATE.

- **Perfect** – The description fully and exactly matches the paper.
- **Good** – Captures the main idea with only minor omissions or inaccuracies.
- **Partial** – Reflects some aspects but misses important details or includes noticeable errors.
- **Inaccurate** – Does not reflect the paper and includes incorrect or fabricated information.

2) Accuracy of the Python implementation: PERFECT, GOOD, PARTIAL, INACCURATE.

- **Perfect** – The implementation fully matches what is described or implied in the paper.
- **Good** – Captures the core idea but lacks some details or precision.
- **Partial** – Only partially reflects the method or contains important mistakes.
- **Inaccurate** – Does not correspond to the paper or is largely fabricated.

A description that faithfully reflects the source paper does not guarantee that the accompanying code does the same. To examine this, the analysis was restricted to the artefacts whose description had been rated PERFECT or GOOD, and the share of those artefacts whose implementation was nonetheless rated INACCURATE was computed (Table 4.6). Two patterns stand out. First, the gap is consistently wider for feature-extraction functions than for forecasting models, across all three LLMs: the features are typically customized and paper-specific, and their faithful implementation depends on details that the source text often leaves under-specified, whereas the models usually correspond to standard, well-established algorithms that the LLM already knows independently of the paper at hand. Second, the size of the gap varies markedly with the generating model. For *llama-3.3-70b-versatile* and *meta-llama/llama-4-scout-17b-16e-instruct*, roughly two thirds to four fifths of the well-described functions were still implemented inaccurately (78.4% and 65.6%, respectively), so an accurate description offers little assurance about the quality of the code. *openai/gpt-oss-120b* is the clear exception, with 47.4% for functions and only 11.3% for models, indicating that for this model description accuracy is a comparatively reliable predictor of implementation accuracy.

Table 4.6: Description-implementation gap: among artefacts with a Perfect or Good description, the share whose code implementation was rated Inaccurate

Artefact type	LLM	*n* (Perfect/Good desc.)	Inaccurate impl. (%)
Feature-extraction functions	llama-3.3-70b-versatile	162	78.4
	meta-llama/llama-4-scout-17b-16e-instruct	163	65.6
	openai/gpt-oss-120b	116	47.4
Forecasting models	llama-3.3-70b-versatile	94	45.7
	meta-llama/llama-4-scout-17b-16e-instruct	90	42.2
	openai/gpt-oss-120b	62	11.3

Taken together, these results suggest that, even when the description is faithful to the source, the correctness of the generated Python implementation depends primarily on the coding capabilities of the LLM that produced it. This assessment should nevertheless be read as an exploratory experiment rather than as an empirically robust conclusion. Using one LLM to evaluate the outputs of another inherits the very failure mode under investigation: the judge model may itself hallucinate, misread the source paper, or apply the rating scale inconsistently, and no human-annotated reference set was available against which its ratings could be validated. The absolute values reported in Table 4.6 should therefore be interpreted as indicative rather than definitive.

Extending this evaluation is a natural direction for future work. The judge could be incorporated into the pipeline as a dedicated LLM-agent node, with a second, independent model acting as a supervisor that re-evaluates a sample of the judge's ratings and flags those that appear unfounded. Calibrating the judge against a small set of manually annotated artefacts would provide a further, and stronger, validation of its reliability.

4.7.3 Adaptability

The proposed extraction approach was evaluated across three distinct use cases (solar, wind, and electricity market price forecasting) requiring no structural changes to the pipeline between domains. This cross-domain applicability demonstrates that the system adapts its knowledge extraction to different forecasting contexts automatically, with the only user intervention being the provision of domain-relevant literature and the corresponding dataset.

As shown in Table 4.4, all three LLMs generated use-case-specific features and models across every domain, confirming that the literature-driven extraction generalizes beyond a single forecasting context. However, adaptability is not uniform across models: *llama-3.3-70b-versatile* failed to produce valid feature functions for the wind use case (0 new input variables generated).

This illustrates that the system's adaptability is bounded by LLM reliability rather than by the pipeline architecture itself, and that LLM selection is a crucial decision in this framework.

Regarding human intervention, the pipeline is designed to operate autonomously through its core stages: literature processing, feature and model extraction, code generation, AutoML configuration and final evaluation. Manual involvement is limited to two boundary points: the initial storage of input literature, and a final validation of the results obtained.

4.7.4 Conclusions

To answer RQ1 directly, the evidence gathered in subsections 4.7.1, 4.7.2, and 4.7.3 supports the following conclusions.

Subsection 4.7.1 demonstrates that the proposed approach successfully transforms unstructured domain knowledge from research literature into structured, ready-to-use outputs (feature engineering functions and model configurations). Table 4.4 shows that this extraction step consistently produced use-case-specific features and models across all three domains, though the proportion of source documents actually contributing usable output ranged from 51.22% to 97.56% depending on the LLM. It also shows that generation volume is a poor indicator for usefulness: *meta-llama/llama-4-scout-17b-16e-instruct* generated by far the largest number of functions, yet only a small fraction of them survived as valid inputs for evaluation. Table 4.5 also shows that, while only a portion of extracted models is currently passed to the AutoML stage (due to the `scikit-learn` compatibility requirement imposed by TPOT), this constraint is architectural rather than fundamental: adopting an alternative AutoML framework would allow a different range of extracted models to be evaluated.

This picture of successful, but incomplete, automation is reinforced by the results of Subsection 4.7.2. By restricting the analysis to the artefacts whose description was judged faithful to the source paper, Table 4.6 shows that a well-described artefact is not necessarily a correctly implemented one. The gap is systematically wider for feature-extraction functions than for forecasting models, which is consistent with the former being customized and paper-specific while the latter typically correspond to standard algorithms already familiar to the LLM. For *llama-3.3-70b-versatile* and *meta-llama/llama-4-scout-17b-16e-instruct*, 78.4% and 65.6% of the well-described functions were nonetheless rated as inaccurately implemented, so an accurate description offered little guarantee of usable code. These findings indicate that, in its current form, the pipeline does not eliminate the need for human intervention in the code-generation step: a substantial share of the automatically generated implementations, particularly for customized feature-extraction functions, would still require manual review or correction before being trusted in a production forecasting system. This limitation was, however, shown to be strongly LLM-dependent rather than intrinsic to the approach itself, with *openai/gpt-oss-120b* exhibiting a considerably narrower gap (47.4% for functions and only 11.3% for models), indicating that careful selection of the underlying generation model can substantially reduce the amount of code requiring human validation.

These integrity results should nevertheless be read with caution. As discussed in Subsection 4.7.2, they were obtained from a single LLM acting as judge, without a human-annotated reference set against which its ratings could be calibrated, and therefore constitute an exploratory indication rather than a definitive measurement.

Subsection 4.7.3 confirms that the knowledge-extraction capability itself extends across multiple forecasting domains without structural reconfiguration, with residual human intervention, specially at the initial provision of literature and dataset, and at a final validation step. However, this same subsection shows that adaptability, like implementation accuracy, is bounded by the

reliability of the underlying LLM rather than by the pipeline architecture.

These conclusions from the subsections support a qualified affirmative answer to RQ1: LLMs can effectively infer domain knowledge from existing literature and translate it into structured, actionable inputs for an AutoML pipeline across multiple forecasting domains, without requiring changes to the pipeline architecture between domains. The extent to which this reduces manual effort, however, depends heavily on the specific LLM used to generate the implementations, and, in its current form, the approach reduces rather than eliminates human intervention. The description-implementation gap reported in Subsection 4.7.2 suggests that a non-trivial share of generated code, especially for feature-extraction functions, is not directly usable, and that the pipeline would benefit from an automated verification stage, such as the supervised judge node proposed in Subsection 4.7.2, capable of flagging low-confidence artefacts before they are passed as priors to the AutoML stage.

4.8 Results - Research Question 2

For the second Research Question, the problem of the proposed approach's competitiveness is addressed:

"Will that approach be competitive, in implementation time and forecasting quality, compared to traditional methods?"

4.8.1 Evaluation baselines

To compare the obtained results regarding forecasting accuracy, the baseline values for the MAE and RMSE were computed, considering the respective datasets for each use case considered (see Table 4.7).

Table 4.7: Baseline Scenario Performance Comparison Across Multi-Domain Metrics

Baseline scenario	**Solar RMSE**	**Solar MAE**	**Wind RMSE**	**Wind MAE**	**Market Price RMSE**	**Market Price MAE**
Baseline 1 (raw features as input)	1596,60	789,69	2624,23	1953,95	61,01	52,62
Baseline 2 (raw features as input & hyperparameter tuning with bayesian optimization)	1563,07	808,15	2484,91	1863,93	56,92	49,10

Note that for every comparison made through the following subsections, $\searrow$ indicates a downgrade and $\nearrow$ an improvement in performance, in result comparison.

4.8.2 Solar power forecasting

4.8.2.1 Solar power forecasting with the full search space

For solar power forecasting, different results were obtained by the considered LLM's , specially due to the quality of the generated features. In Table 4.8, the input variables considered for evaluation and the results obtained for each knowledge base constructed by each LLM are shown.

The used features are grouped into different categories, for better readability:

- **Calendar features**: Includes cyclical/temporal indicators of the forecast timestamp:
  ['hour', 'dayofweek', 'month']
- **Cyclical calendar features**: ['sin_day', 'cos_day', 'sin_dayofweek', 'cos_dayofweek', 'sin_hour', 'cos_hour']
- **Raw variables**: Includes atmospheric value predictions available from the dataset:
  ['cfh_85', 'cfl_85', 'cft_85', 'cfm_85', 'swflx_85', 'temp_85']
- **Previous-day features (24h lag)**: Includes the value of the raw variables (and power) from the same hour on the previous day:
  ['prev_day_power', 'prev_day_irradiation', 'prev_day_temp_85', 'prev_day_cfl_85', 'prev_day_cfm_85', 'prev_day_cfh_85', 'prev_day_cft_85']
- **Lag features (1h)**: Includes shifted values of the raw variables 1h from the current time of evaluation :
  ['lag1_cfh_85', 'lag1_swflx_85', 'lag1_cft_85', 'lag1_cfm_85', 'lag1_temp_85', 'lag1_cfl_85']
- **Lag features (2h)**: Includes shifted values of the raw variables 2h from the current time of evaluation:
  ['lag2_cfh_85', 'lag2_swflx_85', 'lag2_cft_85', 'lag2_cfm_85', 'lag2_temp_85', 'lag2_cfl_85']
- **Lag calendar features (2h)**: Shifted (2h) values of the calendar variables relative to the evaluation time. These features do not carry any additional predictive information beyond the original calendar variables counterparts, since they are deterministic and fully known in advance; they were included inadvertently as a consequence of applying the 2h lagging procedure to all variables:
  ['month_lag_2', 'dayofweek_lag_2', 'hour_lag_2']
- **Rolling mean features(24h)**: Includes the rolling average of raw variables and power, computed over the previous 24h from the day before the current time of evaluation.
  ['cfm_85_rolling_mean', 'temp_85_rolling_mean', 'cfl_85_rolling_mean', 'cft_85_rolling_mean', 'swflx_85_rolling_mean', 'cfh_85_rolling_mean', 'power_rolling_mean', 'rolling_mean_wind_speed']
- **Rolling max features (24h)**: Includes the maximum value reached by each variable over the previous 24h from the day before the current time of evaluation:
  ['max_power_24h', 'max_swflx_85_24h', 'max_temp_85_24h', 'max_cfh_85_24h', 'max_cfl_85_24h', 'max_cfm_85_24h', 'max_cft_85_24h']
- **Standard deviation features**: Includes the standard deviation computed between different forecasting variables:
  ['cross_variable_std_dev']

- **Normalized inputs**: Normalization of prediction values with its maximum and minimum values:

  ['cfh_85_min_max_norm']: Normalization of the cloud cover at high levels with its maximum and minimum values.

  ['normalize_swflx']: Normalization of the cloud cover at high levels with its maximum and minimum values.

- **Other features**: Includes remaining engineered variables that do not belong to the previous categories:

  ['power_shift_28']: Value of generated power shifted by 28h

  ['swflx_85_squared']: The squared value of the irradiance for every timestamp

  ['cloud_fraction_ratio']: Computes the ratio between cloud cover at high levels and the difference between the cloud cover at mid and low levels and the cloud cover at low levels.

  ['cloud_fraction_ratio_24h']: Computes the 24h lag of 'cloud_fraction_ratio'.

Table 4.8: RMSE and MAE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2

Model	Input features	RMSE	MAE
Baseline 1	Raw features as input	1596,60	789,69
Baseline 2	Raw features as input & hyperparameter tuning with bayesian optimization	1563,07	808,15
meta-llama/ llama-4-scout-17b-16e- instruct	Calendar features, 'sin_day', 'cos_day' Raw variables, 'previous_day_power', 'previous_day_irradiation', Lag features (1h), Lag features (2h), Rolling mean features (24h), Rolling max features (24h), Standard deviation features, 'power_shift_28'	1272,22 (↗18,61%)	689,28 (↗14,71%)
openai/gpt-oss-120b	Calendar features, Cyclical calendar features, Raw variables, 'prev_day_power', 'prev_day_temp_85', 'normalize_swflx', 'cfh_85_min_max_norm'	1562,18 (↗0,057%)	780,71 (↗3,38%)
llama-3.3-70b-versatile	Calendar features, Raw variables, Lag Calendar features (2h), Previous-day features (24h lag), Lag features (2h), 'power_rolling_mean', 'swflx_85_squared', 'cloud_fraction_ratio', 'cloud_fraction_ratio_24h'	1205,22 (↗22,89%)	578,20 (↗28,45%)

The best result was generated with *llama 3.3 70B*, with a mean RMSE of 1205.22 W and a mean MAE of 578.20 W, with a Random Forest Regressor (*max_depth* = 15, *min_samples_leaf* = 2, *n_estimators* = 165, *random state* = 42) across all folds in the cross-validation evaluation. For this approach, Figure 4.4 shows the prediction results for the last temporal fold of the dataset.

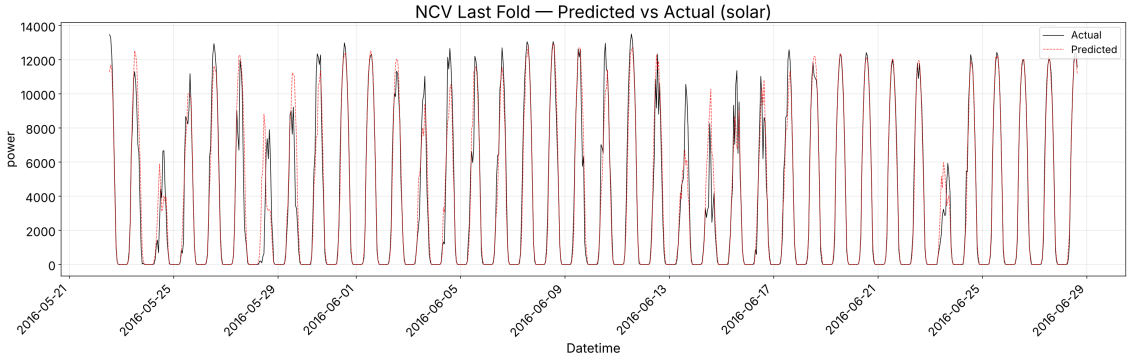


Figure 4.4: Predicted values (in red) vs actual values (in black) for generated power (in W), in the last fold of evaluation for cross-validation

4.8.2.2 Solar Power Forecasting considering a restricted search space

Considering a restricted search space to the top five most referenced models and the functions that showed improvement at the LLM-agent framework stage, through the 'improves_forecasting' flag. the resulting RMSE and MAE values are shown in Table 4.9, as well as the improvement of the error, against the Baseline 2 scenario and the improvement or downgrade of the computational time when compared to the previously obtained in sub-subsection 4.8.2.1.

Table 4.9: Computational time, MAE and RMSE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 results, and computational time of the whole pipeline (s)

Model	**RMSE**	**MAE**	**Excluded features**	**Computational time(s)**	**nº total features**
Baseline 1	1596,60	789,69	[]	13,86	9
Baseline 2	1563,07	808,15	[]	57,00	9
meta-llama/ llama-4-scout-17b-16e-instruct (full search space)	1272,22 (↗18,61%)	689,28 (↗14,71%)	[]	3939,877	42
meta-llama/ llama-4-scout-17b-16e-instruct (restricted search space)	1188,72 (↗23,95%)	616,679 (↗23,69%)	['sin_day', 'cos_day']	3203,16	41 → 39
openai/gpt-oss-120b (full search space)	1562,18 (↗0,05%)	780,71 (↗3,38%)	[]	4610,19	17
openai/gpt-oss-120b (restricted search space)	1562,26 (↗0,05%)	778,67 (↗3,65%)	['previous_day_temp_85', 'sin_hour', 'cos_hour', 'sin_dayofweek']	4342,61	17 → 13
llama-3.3-70b-versatile (full search space)	1205,22 (↗22,89%)	578,20 (↗28,45%)	[]	5145,91	27
llama-3.3-70b-versatile (restricted search space)	1200,86 (↗23,17%)	576,68 (↗28,65%)	['cloud_fraction_ratio', 'cloud_fraction_ratio_24h']	4236,87	27 → 25

For this search space, the best results were obtained for *meta-llama-4-scout-17b-16e-insctruct*, considering a Gradient Boosting Regressor (learning_rate = 0.2428, max_depth = 4, n_estimators = 101).

Comparing the results to the ones obtained in 4.8.2.1, the restriction to the best identified functions and most referenced models showed improvement, by getting closer results to the actual power output.

This scenario then becomes the best one identified, and it will be the one considered for competitiveness analysis. Figure 4.5 represents visually the difference between the actual and the predicted values.

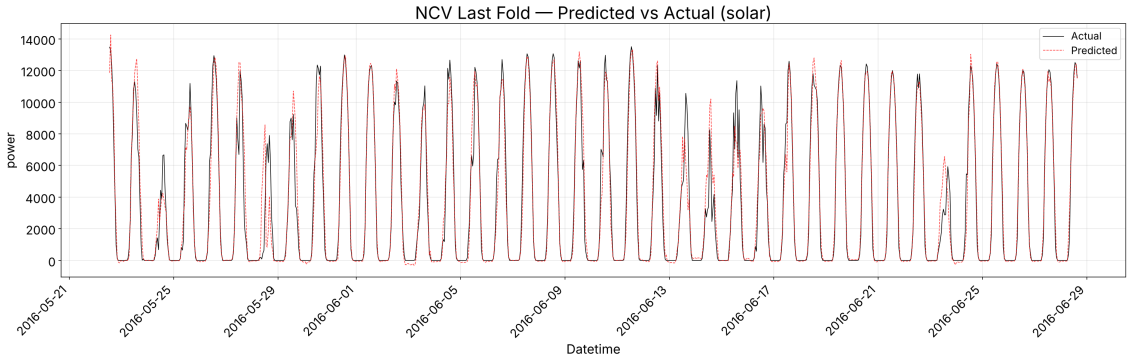


Figure 4.5: Predicted values (in red) vs actual values (in black) for generated power (in W), in the last fold of evaluation for cross-validation

4.8.3 Wind Power Forecasting

4.8.3.1 Wind Power Forecasting with the full search space

For wind power forecasting, it was observed that, from the 3 LLM selected for extraction, only *meta-llama-4scout-17b-16e* produced reasonable results. In Table 4.10, the input features considered are presented for each LLM, as well as the resulting improvement or downgrade in performance, facing the baseline scenario 2 (raw features and bayesian optimization). The *llama-3.3-70b-versatile* is not considered for comparison, since it failed in identifying new features for wind forecasting, as shown in subsection 4.7.1.

The considered feature sets for prediction are presented below, grouping similar features into the same category for better readability:

- **Calendar features**: Includes temporal indicators of the forecast timestamp:

  ['hour', 'dayofweek', 'month']

- **Cyclical calendar features**: ['sin_day', 'cos_day','sin_dayofweek', 'cos_dayofweek']

- **Raw variables**: Includes wind speed and direction predictions (at surface and pressure levels) available from the dataset:

['dir_85', 'dirlev1_85', 'dirlev2_85', 'dirlev3_85', 'mod_85', 'modlev1_85', 'modlev2_85', 'modlev3_85']

- **Previous-day features (24h lag)**: Includes the value of the raw variables and direct measurement variables, such as wind speed and wind direction, from the same hour on the previous day:

  ['previous_day_power', 'previous_day_m_speed', 'previous_day_m_dir', 'previous_day_dir_85', 'previous_day_mod_85']

- **Lag features (180h)**: Includes shifted values of the raw variables 180h from the current time of evaluation: [dir_85_lag_180h, dirlev1_85_lag_180h, dirlev2_85_lag_180h, dirlev3_85_lag_180h, mod_85_lag_180h, modlev1_85_lag_180h, modlev2_85_lag_180h, modlev3_85_lag_180h, 'hour_lag_180h', 'dayofweek_lag_180h', 'month_lag_180h']

- **Rolling mean features (24h)**: Includes the rolling average of raw variables and power, computed over a 24-hour moving window:

  ['power_rolling_mean_24h', 'dir_85_rolling_mean_24h', 'dirlev1_85_rolling_mean_24h', 'dirlev2_85_rolling_mean_24h', 'dirlev3_85_rolling_mean_24h', 'mod_85_rolling_mean_24h', 'modlev1_85_rolling_mean_24h', 'modlev2_85_rolling_mean_24h', 'modlev3_85_rolling_mean_24h']

- **Rolling standard deviation features (24h)**: Includes the rolling standard deviation of raw variables and power, capturing local variability over a 24-hour moving window:

  ['power_rolling_std_24h', 'wind_speed_rolling_std_24h', 'dir_85_rolling_std_24h', 'dirlev1_85_rolling_std_24h', 'dirlev2_85_rolling_std_24h', 'dirlev3_85_rolling_std_24h', 'mod_85_rolling_std_24h', 'modlev1_85_rolling_std_24h', 'modlev2_85_rolling_std_24h', 'modlev3_85_rolling_std_24h']

- **Categorical (binned) features**: Includes discretized encodings of the corresponding continuous wind speed and direction variables:

  ['m_dir_binned', 'dir_85_binned', 'mod_85_binned', 'dirlev1_85_binned', 'dirlev2_85_binned', 'dirlev3_85_binned', 'modlev1_85_binned', 'modlev2_85_binned', 'modlev3_85_binned']

- **Other features**:

  Includes remaining engineered variables that do not belong to the previous categories:

  ['power_lag_204h']: Shifted values of the generated power, 204h from the time of evaluation.

  ['m_speed_lag_34h', 'm_speed_lag_28h']: Multi-horizon lagged wind speed features: Includes observed wind speed lagged by longer horizons (28 and 34 hours), following the multi-lag input structure commonly used for neural-network-based forecasting.

Table 4.10: RMSE and MAE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2

Model	Input features	RMSE	MAE
Baseline 1	Raw features as input	2624,23	1953,95
Baseline 2	Raw features as input & hyperparameter tuning with bayesian optimization	2484,91	1863,93
meta-llama/ llama-4-scout-17b-16e-instruct	Calendar features, Raw variables, Previous-day features (24h lag), Rolling mean features (24h), Rolling standard deviation features (24h), Categorical (binned) features, 'm_speed_lag_34h', 'm_speed_lag_28h', 'sin_day', 'cos_day'	2492,75 (↘0,31%)	1753,78 (↗5,91%)
openai/gpt-oss-120b	Calendar features, Cyclical calendar features, Raw variables, Lag features (180h), 'previous_day_power', 'previous_day_m_speed', 'previous_day_m_dir', 'power_lag_204h'	3501,643 (↘40,92%)	2555,88 (↘37,12%)

It is important to note that some of the LLM generated features may have introduced noise into the input space as a consequence of inaccurate extracted implementations (see Section 4.7.2). This applies to *llama-4-scout-17b-16e-instruct* and in an even worse scenario for *openai/gpt-oss-120b*, which failed to produce a set of significant features. In particular, the incorrectly extracted lag operations (notably those at 180h and 204h) are likely to have degraded performance, since lags applied over inappropriate horizons introduce false temporal relationships that the model may learn as genuine patterns, ultimately worsening forecasting accuracy.

As for the resulting graphs, Figure 4.6 shows how the real and the predicted values differ in a cross-validation evaluation

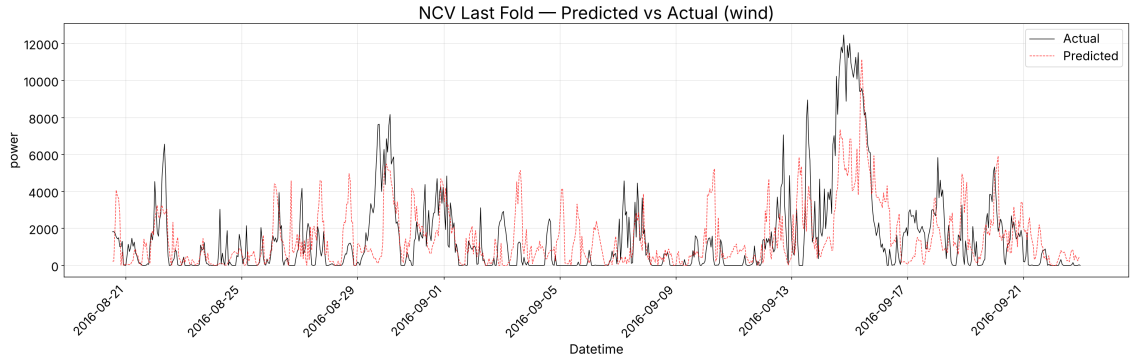


Figure 4.6: Predicted values (in red) vs actual values (in black) for power generation (in kW) in the last fold of evaluation for cross-validation

4.8.3.2 Wind power forecasting considering a restricted search space

For wind power forecasting, it was evidenced some improvements for result accuracy, as shown in Table 4.11.

Table 4.11: Computational time, MAE and RMSE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 results, and computational time of the whole pipeline (s)

Model	RMSE	MAE	Excluded features	Computational time(s)	nº total features
Baseline 1	2624,23	1953,95	[]	12,40	11
Baseline 2	2484,91	1863,93	[]	57,00	11
meta-llama/ llama-4-scout-17b-16e-instruct (full search space)	2492,75 (↘ 0,31%)	1753,78 (↗ 5,91%)	[]	3536,36	49
meta-llama/ llama-4-scout-17b-16e-instruct (restricted search space)	1928,62 (↗22,39%)	1402,27 (↗24,77%)	['previous_day_m_dir', 'previous_day_dir_85', 'previous_day_mod_85', 'shift_m_speed_24', Rolling mean features (24h)]	3930,68	49 → 37
openai/gpt-oss-120b (full search space)	3501,643 (↘ 40,92%)	2555,88 (↘ 37,12%)	[]	3674,28	31
openai/gpt-oss-120b (restricted search space)	3048,31 (↘22,66%)	2213,61 (↘18,76%)	Lag features (180h), 'cos_dayofweek'	4244,56	31 → 17

With this running conditions, some identified functions were discarded and therefore improved forecasting accuracy. Unlike what happened in sub-subsection 4.8.3.1, the results obtained for *meta-llama-4-scout-17b-16-e instruct*, with a Random Forest Regressor (*n_estimators=101, random_state=41*) are meaningful towards the best baseline's error scores, improving the baseline RMSE by 22,39% and the baseline MAE by 24,77%.

Although *openai/gpt-oss-120b* didn't show improvement towards the baseline, the results obtained were still better than what was registered in sub-subsection 4.8.3.1.

Figure 4.7 represents the results of the last fold of evaluation for time series cross validation.

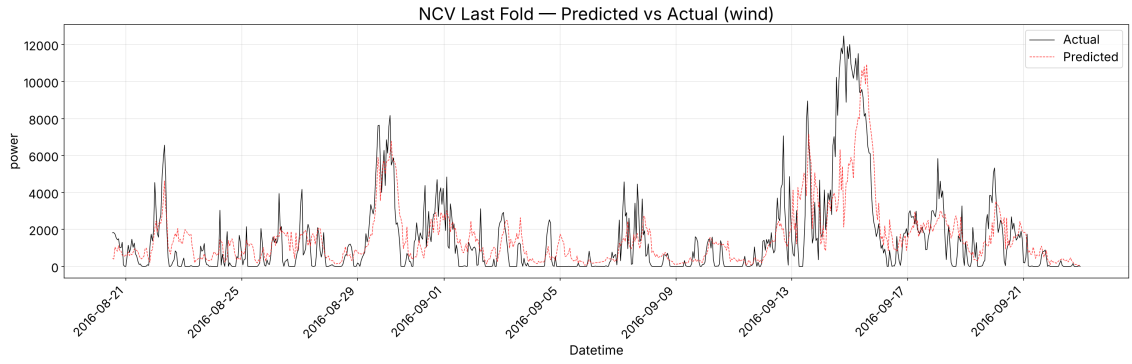


Figure 4.7: Predicted values (in red) vs actual values (in black) for generated power (in kW) for the last fold of evaluation in cross-validation

For competitiveness purposes, these results are the ones to consider, since this search space managed to extract the full potential of the gathered knowledge base.

4.8.4 Electricity Market Prices Forecasting

4.8.4.1 Electricity Market Prices forecasting with the full search space

For the electricity market price forecasting, three LLM were used to extract information, resulting in the input features and errors shown in Table 4.12.

For better readability, the extracted features were grouped into feature sets, that are listed below:

- **Calendar features**: Includes temporal indicators of the forecast timestamp:

  ['hour', 'dayofweek', 'month']

- **Cyclical calendar features**: Sine and cosine encodings applied to the periodic calendar variables, preserving their circular nature:

  ['sin_day', 'sin_hour', 'cos_hour', 'sin_dayofweek', 'cos_dayofweek']

- **Raw variables**: Includes the day-ahead residual demand and renewable generation forecasts for Spain available from the dataset:

  ['res_dmd_es_for', 'solar_pv_es_for', 'solar_thermal_es_for', 'wind_es_for']

- **Previous-day features (24h lag)**: Includes the value of the raw variables and of the market price from the same hour on the previous day:

  ['previous_day_actual_price', 'previous_day_res_dmd_es_for', 'previous_day_solar_pv_es_for', 'previous_day_solar_thermal_es_for', 'previous_day_wind_es_for']

- **Lag features (1h)**: Includes shifted values of the raw variables 1h from the current time of evaluation:

  ['lag1_res_dmd_es_for', 'lag1_solar_pv_es_for', 'lag1_solar_thermal_es_for', 'lag1_wind_es_for']

- **Lag features (2h)**: Includes shifted values of the raw variables 2h from the current time of evaluation:

  ['lag2_res_dmd_es_for', 'lag2_solar_pv_es_for', 'lag2_solar_thermal_es_for', 'lag2_wind_es_for']

- **Rolling mean features (24h)**: Includes the rolling average of the raw variables and of the market price, computed over a 24-hour moving window:

  ['actual_price_rolling_mean_24h', 'res_dmd_es_for_rolling_mean_24h', 'solar_pv_es_for_rolling_mean_24h', 'solar_thermal_es_for_rolling_mean_24h', 'wind_es_for_rolling_mean_24h']

- **Rolling window statistics (24h)**: Includes the maximum, minimum and median values reached by the wind generation forecast over a 24-hour moving window:

  ['wind_es_for_rolling_max_24h', 'wind_es_for_rolling_min_24h', 'wind_es_for_rolling_median_24h']

- **Log lag-ratio features**: Includes the logarithm of each variable divided by the value of that same variable 24 hours earlier:

  ['log_lag_ratio_actual_price', 'log_lag_ratio_res_dmd_es_for', 'log_lag_ratio_solar_pv_es_for', 'log_lag_ratio_solar_thermal_es_for', 'log_lag_ratio_wind_es_for']

- **Derived generation and demand aggregates**: Includes variables constructed by arithmetically combining the raw demand and generation forecasts:

  ['residual_demand_minus_wind']: Difference between the residual demand forecast and the wind generation forecast.

  ['sum_demand_and_generation_es']: Sum of the residual demand forecast and the three renewable generation forecasts.

  ['mean_squared_generation_mw2']: Row-wise mean of the squared photovoltaic, solar thermal and wind generation forecasts.

  ['tzdif']: Difference between the residual demand and solar photovoltaic generation forecasts.

- **Aggregated price history features**: Includes averages of the market price computed over past windows, together with differences between such averages:

  ['avg_price_30h', 'avg_monthly_futures_price_30h', 'diff_avg_price', 'avg_forecast_plus_lagged_price']

- **Other features**:

  ['wind_es_for_diff_1h']: First-order difference of the wind generation forecast between consecutive time steps.

  ['wind_es_for_minmax']: Wind generation forecast rescaled to the [0, 1] range through min-max normalization.

  ['actual_price_lag_28h', 'actual_price_lag_48h']: Market price shifted by horizons longer than one day.

Table 4.12: Model Performance Evaluation with different sets of input features and respective improvement, in percentage, in comparison to the baseline results

Model	**Input features**	**RMSE**	**MAE**
Baseline 1	Raw features as input	61,01	52,62
Baseline 2	Raw features as input & hyperparameter tuning with bayesian optimization	56,92	49,10
meta-llama/ llama-4-scout-17b-16e-instruct	Calendar features, Raw variables, Previous-day features (24h), Lag features (1h), Lag features (2h) Rolling mean features (24h), Log lag-ratio features, 'sin_day', 'cos_hour', 'residual_demand_minus_wind', 'sum_demand_and_generation_es', 'mean_squared_generation_mw2', 'avg_forecast_plus_lagged_price', 'actual_price_lag_28h'	25,07 (↗55,96%)	19,10 (↗61,10%)
openai/gpt-oss-120b	Calendar features, Raw variables, Rolling window statistics 24h, 'sin_hour', 'cos_hour', 'sin_dayofweek', 'cos_dayofweek', 'previous_day_actual_price', 'wind_es_for_rolling_mean_24h', 'actual_price_rolling_mean_24h', 'wind_es_for_diff_1h', 'actual_price_lag_48h', 'wind_es_for_minmax'	34,72 (↗39,00%)	27,54 (↗43,91%)
llama-3.3-70b-versatile	Calendar features, Raw variables, Previous-day features (24h lag), 'tzdif', 'avg_price_30h', 'avg_monthly_futures_price_30h', 'diff_avg_price'	31,51 (↗44,64%)	26,63 (↗45,77%)

The best result was generated with the information extracted by the model *meta-llama/ llama-4-scout-17b-16e- instruct*, with a mean RMSE of 25,07 and a mean MAE of 19,10, for a Gradient Boosting Regressor (*learning_rate = 0,42, max_depth = 4, n_estimators = 101*) across all folds in the cross-validation evaluation. For this set of model and variables, Figure 4.8 shows the prediction results for the last temporal fold of the dataset.

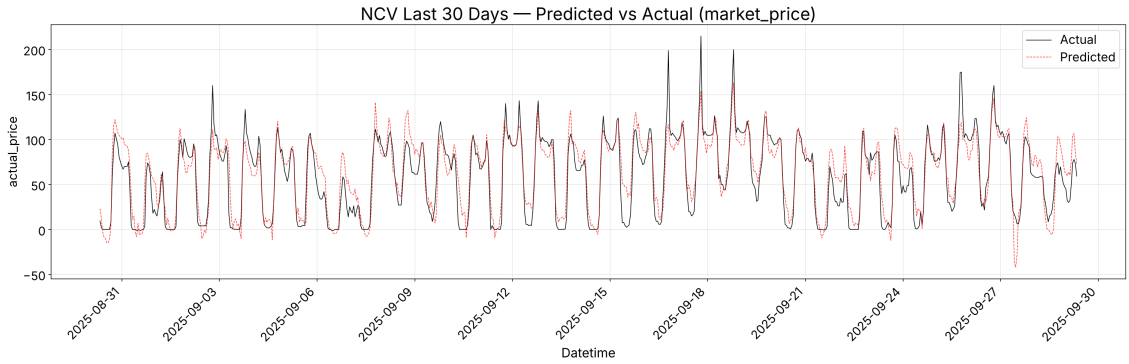


Figure 4.8: Predicted values (in red) vs actual values(in black) for market price (in €/MWh) for the last fold of evaluation in cross-validation

4.8.4.2 Electricity Market Prices forecasting considering a restricted search space

Upon consideration of a tighter search space, for both models and features, it was verified that, despite a notable improvement in computational time for most of the scenarios, the model's accuracy lowered, due to a misleading classification of features that improve forecasting for the market price forecasting use case. Table 4.13 shows the result of the trade-off, comparing the values obtained to the ones in subsection 4.8.4.1.

Table 4.13: Computational time, MAE and RMSE values obtained for every scenario and their improvement, or downgrade in performance, relative to Baseline 2 results, and computational time of the whole pipeline (s)

Model	RMSE	MAE	Excluded features	Computational time(s)	nº total features
Baseline 1	61,01	52,62	[]	12,34	7
Baseline 2	56,92	49,10	[]	69,94	7
meta-llama/ llama-4-scout-17b-16e-instruct (full search space)	25,07 (↗55,96%)	19,10 (↗61,10%)	[]	4221,30	41
meta-llama/ llama-4-scout-17b-16e-instruct (restricted search space)	29,19 (↗48,71%)	23,4 (↗52,34%)	'sin_day', Rolling mean features (24h), Log lag-ratio features, 'avg_forecast_plus_lagged_price', 'sum_demand_and_generation_es'	786,85	41 → 28
openai/gpt-oss-120b (full search space)	34,72 (↗39,00%)	27,54 (↗43,91%)	[]	4148,28	21
openai/gpt-oss-120b (restricted search space)	37,81 (↗33,58%)	30,41 (↗38,06%)	Rolling window statistics (24h), 'wind_es_for_rolling_mean_24h', 'actual_price_rolling_mean_24h', 'wind_es_for_minmax', 'sin_hour', 'residual_demand_minus_wind']	4371,06	21 → 13
llama-3.3-70b-versatile (full search space)	31,51 (↗44,64%)	26,63 (↗45,77%)	[]	5242,20	18
llama-3.3-70b-versatile (restricted search space)	38,34 (↗32,65%)	31,07 (↗36,72%)	Previous-day features (24h lag), 'sum_demand_and_generation_es', 'diff_avg_price', 'avg_price_30h'	4039,25	18 → 10

The pipeline identified that led to better results, between the different restricted search spaces, was found by *meta-llama-4-scout-17b-16e-instruct*: MLPRegressor(*hidden_layer_sizes=101*).

Figure 4.9 represents the results of the last fold of evaluation for time series cross validation.

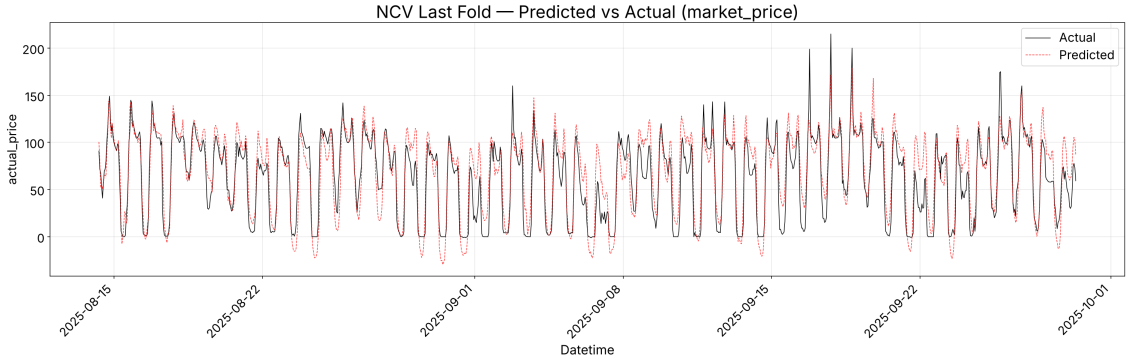


Figure 4.9: Predicted values (in red) vs actual values(in black) for market price (in €/MWh) for the last fold of evaluation in cross-validation

It is then possible to conclude that the market price use case didn't benefit, in forecasting accuracy (comparing to the full search space results), from these restrictions. On the other hand, the computational time was lower for most of the scenarios, due to less load of hyperparameter optimization.

4.8.5 Competitiveness

This subsection compares the forecasting accuracy obtained by the proposed approach against reference values reported in the literature, for the solar and wind energy generation forecasts. As for market price forecasting, it is only addressed the values obtained against the strongest baseline presented.

For solar power forecasting, considering a peak installed PV capacity of 16.32 kW, the proposed approach (restricted search space) achieved normalized errors of MAE = 3.78% and RMSE = 7.28%. As an additional comparison point, night periods were removed from the dataset and the evaluation repeated, yielding MAE = 7.97% and RMSE = 11.67%. Against the values reported by [71] for one year of history on the same dataset (normalized MAE = 6.11%, RMSE = 9.73%), neither result represents an improvement, suggesting that there remains room to increase the efficiency of the proposed approach, namely by more effectively filtering out harmful or poorly optimized candidate functions and models, potentially through improved LLM prompting strategies.

For wind power forecasting, considering an installed capacity of 17.56 MW, the restricted search space produced normalized errors of MAE = 7.99% and RMSE = 10.98%, compared to MAE = 6.22% and RMSE = 8.94% reported by [71] for the same site. Although these results fall short of the reference values, competitiveness could plausibly be improved with a broader base of

domain knowledge, since only a limited portion of the extracted literature could be leveraged for this use case due to parsing failures and inaccuracies in the LLM-generated responses.

For electricity market price forecasting (MIBEL), it was made a direct comparison with the baseline scenarios that were computed for the market price dataset considered. As reported in sub-subsection 4.8.4.2, the best result observed was with a full search space for the AutoML, with an RMSE of 25.07 €/MWh that corresponds to a 55,96% of improvement towards the baseline 2 scenario.

Overall, the proposed approach didn't achieve an ideal competitive forecasting accuracy across all three domains evaluated. However, solar power and market price forecasting show a comparatively narrower gap to their respective benchmarks and therefore greater potential for improvement, whereas wind power forecasting remains the most challenging use case for the proposed approach.

4.8.6 Computational effort

The second Research Question concerns competitiveness not only in forecasting quality, but also in computational time. This subsection therefore quantifies the computational cost of the proposed approach and positions it against the cost of the baseline scenarios evaluated under the same conditions.

The total workload divides into two phases with distinct cost profiles. The first is the extraction of the knowledge base, in which the selected LLM parses the research documents and identifies candidate models, hyperparameter ranges and engineered features. This cost is incurred once per knowledge base and is subsequently amortised over every forecast produced. It is not a stable indicator, since it depends jointly on the parsing capability of the model and on the size of the documents considered, but it remained between 10 and 25 minutes across all observations. The second phase comprises the AutoML-based hyperparameter optimization and the subsequent cross-validation evaluation, which is repeated per use case and dominates the total runtime.

Table 4.14 reports the time spent by the pipeline on this second phase for every use case and search-space condition, alongside the two baseline scenarios that led to the results presented in subsection 4.8.5. All measurements were obtained on the same machine, so the values are directly comparable with one another. They do, however, cover different amounts of work: the baselines fit a model whose structure is fixed in advance (with Baseline 2 additionally performing a per-fold Bayesian optimization of its hyperparameters), whereas the two search-space conditions cover a complete pipeline comprising model search, hyperparameter optimization and cross-validated evaluation.

Table 4.14: Computational effort of the proposed approach across use cases and search-space conditions, compared with the baseline scenarios

Use case	Baseline 1 (raw features)	Baseline 2 (raw features + Bayesian opt.)	Full search space	Restricted search space
Solar	13,86	57,00	3939,88	**3203,16**
Wind	12,40	57,00	3536,36	**3930,68**
Market price	12,34	69,94	**4221,30**	786,85

All times in seconds. Full and restricted search spaces correspond to the AutoML framework scenario using the knowledge base extracted by *meta-llama/llama-4-scout-17b-16e-instruct*, and cover the complete pipeline (AutoML search, hyperparameter optimization and cross-validated evaluation). Bold values identify the configuration kept for result analysis for each use case. All runs were executed on an Intel Core i5-1135G7 with 16 GB RAM under Windows 11.

The baseline scenarios are between one and two orders of magnitude lighter than the complete pipeline, completing in under 70 seconds against roughly one hour: for solar, Baseline 2 is around 69 times faster than the full search space, and comparable ratios hold for the remaining use cases. This gap is the cost of searching over models, hyperparameters and engineered features rather than fixing them in advance, and it is the cost against which the accuracy gains reported in subsections 4.8.2, 4.8.3 and 4.8.4 must be weighed. It also reflects a different distribution of effort: the baselines re-optimize a fixed model structure at every fold, whereas the proposed approach concentrates its expenditure in a search phase whose outcome is reused thereafter.

As expected, reducing the search space improved computational time in most cases. The effect is by far the largest for electricity market prices, where the restriction reduced runtime from 4221,30 to 786,85 seconds, a reduction of more than 81%, although at the cost of accuracy, as discussed in sub-subsection 4.8.4.2. For solar forecasting the restriction reduced runtime by roughly 19% while simultaneously improving accuracy, making it the only case in which the trade-off was undeniably favorable. Wind forecasting was the exception, with the restricted configuration proving 11,2% slower than the full search space despite evaluating fewer features. Since that same restriction produced a substantial accuracy gain, the result indicates that runtime is governed by the composition of the surviving model search space, and therefore by the cost of hyperparameter optimization, rather than by the number of features retained.

It is also important to reiterate that the values presented in Table 4.14 are intended solely for illustrative purposes, providing a visual comparison of the computational effort required by the different scenarios rather than serving as absolute performance measurements. They were obtained from single runs on consumer-grade hardware and would vary with dataset size, implementation and the degree of parallelism employed.

It can nevertheless be concluded that the approach scales reasonably to new domains without prohibitive computational cost, the runtime remaining within approximately one hour per use case across three structurally different forecasting problems, while noting that this cost will grow with dataset size and with the breadth of the search space derived from the knowledge base.

4.8.7 Conclusions

To answer RQ2 directly, the proposed approach did not reach the accuracy of the external reference values in the two use cases for which such values were available, but achieved errors of the same order of magnitude in both, at a computational cost that remains modest in absolute terms, as detailed in subsections 4.8.5 and 4.8.6.

Regarding forecasting quality, solar and wind power forecasting were compared against the values reported in [71] for the same two sites. For solar, evaluated under comparable conditions (night periods removed), the proposed approach obtained a normalized MAE of 7,97% and RMSE of 11,67% against reference values of 6,11% and 9,73%, corresponding to deficits of roughly 30% and 20% respectively. For wind, the restricted search space produced a normalized MAE of 7,99% and RMSE of 10,98% against reference values of 6,22% and 8,94%, deficits of approximately 28% and 23%. It should be mentioned that these gaps were obtained without any domain-specific manual feature engineering or variable selection: all candidate models, hyperparameter ranges and features were derived automatically from the literature survey. The wind use case is nonetheless the most challenging from the perspective of knowledge extraction rather than of final accuracy, since only one of the three LLMs tested produced usable feature functions for this domain, and acceptable results were obtained only after restricting the search space to discard implementations that had introduced noise into the input space.

For electricity market price forecasting, no external reference was used, and the assessment was therefore made against the internal baselines evaluated under identical conditions. Here the proposed approach produced its largest relative gain of the three use cases, reducing the RMSE from 56,92 to 25,07 €/MWh and the MAE from 49,10 to 19,10 €/MWh with respect to Baseline 2, improvements of 55,96% and 61,10%. This result is not directly comparable with the solar and wind figures, since it measures improvement over a fixed-model baseline rather than a gap to an externally published benchmark, and it should be read as evidence that the extracted knowledge base added substantial predictive information in this domain, not as evidence of competitiveness against the state of the art in electricity price forecasting.

Regarding implementation time, the AutoML-based evaluation pipeline ran in between 3203 and 4221 seconds per use case, that is, between 0,89 and 1,17 hours, to which 10 to 25 minutes of knowledge base extraction must be added, giving an end-to-end cost of roughly 1,1 to 1,6 hours per use case. Taken in isolation, this exceeds the runtime of a traditional regression model applied directly to a pre-engineered feature set, and it is one to two orders of magnitude above the baseline scenarios, which completed in under 70 seconds. However, this comparison is incomplete if total development effort is considered. Traditional forecasting pipelines require substantial upfront investment in literature review, manual feature engineering and iterative model selection, processes that typically span days or weeks of expert effort and that are not reflected in any reported runtime. The proposed approach automates these stages, reducing human intervention to the provision of input literature and a final validation of results.

Overall, the evidence gathered supports a qualified answer to RQ2. The proposed LLM-assisted AutoML pipeline is not competitive with domain-tuned reference models in terms of forecasting accuracy, falling short by between 20% and 30% depending on the use case and metric. It is, however, competitive in terms of implementation effort, achieving errors within the same order of magnitude as hand-specified, domain-tuned models while replacing the manual knowledge acquisition stage entirely, and it clearly outperformed the untuned and Bayesian-optimized baselines in every use case evaluated.

The approach therefore trades a measurable loss in accuracy for the removal of the manual knowledge acquisition stage, and the results obtained suggest that a broader knowledge base and a more capable LLM could narrow the remaining gap.

Chapter 5

Conclusions & Future Work

5.1 Summary of findings

This dissertation investigated whether large language models (LLMs), combined with an AutoML framework, can automate the extraction of domain knowledge from scientific literature and use it to build competitive, low-effort forecasting pipelines for the energy sector. Two research questions guided this investigation, and the evidence gathered allows both to be answered.

Regarding RQ1, Section 4.7 showed that LLMs can extract structured, ready-to-use feature and model definitions from unstructured research documents, without requiring architectural changes between forecasting domains (solar, wind, and market price). However, this capability is not uniform, and it is markedly weaker for code than for text. The integrity assessment of Section 4.7.2 showed that, even among the artefacts whose description was judged faithful to the source paper, a large share of the corresponding implementations were rated inaccurate: 78,4%, 65,6% and 47,4% of feature-extraction functions for *llama-3.3-70b-versatile*, *meta-llama/llama-4-scout-17b-16e-instruct* and *openai/gpt-oss-120b* respectively, against 45,7%, 42,2% and 11,3% for forecasting models. This indicates that the summarization capability of current LLMs is more mature than their ability to translate mathematical or paper-specific logic into dataset-correct implementations, particularly when the target dataset differs in structure or variable naming from the one used in the original paper, and that the reliability of this translation depends heavily on the specific model employed. These figures were produced by a single LLM acting as judge, without a human-annotated reference set, and are therefore indicative rather than definitive.

Regarding RQ2, the proposed approach did not outperform the external reference values in the two use cases for which such values were available, though the size of the gap varied across domains. Solar and wind power forecasting fell short of the values reported in [71] for the same sites by roughly 20% to 30% in normalized MAE and RMSE, with wind proving the most challenging use case (normalized MAE of 7,99% against a reference value of 6,22%); this outcome is attributable primarily to feature-extraction quality rather than to a limitation of the pipeline architecture itself, since only a limited portion of the literature could be exploited for this domain due to inaccuracies in the LLM-generated implementations. For electricity market price forecasting,

evaluated against the internal baselines rather than an external benchmark, the approach achieved its largest relative gain, improving the Baseline 2 RMSE by 55,96% and the MAE by 61,10%. In terms of implementation effort, the full pipeline required between 0,89 and 1,17 hours of AutoML search and cross-validated evaluation per use case, plus 10 to 25 minutes of knowledge base extraction, giving roughly 1,1 to 1,6 hours end-to-end. While this exceeds the runtime of a single traditional model fit by one to two orders of magnitude, it replaces what is typically days to weeks of manual literature review and feature engineering.

Taken together, these results support a qualified affirmative answer to RQ1 and a qualified answer to RQ2: LLMs can meaningfully reduce, though not eliminate, the manual effort involved in building domain-informed forecasting pipelines, and the resulting models, while consistently better than the untuned and Bayesian-optimized baselines, remain short of domain-tuned reference models in forecasting accuracy.

5.2 Limitations

Regarding the limitations related with this dissertation work, it was found that the implementation reliability was a problem. As stated in Section 4.7.2, a substantial share of LLM-generated feature implementations, particularly for customized, paper-specific functions, were rated inaccurate by the "judge" model, and a faithful description offered limited assurance of a correct implementation for two of the three LLMs tested. This means the pipeline, in its current form, still requires a validation step, whether manual or automated, before extracted knowledge can be trusted at scale.

For the AutoML framework, the AutoML search was only executed once on the earliest 80% of each dataset and the resulting pipeline reused across the subsequent evaluation folds. This mirrors a deployment protocol in which an architecture search is performed once on historical data and the resulting pipeline periodically refit, and it was necessary given the cost of running the search per fold, but it means the selected pipeline structure, though not any fitted parameters, is held fixed throughout the evaluation. Also, the choice of TPOT constrained the AutoML stage to `scikit-learn` compatible pipelines, meaning that a portion of the models extracted from literature could not be evaluated regardless of extraction quality. This is an architectural choice rather than a fundamental limitation of the proposed approach.

The integrity assessment in Section 4.7.2 relied on a single "judge" model (*gemma4-31b*), with no human-annotated reference set against which its ratings could be calibrated. Since this evaluation is itself LLM-based rather than deterministic, the reported figures are not independent of the evaluator, and a different judge could plausibly rate the same outputs differently.

The literature coverage is also a limitation, since the knowledge base for each use case was built from approximately 15 papers, a sample sufficient to demonstrate feasibility but modest for saturating the space of feature engineering techniques reported in the literature. This was particularly limiting for the market price use case, where several ideas extracted from source papers could not be implemented due to raw variables not being available in the dataset used for testing.

Finally, the computational cost of the approach (detailed in Section 4.8.6), is dominated by hyperparameter optimization rather than by feature count, as evidenced by the wind case, where a restricted search space containing fewer features proved 11,2% slower than the full one. Runtime will therefore scale primarily with the breadth of the model search space derived from the knowledge base and with dataset size, and the absence of an intermediate filtering step means that low-quality extracted models consume search budget without contributing to accuracy. The reported times were also obtained from single runs on consumer-grade hardware and are indicative rather than absolute.

5.3 Future work

Based on the identified limitations, several directions can be explored in future work to further improve the results achieved in this dissertation. As proposed in Section 4.7.2, the judge could be incorporated as a dedicated LLM-agent node, and its ratings used as an intermediate filtering step between extraction and AutoML, discarding low-confidence implementations before they propagate into the forecasting pipeline. This would directly target the implementation-accuracy gap identified as the pipeline's principal current weakness. Also, for validating the judge's responses, it could be interesting to add a second, independent model as a supervisor that re-evaluates a sample of the judge's ratings, and calibrating the judge against a small set of manually annotated artefacts, would convert the exploratory integrity assessment reported here into a measurement that can be relied upon quantitatively.

As for the prompting strategies, some changes could be made to recover functions that are currently discarded outright due to syntax or leakage issues. To do this, a re-prompting or self-correction strategy, where the LLM is given the opportunity to revise a feature implementation after an execution or validation failure, could prove useful.

To expand the ML models considered, it can also be pondered to test frameworks beyond TPOT, particularly ones supporting non-scikit-learn architectures such as gradient-boosted libraries with native categorical support or deep learning models, would clarify whether the accuracy gap observed for wind power forecasting is attributable to feature quality alone, or also to the constrained model search space. Also, it is crucial to repeat the AutoML search at intervals along the evaluation window, rather than once at the start, would quantify how much of the reported performance depends on the pipeline selected in the initial search, and would allow the approach to adapt to regime changes such as those observed in the market price dataset.

As stated earlier in the previous section, expanding the number of source documents per use case would allow a larger share of literature-derived ideas to be tested against datasets containing the required raw variables, and would provide a more robust basis for the "most-referenced" model and "improves-forecasting" feature selection heuristics used in the restricted search space scenarios.

Implementation accuracy varied substantially across the three LLMs tested, with *openai/gpt-oss-120b* consistently outperforming the other two on code correctness despite extracting fewer

features overall. Future work could also explore ensembling extraction across multiple LLMs, or systematically benchmarking additional models, to identify the most reliable configuration for this task.

References

[1] Nikolaos Koutantos, Maria Fotopoulou, and Dimitrios Rakopoulos. "Automated Machine Learning for Optimized Load Forecasting and Economic Impact in the Greek Wholesale Energy Market". en. In: *Applied Sciences* 14.21 (Oct. 2024), p. 9766. ISSN: 2076-3417. DOI: 10.3390/app14219766. URL: https://www.mdpi.com/2076-3417/14/21/9766.

[2] Kasim Zor, Oguzhan Timur, and Ahmet Teke. "A state-of-the-art review of artificial intelligence techniques for short-term electric load forecasting". en. In: *2017 6th International Youth Conference on Energy (IYCE)*. Budapest, Hungary: IEEE, June 2017, pp. 1–7. ISBN: 978-1-5090-6409-0. DOI: 10.1109/IYCE.2017.8003734. URL: http://ieeexplore.ieee.org/document/8003734/.

[3] Paul Ghelasi and Florian Ziel. "From day-ahead to mid and long-term horizons with econometric electricity price forecasting models". In: *Renewable and Sustainable Energy Reviews* 217 (July 2025), p. 115684. ISSN: 13640321. DOI: 10.1016/j.rser.2025.115684. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364032125003570.

[4] Zhijin Wang et al. "A comprehensive review of deep learning for solar nowcasting: Enhancing accuracy, reliability, and interpretability". en. In: *Applied Energy* 407 (Mar. 2026), p. 127378. ISSN: 03062619. DOI: 10.1016/j.apenergy.2026.127378. URL: https://linkinghub.elsevier.com/retrieve/pii/S0306261926000309.

[5] Ikrame Arrassi et al. "A Comprehensive Review of AI-based Wind Power Forecasting Over Multiple Time Horizons". en. In: *Process Integration and Optimization for Sustainability* (Mar. 2026). ISSN: 2509-4238, 2509-4246. DOI: 10.1007/s41660-026-00730-z. URL: https://link.springer.com/10.1007/s41660-026-00730-z.

[6] Yuying Xie et al. "An overview of deterministic and probabilistic forecasting methods of wind energy". en. In: *iScience* 26.1 (Jan. 2023), p. 105804. ISSN: 25890042. DOI: 10.1016/j.isci.2022.105804. URL: https://linkinghub.elsevier.com/retrieve/pii/S2589004222020776.

[7] Hosna Khajeh and Hannu Laaksonen. "Applications of Probabilistic Forecasting in Smart Grids: A Review". en. In: *Applied Sciences* 12.4 (Feb. 2022), p. 1823. ISSN: 2076-3417. DOI: 10.3390/app12041823. URL: https://www.mdpi.com/2076-3417/12/4/1823.

[8] Md Meftahul Ferdaus et al. "Foundation models for clean energy forecasting: A comprehensive review". en. In: *Renewable and Sustainable Energy Reviews* 226 (Jan. 2026), p. 116452. ISSN: 13640321. DOI: 10.1016/j.rser.2025.116452. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364032125011256.

[9] Caixia Liu et al. "A fractional grey reservoir computing prediction model and its application in clean energy forecasting". en. In: *Energy* 342 (Jan. 2026), p. 139662. ISSN: 03605442. DOI: 10.1016/j.energy.2025.139662. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360544225053046.

[10] Jie Yan et al. "The economic value of wind power forecasting: a data-driven method and its application in various scenarios". en. In: *Energy* 340 (Dec. 2025), p. 139151. ISSN: 03605442. DOI: 10.1016/j.energy.2025.139151. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360544225047930.

[11] Aristeidis Mystakidis et al. "Energy Forecasting: A Comprehensive Review of Techniques and Technologies". en. In: *Energies* 17.7 (Mar. 2024), p. 1662. ISSN: 1996-1073. DOI: 10.3390/en17071662. URL: https://www.mdpi.com/1996-1073/17/7/1662.

[12] Ignacio Rojas, Gonzalo Joya, and Andreu Catala, eds. *Advances in Computational Intelligence: 18th International Work-Conference on Artificial Neural Networks, IWANN 2025, A Coruña, Spain, June 16–18, 2025, Proceedings, Part I*. en. Vol. 16008. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2026. ISBN: 978-3-032-02724-5 978-3-032-02725-2. DOI: 10.1007/978-3-032-02725-2. URL: https://link.springer.com/10.1007/978-3-032-02725-2.

[13] Frédéric Mirindi and Derrick Mirindi. "SEF-Net: A Hybrid Deep Learning Architecture for Multi-step Forecasting in Sustainable Energy Markets". en. In: *Advances in Computational Intelligence*. Ed. by Ignacio Rojas, Gonzalo Joya, and Andreu Catala. Vol. 16009. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2026, pp. 219–231. ISBN: 978-3-032-02727-6 978-3-032-02728-3. DOI: 10.1007/978-3-032-02728-3_18. URL: https://link.springer.com/10.1007/978-3-032-02728-3_18.

[14] Ping Song, Adegboyega Ojo, and Edward Curry. "Trustworthy requirements for foundation models—A comprehensive survey and roadmap". en. In: *Engineering Applications of Artificial Intelligence* 163 (Jan. 2026), p. 113111. ISSN: 09521976. DOI: 10.1016/j.engappai.2025.113111. URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197625031422.

[15] Tao Hong et al. "Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond". en. In: *International Journal of Forecasting* 32.3 (July 2016), pp. 896–913. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2016.02.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0169207016000133.

[16] Tarek AlSkaif et al. "A systematic analysis of meteorological variables for PV output power estimation". en. In: *Renewable Energy* 153 (June 2020), pp. 12–22. ISSN: 09601481. DOI: 10.1016/j.renene.2020.01.150. URL: https://linkinghub.elsevier.com/retrieve/pii/S0960148120301725.

[17] Kun Li et al. "The significance of calendar effects in the electricity market". en. In: *Applied Energy* 235 (Feb. 2019), pp. 487–494. ISSN: 03062619. DOI: 10.1016/j.apenergy.2018.10.124. URL: https://linkinghub.elsevier.com/retrieve/pii/S0306261918316933.

[18] Nikos Tsalikidis et al. "Energy load forecasting: one-step ahead hybrid model utilizing ensembling". en. In: *Computing* 106.1 (Jan. 2024), pp. 241–273. ISSN: 0010-485X, 1436-5057. DOI: 10.1007/s00607-023-01217-2. URL: https://link.springer.com/10.1007/s00607-023-01217-2.

[19] Farrukh Hafeez et al. “Accurate Solar Radiation Forecasting Using Spectral Feature Engineering and Bayesian Optimization”. en. In: *Eng* 7.2 (Feb. 2026), p. 77. ISSN: 2673-4117. DOI: 10.3390/eng7020077. URL: https://www.mdpi.com/2673-4117/7/2/77.

[20] Daniel P. Jeong, Zachary C. Lipton, and Pradeep Ravikumar. *LLM-Select: Feature Selection with Large Language Models*. en. arXiv:2407.02694 [cs]. Apr. 2025. DOI: 10.48550/arXiv.2407.02694. URL: http://arxiv.org/abs/2407.02694.

[21] Muhammad Uzair, Radwa El Shawi, and Stefania Tomasiello. “Context-Aware AutoML for Accurate Wheat Disease Detection”. In: *Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference co-located with the EDBT/ICDT 2025 Joint Conference, Barcelona, Spain, March 25, 2025*. Ed. by Matthias Boehm and Khuzaima Daudjee. Vol. 3946. CEUR Workshop Proceedings. CEUR-WS.org, 2025. URL: https://ceur-ws.org/Vol-3946/DARLI-AP-2.pdf.

[22] Tymoteusz Miller et al. “AI in Context: Harnessing Domain Knowledge for Smarter Machine Learning”. en. In: *Applied Sciences* 14.24 (Dec. 2024), p. 11612. ISSN: 2076-3417. DOI: 10.3390/app142411612. URL: https://www.mdpi.com/2076-3417/14/24/11612.

[23] Zeyu Wang et al. “Principles, research status, and prospects of feature engineering for data-driven building energy prediction: A comprehensive review”. en. In: *Journal of Building Engineering* 58 (Oct. 2022), p. 105028. ISSN: 23527102. DOI: 10.1016/j.jobe.2022.105028. URL: https://linkinghub.elsevier.com/retrieve/pii/S2352710222010385.

[24] Xun Zhao, Zheng Grace Ma, and Bo Nørregaard Jørgensen. “An End-to-End Data and Machine Learning Pipeline for Energy Forecasting: A Systematic Approach Integrating MLOps and Domain Expertise”. en. In: *Information* 16.9 (Sept. 2025), p. 805. ISSN: 2078-2489. DOI: 10.3390/info16090805. URL: https://www.mdpi.com/2078-2489/16/9/805.

[25] Farrukh Hafeez et al. “Accurate Solar Radiation Forecasting Using Spectral Feature Engineering and Bayesian Optimization”. en. In: *Eng* 7.2 (Feb. 2026), p. 77. ISSN: 2673-4117. DOI: 10.3390/eng7020077. URL: https://www.mdpi.com/2673-4117/7/2/77.

[26] Anh Truong et al. “Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools”. en. In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. arXiv:1908.05557 [cs.LG]. Nov. 2019, pp. 1471–1479. DOI: 10.1109/ICTAI.2019.00209. URL: http://arxiv.org/abs/1908.05557.

[27] Siyi Liu, Chen Gao, and Yong Li. “AgentHPO: Large Language Model Agent for Hyper-Parameter Optimization”. In: *Conference on Parsimony and Learning*. Ed. by Beidi Chen et al. Vol. 280. Proceedings of Machine Learning Research. PMLR, 24–27 Mar 2025, pp. 1146–1169. URL: https://proceedings.mlr.press/v280/liu25c.html.

[28] Konstantinos Filippou et al. “Structure Learning and Hyperparameter Optimization Using an Automated Machine Learning (AutoML) Pipeline”. en. In: *Information* 14.4 (Apr. 2023), p. 232. ISSN: 2078-2489. DOI: 10.3390/info14040232. URL: https://www.mdpi.com/2078-2489/14/4/232.

[29] Christian O'Leary, Conor Lynch, and Farshad Ghassemi Toosi. "A Comparative Analysis of Automated Machine Learning Libraries for Electricity Price Forecasting". en. In: *Applied Computer Systems* 29.2 (Dec. 2024), pp. 43–52. ISSN: 2255-8691. DOI: 10.2478/acss-2024-0020. URL: https://www.sciendo.com/article/10.2478/acss-2024-0020.

[30] Chaobo Zhang et al. "Automated machine learning-based building energy load prediction method". en. In: *Journal of Building Engineering* 80 (Dec. 2023), p. 108071. ISSN: 23527102. DOI: 10.1016/j.jobe.2023.108071. URL: https://linkinghub.elsevier.com/retrieve/pii/S2352710223022519.

[31] Can Wang et al. "Automated Machine Learning for Short-term Electric Load Forecasting". en. In: *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. Xiamen, China: IEEE, Dec. 2019, pp. 314–321. ISBN: 978-1-7281-2485-8. DOI: 10.1109/SSCI44817.2019.9002839. URL: https://ieeexplore.ieee.org/document/9002839/.

[32] Ranzhen Ren et al. "A consistent framework for decomposition-based time series forecasting". en. In: *Computers and Electrical Engineering* 129 (Jan. 2026), p. 110773. ISSN: 00457906. DOI: 10.1016/j.compeleceng.2025.110773. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045790625007165.

[33] Zhe Zhang et al. "Short-term time series prediction model based on TFA and IVMD-EMD Quadratic decomposition: a case study on power time series data". en. In: *Electric Power Systems Research* 254 (May 2026), p. 112610. ISSN: 03787796. DOI: 10.1016/j.epsr.2025.112610. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378779625011976.

[34] Mahshid Khatiriolyaee, Li Yang, and Richard W. Pazzi. "A comprehensive review of traffic prediction: From traditional machine learning to AutoML". en. In: *Neurocomputing* 666 (Feb. 2026), p. 132148. ISSN: 09252312. DOI: 10.1016/j.neucom.2025.132148. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231225028206.

[35] Trang T Le, Weixuan Fu, and Jason H Moore. "Scaling tree-based automated machine learning to biomedical big data with a feature set selector". en. In: *Bioinformatics* 36.1 (Jan. 2020). Ed. by Janet Kelso, pp. 250–256. ISSN: 1367-4803, 1367-4811. DOI: 10.1093/bioinformatics/btz470. URL: https://academic.oup.com/bioinformatics/article/36/1/250/5511404.

[36] John Dagdelen et al. "Structured information extraction from scientific text with large language models". en. In: *Nature Communications* 15.1 (Feb. 2024), p. 1418. ISSN: 2041-1723. DOI: 10.1038/s41467-024-45563-x. URL: https://www.nature.com/articles/s41467-024-45563-x.

[37] Max Dreger, Kourosh Malek, and Michael Eikerling. "Large language models for knowledge graph extraction from tables in materials science". en. In: *Digital Discovery* 4.5 (2025), pp. 1221–1231. ISSN: 2635-098X. DOI: 10.1039/D4DD00362D. URL: https://xlink.rsc.org/?DOI=D4DD00362D.

[38] Gabriel Antonesi et al. "A systematic review of transformers and large language models in the energy sector: towards agentic digital twins". en. In: *Applied Energy* 401 (Dec. 2025), p. 126670. ISSN: 03062619. DOI: 10.1016/j.apenergy.2025.126670. URL: https://linkinghub.elsevier.com/retrieve/pii/S030626192501400X.

[39] Lei Huang et al. "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions". en. In: *ACM Transactions on Information Systems* 43.2 (Mar. 2025), pp. 1–55. ISSN: 1046-8188, 1558-2868. DOI: 10.1145/3703155. URL: https://dl.acm.org/doi/10.1145/3703155.

[40] Hong-Jun Yoon et al. "Scaling open-weight large language models for hydropower regulatory information extraction: A systematic analysis". en. In: *Environmental Modelling & Software* 203 (Aug. 2026), p. 107030. ISSN: 13648152. DOI: 10.1016/j.envsoft.2026.107030. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364815226001775.

[41] Jinglue Xu et al. *Large Language Models Synergize with Automated Machine Learning*. en. arXiv:2405.03727 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2405.03727. URL: http://arxiv.org/abs/2405.03727.

[42] Eunyong Ha et al. "AI-based nanotoxicity data extraction and prediction of nanotoxicity". en. In: *Computational and Structural Biotechnology Journal* 29 (2025), pp. 138–148. ISSN: 20010370. DOI: 10.1016/j.csbj.2025.03.052. URL: https://www.sciencedirect.com/science/article/pii/S2001037025001175.

[43] Juyong Jiang et al. *A Survey on Large Language Models for Code Generation*. arXiv:2406.00515 [cs]. Nov. 2024. DOI: 10.1145/3747588. URL: http://arxiv.org/abs/2406.00515.

[44] Huanting Wang et al. *AI Agentic Programming: A Survey of Techniques, Challenges, and Opportunities*. en. Version Number: 2. 2025. DOI: 10.48550/ARXIV.2508.11126. URL: https://arxiv.org/abs/2508.11126.

[45] Noah Hollmann, Samuel Müller, and Frank Hutter. *Large Language Models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering*. 2023. arXiv: 2305.03403 [cs.AI]. URL: https://arxiv.org/abs/2305.03403.

[46] Siyuan Guo et al. *DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning*. 2024. arXiv: 2402.17453 [cs.LG]. URL: https://arxiv.org/abs/2402.17453.

[47] Sirui Hong et al. *Data Interpreter: An LLM Agent For Data Science*. 2024. arXiv: 2402.18679 [cs.AI]. URL: https://arxiv.org/abs/2402.18679.

[48] Jun Shern Chan et al. *MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering*. en. arXiv:2410.07095 [cs.CL]. Feb. 2025. DOI: 10.48550/arXiv.2410.07095. URL: http://arxiv.org/abs/2410.07095.

[49] Jaehyun Nam et al. *MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement*. en. arXiv:2506.15692 [cs.LG]. Aug. 2025. DOI: 10.48550/arXiv.2506.15692. URL: http://arxiv.org/abs/2506.15692.

[50] Xiangru Tang et al. *ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code*. en. arXiv:2311.09835 [cs.CL]. Aug. 2024. DOI: 10.48550/arXiv.2311.09835. URL: http://arxiv.org/abs/2311.09835.

[51] GLM-4 5 Team et al. *GLM-4.5: Agentic, Reasoning, and Coding (ARC) Foundation Models*. en. arXiv:2508.06471 [cs.CL]. Aug. 2025. DOI: 10.48550/arXiv.2508.06471. URL: http://arxiv.org/abs/2508.06471.

[52] LangChain. *LangChain Documentation*. 2026. URL: https://python.langchain.com/.

[53] Jiale Liu and Nanzhe Wang. *AutoSurrogate: An LLM-Driven Multi-Agent Framework for Autonomous Construction of Deep Learning Surrogate Models in Subsurface Flow*. 2026. arXiv: 2604.11945 [cs.LG]. URL: https://arxiv.org/abs/2604.11945.

[54] Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. *AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML*. en. arXiv:2410.02958 [cs]. June 2025. DOI: 10.48550/arXiv.2410.02958. URL: http://arxiv.org/abs/2410.02958.

[55] Yun-Da Tsai et al. *AutoML-GPT: Large Language Model for AutoML*. en. arXiv:2309.01125 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2309.01125. URL: http://arxiv.org/abs/2309.01125.

[56] Penghang Liu et al. *TS-Agent: A Time Series Reasoning Agent with Iterative Statistical Insight Gathering*. en. arXiv:2510.07432 [cs]. Oct. 2025. DOI: 10.48550/arXiv.2510.07432. URL: http://arxiv.org/abs/2510.07432.

[57] Daqin Luo et al. "AutoM^3 L: An Automated Multimodal Machine Learning Framework with Large Language Models". en. In: *Proceedings of the 32nd ACM International Conference on Multimedia*. Melbourne VIC Australia: ACM, Oct. 2024, pp. 8586–8594. ISBN: 979-8-4007-0686-8. DOI: 10.1145/3664647.3680665. URL: https://dl.acm.org/doi/10.1145/3664647.3680665.

[58] Xixun Lin et al. *LLM-based Agents Suffer from Hallucinations: A Survey of Taxonomy, Methods, and Directions*. 2025. arXiv: 2509.18970 [cs.AI]. URL: https://arxiv.org/abs/2509.18970.

[59] Costin Bădică et al. *Contemporary Agent Technology: LLM-Driven Advancements vs Classic Multi-Agent Systems*. 2025. arXiv: 2509.02515 [cs.MA]. URL: https://arxiv.org/abs/2509.02515.

[60] Zeeshan Rasheeda et al. *LLM-Based Multi-Agent Systems for Code Generation: A Multi-Vocal Literature Review*. 2026. arXiv: 2604.16321 [cs.SE]. URL: https://arxiv.org/abs/2604.16321.

[61] Çağatay Demiralp et al. *Making LLMs Work for Enterprise Data Tasks*. 2024. arXiv: 2407.20256 [cs.DB]. URL: https://arxiv.org/abs/2407.20256.

[62] Tangjie Wu and Qiang Ling. "STELLM: Spatio-temporal enhanced pre-trained large language model for wind speed forecasting". en. In: *Applied Energy* 375 (Dec. 2024), p. 124034. ISSN: 03062619. DOI: 10.1016/j.apenergy.2024.124034. URL: https://linkinghub.elsevier.com/retrieve/pii/S030626192401417X.

[63] LangGraph. *LangGraph Documentation*. 2026. URL: https://docs.langchain.com/oss/python/langgraph/overview.

[64] LangGraph. *LangGraph Context Overview*. 2026. URL: https://docs.langchain.com/oss/python/concepts/context.

[65] Scikit-learn. *Scikit-learn documentation*. https://scikit-learn.org/stable/about.html.

[66] TPOT. *TPOT documentation*. https://epistasislab.github.io/tpot/latest/Tutorial/2_Search_Spaces/.

[67] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automated Machine Learning: Methods, Systems, Challenges*. en. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-05317-8 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5. URL: http://link.springer.com/10.1007/978-3-030-05318-5.

[68] *HistGradientBoostingRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html.

[69] *Sotavento Wind Farm*. URL: https://www.sotaventogalicia.com/en/sotavento-galicia-s-a/.

[70] Scikit-learn. *Scikit-learn documentation for Supervised Learning*. https://scikit-learn.org/stable/supervised_learning.html.

[71] Jose R. Andrade and Ricardo J. Bessa. "Improving Renewable Energy Forecasting With a Grid of Numerical Weather Predictions". en. In: *IEEE Trans. Sustain. Energy* 8.4 (Oct. 2017), pp. 1571–1580. ISSN: 1949-3029, 1949-3037. DOI: 10.1109/TSTE.2017.2694340. URL: http://ieeexplore.ieee.org/document/7903735/.

Appendix A

Code execution and validation

1. Feature Execution Loop

```
local_var = {"df": df}

if state.functions['functions'][i]['complete'] == False:
    print(f"The function '{state.functions['functions'][i]['name']}' is marked as incomplete. Skipping execution.")
    state.functions['functions'][i]['errors'] = "Function marked as incomplete by LLM, thus not executed."
    state.functions['functions'][i]['compiles'] = False
    continue

try:
    exec(state.functions['functions'][i]['python_implementation'], globals(), local_var)
except Exception as e:
    print(f"Failed to execute feature function '{state.functions['functions'][i]['name']}': {type(e).__name__}: {str(e)}\n")
    state.functions['functions'][i]['errors'] = f"Execution error: {type(e).__name__}: {str(e)}"
    state.functions['functions'][i]['compiles'] = False
    continue

# --- Baseline features ---
df['hour'] = df[datetime].dt.hour
df['dayofweek'] = df[datetime].dt.dayofweek
df['month'] = df[datetime].dt.month
og_columns = df.columns.tolist()

# --- Recuperar funcao do escopo local ---
try:
    aux = local_var[state.functions['functions'][i]['name']]
except Exception as e:
    print(f"The function '{state.functions['functions'][i]['name']}' was not found in local scope: {e}")
    state.functions['functions'][i]['errors'] = f"Function not in local scope: {str(e)}"
    state.functions['functions'][i]['compiles'] = False
    continue

# --- Executar a funcao ---
try:
```

```
    df = aux(df)
    state.functions['functions'][i]['compiles'] = True
except Exception as e:
    print(f"The feature '{state.functions['functions'][i]['name']}' failed to execute: {type(e).__name__}: {str(e)}\n")
    state.functions['functions'][i]['compiles'] = False
    state.functions['functions'][i]['errors'] = f"Execution failed: {type(e).__name__}: {str(e)}"
    continue

# --- Validacao do tipo de retorno ---
if type(df) == tuple:
    print(f"The function '{state.functions['functions'][i]['name']}' returned a tuple instead of a dataframe. Skipping execution.")
    continue

if (type(df) == bool):
    print("The dataframe returned is a boolean value, which is not valid...")
    continue

if type(df) != pd.DataFrame:
    print(f"The function '{state.functions['functions'][i]['name']}' did not return a valid dataframe. Skipping execution.")
    state.functions['functions'][i]['errors'] = "Function did not return a valid dataframe."
    continue

df = df.dropna(subset=[target_column])
```

2. Validation & Horizon Evaluation

```
def validate_feature_to_horizon(df, features, target_column, horizon, start_date, n_forecasts=20):
    df_aux = df.copy()
    if horizon.lower() == "intra-day":
        offset = 1
    elif horizon.lower() == "day-ahead":
        offset = 24
    else:
        raise ValueError("Unknown horizon")
    datetime_col = df_aux.columns[0]
    df_aux[datetime_col] = pd.to_datetime(df_aux[datetime_col]).dt.tz_localize(None)
    df_aux = df_aux.sort_values(datetime_col)
    feature = features[0]
    if feature not in df_aux.columns:
        return {"horizon": horizon, "prediction": None, "applicable": False,
                "reason": f"Feature '{feature}' not found"}
    if not np.issubdtype(df_aux[feature].dtype, np.number):
        return {"horizon": horizon, "prediction": None, "applicable": False,
                "reason": f"Feature '{feature}' is non-numeric"}
    df_aux["hour"] = df_aux[datetime_col].dt.hour
    df_aux["dayofweek"] = df_aux[datetime_col].dt.dayofweek
    df_aux["month"] = df_aux[datetime_col].dt.month
    df_aux = df_aux.dropna(subset=[target_column])
    start_date = pd.to_datetime(start_date).tz_localize(None)
    if start_date not in df_aux[datetime_col].values:
```

```
        return {"horizon": horizon, "prediction": None, "applicable": False,
                "reason": "start_date not found in dataset"}
    start_idx = df_aux.index[df_aux[datetime_col] == start_date][0]
    baseline_errors = []
    feature_errors = []
    for i in range(n_forecasts):
        forecast_time = start_date + pd.Timedelta(hours=i)
        target_time = forecast_time + pd.Timedelta(hours=offset)
        if target_time not in df_aux[datetime_col].values:
            break
        train_data = df_aux[df_aux[datetime_col] <= forecast_time]
        if len(train_data) < 50:
            continue
        pred_row = df_aux[df_aux[datetime_col] == target_time]
        if pred_row.empty:
            continue
        # Baseline model
        Xb_train = train_data[["hour", "dayofweek", "month"]].to_numpy(dtype=float)
        y_train = train_data[target_column].to_numpy(dtype=float)
        base_model = ensemble.HistGradientBoostingRegressor(random_state=42)
        base_model.fit(Xb_train, y_train)
        Xb_pred = pred_row[["hour", "dayofweek", "month"]].to_numpy(dtype=float)
        pred_baseline = base_model.predict(Xb_pred)[0]
        # Feature model
        X_train = train_data[["hour", "dayofweek", "month", feature]].to_numpy(dtype=float)
        model = ensemble.HistGradientBoostingRegressor(random_state=42)
        model.fit(X_train, y_train)
        X_pred = pred_row[["hour", "dayofweek", "month", feature]].to_numpy(dtype=float)
        pred_feature = model.predict(X_pred)[0]
        actual_value = pred_row[target_column].values[0]
        error_baseline = abs(actual_value - pred_baseline)
        error_feature = abs(actual_value - pred_feature)
        baseline_errors.append(error_baseline)
        feature_errors.append(error_feature)
        print(
            f"Forecast {i+1}: "
            f"target={target_time} | "
            f"baseline={pred_baseline:.3f} | "
            f"feature={pred_feature:.3f} | "
            f"actual={actual_value:.3f}"
        )
    if len(baseline_errors) == 0:
        return {"horizon": horizon, "prediction": None, "applicable": False,
                "reason": "No valid forecasts generated"}
    baseline_mae = np.mean(baseline_errors)
    feature_mae = np.mean(feature_errors)
    improvement = baseline_mae - feature_mae
    applicable = improvement > 0
    print("\nRESULTS")
    print("Baseline MAE:", baseline_mae)
    print("Feature MAE:", feature_mae)
    return {
        "horizon": horizon,
        "baseline_mae": baseline_mae,
        "feature_mae": feature_mae,
        "improvement": improvement,
        "applicable": applicable,
```

```
        "n_forecasts": len(baseline_errors)
    }

% --- Modelo baseline (RMSE) ---
X_old = df_old[base_features].to_numpy(dtype=float)
y_old = df_old[target_column].to_numpy(dtype=float)
split_idx = int(len(X_old) * 0.8)
X_train_old, X_test_old = X_old[:split_idx], X_old[split_idx:]
y_train_old, y_test_old = y_old[:split_idx], y_old[split_idx:]
reg_old = ensemble.HistGradientBoostingRegressor(random_state=42)
reg_old.fit(X_train_old, y_train_old)
y_pred_old = reg_old.predict(X_test_old)
mse_old = np.sqrt(mean_squared_error(y_test_old, y_pred_old))

% --- Modelo com a nova feature (RMSE) ---
X = df[feature_cols].to_numpy(dtype=float)
y = df[target_column].to_numpy(dtype=float)
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]
reg = ensemble.HistGradientBoostingRegressor(random_state=42)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
mse_new = np.sqrt(mean_squared_error(y_test, y_pred))

% --- Comparacao de melhoria ---
if mse_new < mse_old:
    state.functions['functions'][i]["improves_forecasting"] = True
else:
    state.functions['functions'][i]["improves_forecasting"] = False
```

Appendix B

Used Prompts for the LLM

B.1 Feature-Function Extraction Prompt

Task description

```
List the feature functions used in the article that can improve predicition models
and a brief description of each.

Make sure to identify the case of use for each function, indicating if it's for
market price, solar, wind or load forecasting

Also, for each function return python code in order to implement it
```

Forecasting horizon

```
ALWAYS reply with one of the following forecasting horizons:
 - very-short term:  from a few minutes to up to 24 hours ahead
 - short-term:  24 hours or more ahead
 - medium-term:  1 week or more ahead
```

Function type and classification rules

```
For function_type, output exactly one of the following values:  'raw_function',
'feature_function' or raw_calendar_function.  No other values are allowed.

Definitions and decision rules:
Use raw_function for direct, original input variables taken from the data source
with no transformation or feature engineering applied (e.g., load(d), hour,
wind_speed).  Example of raw functions are those that are directly derived from the
timestamp without modification.  For raw calendar functions, use the function_type
'raw_calendar_function'

Use feature_function for any function derived from one or more raw functions
through feature engineering, including transformations, aggregations, encodings,
lags, rolling windows, seasonality terms, normalizations, or interaction terms.

Classification rule:
- If the function is a direct measurement or basic field, classify it as
raw_function.
- If the function is computed from other functions or modified in any way, classify
it as feature_function.
```

Classification examples

```
Examples:
- load(d) -> raw_function
- hour -> raw_function
- wind_speed -> raw_function
- rolling_mean(load, 24) -> feature_function
- sin(2 * pi* hour / 24) -> feature_function
- load(d-1) -> feature_function
```

Output and exclusion rules

Output format

Output a JSON array with each function as a separate object that can be parsed as is.

Exclusions

Don't save functions that output multiple features. Don't include standard functions like 'Pearson Correlation Coefficient', 'Autocorrelation' or Standard Deviation

Data leakage

If global statistics like mean, variance, quantiles, or correlation, are presented, don't apply them in the **{LEAKAGE_COLUMNS}** columns, as they can lead to data leakage. If the article mentions them, make sure to apply them only to the {FREE_COLUMNS} columns.

Completeness and model rules

Make sure to verify if the implementation of the function is complete and can be executed without errors. If it isn't, return 'False' in the 'complete' field. If it is complete and executable, return 'True' in the 'complete' field
Don't return models

Case of use and naming

Return a case of use between wind, solar, load or market price, depending on the one that is more adequate considering the context given in the article.

Following the example below, make sure that 'name' doesn't have any spacing between the words of the string

Source and paper information

Make sure to include also the paper from where this function was extracted, which is stored in {state.paper_path.stem}

The field has_info_paper should only be 'TRUE' if the paper explicitly mentions the function and gives some information about it, such as a description of how it works, its formula, or its parameters. If the function is only implied or can be inferred from the paper but isn't explicitly mentioned or described, set has_info_paper to 'False'.

Parameter assumptions and lags

```
If the parameters needed to implement the function aren't explicitly mentioned in
the article, make reasonable assumptions based on the context and specify those
assumptions in the description field of the function.  For example, if a rolling
mean is mentioned but the window size isn't specified, you can assume a common
window size like 24 for hourly data and mention this assumption in the description.
If the function relies on lagged data, return True in the field for
'relies_on_lags'
```

Case-of-use and duplication rules

```
Each function must have exactly one case_of_use from:  ['wind', 'solar', 'load',
'market price'].

Don't save metrics such as variance or correlation as functions.

Default behavior:
- Assign the SINGLE most appropriate case_of_use based on the primary context in
the article.
- Do NOT duplicate functions if they are generally applicable but not explicitly
discussed for multiple domains.

Duplication rule (strict):
- ONLY duplicate a function if the article explicitly states or clearly
demonstrates that the same function is used for multiple case_of_use categories.
- If this condition is not clearly met, DO NOT duplicate.

If duplication is required:
- Create one entry per case_of_use
- Modify the function name to be unique using a suffix (e.g., feature_solar,
feature_wind)
- Ensure each duplicated entry is clearly justified by the article context

Strict constraint:
- Do NOT create duplicate functions based on assumption or general knowledge.
- Duplication without explicit evidence from the article is NOT allowed.
Under no circumstances should a function contain multiple case_of_use values.
```

Lag, lead and split rules

```
LAG RULE: Apply lags as specified or as reasonable defaults, for the intended
columns.
EXCEPTION: for columns in {LEAKAGE_COLUMNS}, the minimum lag is 24 hours (e.g.
shift(24)).  Never use shift(1) or shift(12) for these columns.

If the function iterates over a list of columns to apply lags, REMOVE any column
from that list that appears in {LEAKAGE_COLUMNS} and apply it separately with a
minimum lag of 24 (e.g.  shift(24)).  For example, if the function lags ['power',
'temperature'] with shift(2), 'power' must be moved to a separate loop with
shift(24), and only 'temperature' keeps shift(2).

For {FREE_COLUMNS}, lags of any size are allowed, including 1-hour lags (shift(1)).
Apply whatever lag size the paper specifies, or a reasonable default if
unspecified.

LEAD RULE: If negative shifts (leads) are used, apply them ONLY to the columns in
{FREE_COLUMNS}

SPLIT RULE: If a function uses both lags (shift positive) and leads (shift
negative), return them as two separate JSON objects.  Name them with suffixes
'_lag' and '_lead'.  Do not merge lag and lead logic into one function.
```

Required JSON output format

```
REQUIRED JSON OUTPUT FORMAT:
Make sure to return only the JSON, without any additional text or formatting
:
{"functions":  [
  {
    "name":  "<string>",
    "function_type":  "<string>",
    "short_description":  "<string>",
    "description":  "<string>",
    "python_implementation":  "<string with escaped Python code>",
    "horizon":  "<string>",
    "case_of_use":  "<string>",
    "complete":  "<boolean>",
    "has_info_paper":  "<boolean>",
    "source":  "{state.paper_path.stem},
    "relies_on_lags":  "<boolean>" "
  }
]}
```

B.2 Regression-Model Extraction Prompt

Task description

```
Based on the article provided, list the regression models that were used.  For each
one, give information about their horizon for prediction, time resolution and
hyperparameters.  If you can't find the information that was asked, return 'Could
not find'

Also, make sure to specify the case of use for each model, as well as the
prediction variables used, as specified in the paper

Make the description as concise as possible, and make sure to build code sufficient
to be used to fully implement the function in Python.

Only return a response according to the following structure:
Following the example below, make sure that 'name' doesn't have any spacing between
the words of the string
```

Completeness and compatibility

```
Also, check if the python implemmentation is compatible with scikit-learn.  If it
is, return True in sklearn_compatible, if it isn't, return False.

If the python implemmentation is only a draft or a placeholder, return 'False' in
complete, otherwise, if it is complete and executable, return 'True'.
```

Source and model naming

```
Make sure to include also the paper from where this model was extracted, which is
stored in {state.paper_path.stem}

The field 'name' must be the same as the name of the import in the python
implementation.  For example, if the model is a MPLRegressor, imported in the
python implementation as 'from sklearn.neural_network import MLPRegressor' and then
instantiated as 'model = MLPRegressor(hidden_layer_sizes=(100,),
activation='relu')', the name of the model should be 'MLPRegressor'
```

Search space

```
Also, in the 'search_space' field, if the python implementation includes imports
from sklearn, return a string in the format of the following examples:  If you
have:  from sklearn.neural_network import MLPRegressor (...), the field search
space must be:  'MLPRegressor:  sklearn.neural_network'.  Another example:  from
sklearn.ensemble import GradientBoostingRegressor must be:
'GradientBoostingRegressor:  sklearn.ensemble'.  If there are no imports from
sklearn in the python implementation, return 'N/A' in the search_space field
```

Hyperparameters

```
HYPERPARAMETERS RULE: If there are hyperparameters specified in the paper, give
them a value based on the information in the paper.  The returned values for the
'hyperparameters' field must be in the format :  'hyperparameter':  'value'.  An
EXAMPLE of this is shown below:

n_estimators:  500, learning_rate:  0.05, max_depth:  5, min_samples_split:  150,
min_samples_leaf:  20
```

Required JSON output format

```
REQUIRED JSON OUTPUT FORMAT:
{"models": [
  {
    "name": "<string>",
    "horizon": "<string>",
    "time_resolution": "<string>",
    "description": "<string>",
    "python_implementation": "<string with escaped Python code>",
    "hyperparameters": "<string>",
    "sklearn_compatible": "<boolean>",
    "case_of_use": "<string>",
    "prediction_variables": "<string>",
    "complete": "<boolean>",
    "search_space": "<string>"
    "source": "{state.paper_path.stem}"
  }
]}
```

B.3 Feature-Function Adaptation Prompt

Task description

```
Adapt the following Python function implementation to work with the dataframe
columns provided:

Current implementation:
{state.functions['functions'][i]['python_implementation']}

Dataframe columns:  {list(df.columns)}
```

Requirements

```
- The function must be named:  {state.functions['functions'][i]['name']}
- Use ONLY existing dataframe columns in calculations
- Return the modified dataframe with the new feature column added
- Output ONLY raw Python code (no markdown fences like "`python, no explanations)
- Ensure the code is directly executable with Python exec()
- For non-numerical features, adapt the code so that you return integer identifiers
instead of a string identifier.  For example, if you have a column with categories
like 'A', 'B', 'C', return 0, 1, 2 respectively
- For lag based functions, consider the horizon of the function.  For short-term
horizons, generate a 24-hour lag, for very-short term consider a 1-hour lag
- If you don't have enough information in the dataframe's columns to implement the
function, update {state.functions['functions'][i]['has_info_dataset']} to FALSE
- For lag based functions, make sure to don't consider future data.  For example,
if you are generating a lag feature for the target variable 'actual_price' with a
horizon of 24 hours, make sure to use only data from the past 24 hours and not from
the future.  This is crucial to avoid data leakage and ensure that the feature is
realistic and can be used in a real forecasting scenario.
```

Lag rule

```
LAG RULE: Apply lags as specified or as reasonable defaults, for the intended
columns EXCEPTION: for columns in {LEAKAGE_COLUMNS}, the minimum lag is 24 hours
(e.g.  shift(24)).  Never use shift(1) or shift(12) for these columns.

If the function iterates over a list of columns to apply lags, REMOVE any column
from that list that appears in {LEAKAGE_COLUMNS} and apply it separately with a
minimum lag of 24 (e.g.  shift(24)).  For example, if the function lags ['power',
'wind_speed'] with shift(2), 'power' must be moved to a separate loop with
shift(24), and only 'wind_speed' keeps shift(2).
```

Split rule

```
SPLIT RULE: If a function applies BOTH lags (past values, e.g.  shift(24)) AND
leads (future values, e.g.  shift(-24)), you MUST return them as two separate JSON
objects in the output array.  Do NOT combine them in one function.  Example:  if
the paper describes a function 'price_lag_lead' that uses price(t-24) and
price(t+24), return:
  1) name='price_lag' with only the lag logic (shift(+24)), relies_on_lags=True
  2) name='price_lead' with only the lead logic (shift(-24)), relies_on_lags=False
In the description of each, note:  'Split from original function price_lag_lead:
lag part' / 'lead part'.

If a function only has lags OR only has leads, do NOT split it.
```

B.4 Cross-Domain Applicability Prompt

Task description

```
Check if the following function is applicable to other cases of use (wind, solar,
load, market price), based on its description and implementation.  The function is:

Name:  {fn['name']}
Description:  {fn['description']}
Implementation:  {fn['python_implementation']}
Current case of use:  {fn['case_of_use']}

If it is applicable to other cases of use, return a JSON list where each entry has
the same structure as below but with the new case_of_use filled in.  Only change
the 'case_of_use' field.  Keep all other fields identical to the original.
```

Required JSON structure

```
Required JSON structure for each new entry:
{
  "name":  "{fn['name']}",
  "function_type":  "{fn.get('function_type', '')}",
  "short_description":  "{fn.get('short_description', '')}",
  "description":  "{fn.get('description', '')}",
  "python_implementation":  "{fn.get('python_implementation', '')}",
  "horizon":  "{fn.get('horizon', '')}",
  "case_of_use":  "<NEW CASE OF USE: one of wind, solar, load, market price>",
  "complete":  {fn.get('complete', True)},
  "compiles":  false,
  "errors":  "",
  "improves_forecasting":  false,
  "improves_intra_day":  false,
  "improves_day_ahead":  false,
  "source":  "{fn.get('source', '')}",
  "has_info_paper":  {fn.get('has_info_paper', False)},
  "has_info_dataset":  {fn.get('has_info_dataset', True)},
  "relies_on_lags":  {fn.get('relies_on_lags', False)}
}

If it is NOT applicable to any other case of use, return exactly:  false
Return only the JSON list or false, no additional text or formatting.
```

Appendix C

Considered research documents for each use case

C.1 Solar Power Forecasting

Table C.1: Literature Matrix for Solar Power Forecasting

Research Documents
Ait Abdelmoula, Ibtihal, et al. "A photovoltaic power prediction approach enhanced by feature engineering and stacked machine learning model." *Energy Reports* 8 (2022): 1288-1300.
Abuella, Mohamed, and Badrul Chowdhury. "Solar power forecasting using artificial neural networks." *2015 North American Power Symposium (NAPS)*. IEEE, 2015.
Al-Ali, Elham M., et al. "Solar energy production forecasting based on a hybrid CNN-LSTM-transformer model." *Mathematics* 11.3 (2023): 676.
Al-Dahidi, Sameer, et al. "Forecasting solar photovoltaic power production: A comprehensive review and innovative data-driven modeling framework." *Energies* 17.16 (2024): 4145.
Andrade, José R., and Ricardo J. Bessa. "Improving renewable energy forecasting with a grid of numerical weather predictions." *IEEE Transactions on Sustainable Energy* 8.4 (2017): 1571-1580.
Elsaiti, Meftah, and Adel Merabet. "Solar power forecasting using deep learning techniques." *IEEE Access* 10 (2022): 31692-31698.
Elsheikh, Ammar H., et al. "Utilization of LSTM neural network for water production forecasting of a stepped solar still with a corrugated absorber plate." *Process Safety and Environmental Protection* 148 (2021): 273-282.
Ledmaoui, Younes, et al. "Forecasting solar energy production: A comparative study of machine learning algorithms." *Energy Reports* 10 (2023): 1004-1012.
Munawar, Usman, and Zhanle Wang. "A framework of using machine learning approaches for short-term solar power forecasting." *Journal of Electrical Engineering & Technology* 15.2 (2020): 561-569.
Pedro, Hugo TC, and Carlos FM Coimbra. "Assessment of forecasting techniques for solar power production with no exogenous inputs." *Solar Energy* 86.7 (2012): 2017-2028.
"Public Holidays in Energy Forecasting." Workshop on Energy Analytics, Cairns, Australia. Conference presentation, GEFCom 2017.

Table C.1 – continued

Research Documents
Shedbalkar, Kaustubha, and D. S. More. "Refining solar energy forecasting with optimization and feature engineering." *Journal of Integrated Science and Technology* 13.3 (2025): 1054-1054.
Subramanian, E., et al. "Solar power prediction using machine learning." *arXiv preprint arXiv:2303.07875* (2023).
Voyant, Cyril, et al. "Machine learning methods for solar radiation forecasting: A review." *Renewable Energy* 105 (2017): 569-582.
Wang, Zhijin, et al. "A comprehensive review of deep learning for solar nowcasting: Enhancing accuracy, reliability, and interpretability." *Applied Energy* 407 (2026): 127378.
Yang, Dazhi, et al. "Analyzing big time series data in solar engineering using features and PCA." *Solar Energy* 153 (2017): 317-328.

C.2 Electricity Market Prices Forecasting

Table C.2: Literature Matrix for Electricity Market Price Forecasting

Research documents
Amjady, Nima. "Day-ahead price forecasting of electricity markets by a new fuzzy neural network." *IEEE Transactions on power systems* 21.2 (2006): 887-896.
Amjady, Nima, and Meisam Hemmati. "Day-ahead price forecasting of electricity markets by a hybrid intelligent system." *European Transactions on Electrical Power* 19.1 (2009): 89-102.
Anbazhagan, S., and Narayanan Kumarappan. "Day-ahead deregulated electricity market price forecasting using recurrent neural network." *IEEE Systems Journal* 7.4 (2012): 866-872.
Barta, Gergo, et al. "GEFCom 2014—Probabilistic Electricity Price Forecasting." Global Energy Forecasting Competition 2014, 2014.
Bento, Pedro MR, et al. "Short-term price forecasting in the Iberian electricity market: Sensitivity assessment of the exogenous variables influence." 2022 IEEE.
Bessa, Ricardo J., and Manuel A. Matos. "The role of an aggregator agent for EV in the electricity market." *7th Mediterranean Conference and Exhibition on Power Generation, Transmission, Distribution and Energy Conversion (MedPower 2010)*. IET, 2010.
de Simon-Martin, Miguel, et al. "Electricity Spot Prices Forecasting for MIBEL by using Deep Learning: a comparison between NAR, NARX and LSTM networks." *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*. IEEE, 2020.
Faia, Ricardo, et al. "Portfolio optimization for electricity market participation with particle swarm." *2015 26th international workshop on Database and Expert Systems Applications (DEXA)*. IEEE, 2015.
Faia, Ricardo, et al. "Multi-objective portfolio optimization of electricity markets participation." *2018 Power Systems Computation Conference (PSCC)*. IEEE, 2018.
Ghelasi, Paul, and Florian Ziel. "From day-ahead to mid and long-term horizons with econometric electricity price forecasting models." *Renewable and Sustainable Energy Reviews* 217 (2025): 115684.
Gough, Matthew, et al. "Bidding strategies for virtual power plants in the Iberian electricity market." *2021 IEEE International Conference on Environment and Electrical Engineering and 2021 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*. IEEE, 2021.
Gonçalves, Carla, et al. "Explanatory and causal analysis of the mibel electricity market spot price." *2019 IEEE Milan PowerTech*. IEEE, 2019.
Imani, Mahmood Hosseini, et al. "Forecasting electricity price in different time horizons: an application to the Italian electricity market." *IEEE Transactions on Industry Applications* 57.6 (2021): 5726-5736.
Pao, Hsiao-Tien. "Forecasting electricity market pricing using artificial neural networks." *Energy conversion and management* 48.3 (2007): 907-912.
Shafie-khah, M., M. Parsa Moghaddam, and M. K. Sheikh-El-Eslami. "Price forecasting of day-ahead electricity markets using a hybrid forecast method." *Energy Conversion and Management* 52.5 (2011): 2165-2169.
Teixeira, Inês M., Ana Paula Barroso, and Teresa Marques. "Extreme learning machine for short and mid-term electricity spot prices forecasting." *2021 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2021.

C.3 Wind Power Forecasting

Table C.3: Literature Matrix for Wind Power Forecasting

Research documents
Andrade, José R., and Ricardo J. Bessa. "Improving renewable energy forecasting with a grid of numerical weather predictions." *IEEE Transactions on Sustainable Energy* 8.4 (2017): 1571-1580.
Arrassi, Ikrame, et al. "A comprehensive review of ai-based wind power forecasting over multiple time horizons." *Process Integration and Optimization for Sustainability* (2026): 1-23.
Augustyn, Aleksandra, and Jacek Kamiński. "A review of methods applied for wind power generation forecasting." *Polityka Energetyczna* 21.2 (2018): 139-150.
Barbosa de Alencar, David, et al. "Different models for forecasting wind power generation: Case study." *Energies* 10.12 (2017): 1976.
De Caro, Fabrizio, et al. "DAFT-E: Feature-based multivariate and multi-step-ahead wind power forecasting." *IEEE Transactions on sustainable energy* 13.2 (2021): 1199-1209.
Guan, Shijie, et al. "Ultra-short-term wind power prediction method combining financial technology feature engineering and XGBoost algorithm." *Heliyon* 9.6 (2023).
Habib, Md Ahasan, and M. J. Hossain. "Revolutionizing wind power Prediction—The future of energy forecasting with advanced deep learning and strategic feature engineering." *Energies* 17.5 (2024): 1215.
Jeon, Jooyoung, and Fotios Petropoulos. "Temporal Hierarchical Wind Power Density Forecasting." ISEA, 23 June 2017, Cairns, Australia. Conference presentation.
Li, Yongguo, Jiayi Pan, and Jiangdong Wang. "A hybrid framework for offshore wind power forecasting: integrating CNN-BiGRU-XGBoost with advanced feature engineering and analysis." *Energies* 18.19 (2025): 5153.
Lu, Peng, et al. "Short-term wind power forecasting based on meteorological feature extraction and optimization strategy." *Renewable Energy* 184 (2022): 642-661.
Kisvari, Adam, Zi Lin, and Xiaolei Liu. "Wind power forecasting—A data-driven method along with gated recurrent neural network." *Renewable Energy* 163 (2021): 1895-1909.
Medina Maçaira, P.; Monteiro Cyrillo, Y.; Cyrino Oliveira, F.L.; Castro Souza, R. "Including Wind Power Generation in Brazil's Long-Term Optimization Model for Energy Planning." *Energies* 2019, 12, 826.
Pinson, Pierre, and Georges Kariniotakis. "On-line assessment of prediction risk for wind power production forecasts." *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology* 7.2 (2004): 1119-1322.
Singh, Upma, et al. "A machine learning-based gradient boosting regression approach for wind power production forecasting: A step towards smart grid environments." *Energies* 14.16 (2021): 5196.

Appendix D

Pipeline of all system performed tasks

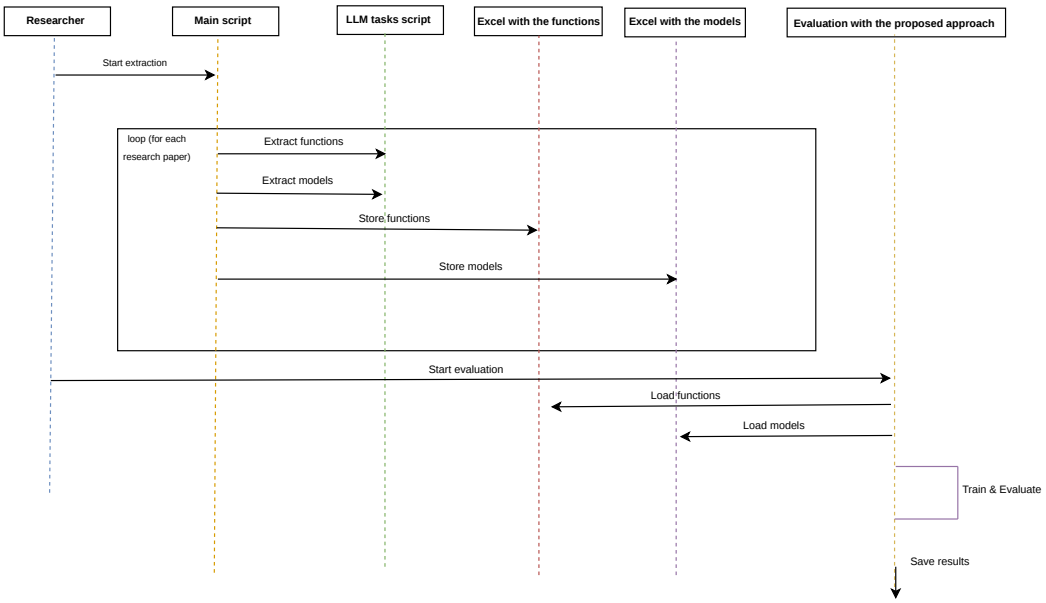


Figure D.1: Diagram that represents the overall workflow from information extraction and validation until the final evaluation with AutoML.