

From ML2 to ML2+: Integrating Time Series Forecasting in Model-Driven Engineering of Smart IoT Applications

Zahra Mardani Korani^{1,4}^a, Moharram Challenger²^b, Armin Moin³^c, João Carlos Ferreira⁴^d,
Alberto Rodrigues da Silva⁵^e, Gonçalo Vitorino Jesus¹^f, Elsa Lourenço Alves¹^g
and Ricardo Correia⁶

¹LNEC, Portugal

²Department of Computer Science, University of Antwerp & Flanders Make, Belgium

³Department of Computer Science, University of Colorado, Colorado Springs, CO, U.S.A.

⁴ISCTE, Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, 1649-026 Lisbon, Portugal

⁵INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

⁶BioGHP, Portugal

{zmardani, gjesus, ealves}@lnec.pt, moharram.challenger@uantwerpen.be, amoin@uccs.edu,


Keywords: Model-Driven Engineering, Machine Learning, IoT, Time Series Forecasting.


Abstract: Time-series forecasting is essential for anomaly detection, predictive maintenance, and real-time optimization in IoT environments, where sensor data is sequential. However, most model-driven engineering (MDE) frameworks lack specialized mechanisms to capture temporal dependencies, restricting the creation of intelligent and adaptive IoT systems. IoT inherently involves sequential data, yet most frameworks do not support time-series forecasting, essential for real-world systems. This paper presents ML2+, an enhanced version of the ML-Quadrat framework that integrates software engineering (SE) with machine learning (ML) in model-driven engineering. ML2+ allows users to define models, things, and messages for time-series forecasting. We evaluated ML2+ through two IoT use cases, focusing on development time, performance metrics, and lines of code (LOC). Results show that ML2+ maintains prediction accuracy similar to manual coding while significantly reducing development time by automating tedious tasks for developers. By automating feature engineering, model training, and evaluation for time-series data, ML2+ streamlines forecasting, improving scalability. ML2+ supports various forecasting models, including deep learning, statistical, and hybrid models. It offers preprocessing capabilities such as handling missing data, creating lagged features, and detecting data seasonality. The tool automatically generates code for time-series forecasting, making it easier for developers to train and deploy ML models without coding.


1 INTRODUCTION


Model-driven engineering for the Internet of Things (MDE4IoT) has gained significant attention in recent years due to its ability to improve the efficiency, predictability, and maintainability of IoT system development. Using models as the main artifacts for soft-


ware design and implementation, MDE4IoT offers a high-level abstract representation of the system, out of which the source code and other artifacts can automatically be generated (Da Silva, 2015; Alulema et al., 2020). To further enhance its capabilities, data analytics and machine learning (ML) techniques have been incorporated into the development process (Moin et al., 2022b; Moin et al., 2022a; Kirchhof et al., 2022). This integration has enabled the creation of more intelligent and adaptive IoT systems. However, despite these advances, existing MDE4IoT frameworks often lack native, out-of-the-box support for time series forecasting, an essential component for applications that rely on sequential or temporal data.


^a <https://orcid.org/0000-0001-9144-7964>


^b <https://orcid.org/0000-0002-5436-6070>

^c <https://orcid.org/0000-0002-8484-7836>

^d <https://orcid.org/0000-0002-6662-0806>

^e <https://orcid.org/0000-0002-7900-9846>

^f <https://orcid.org/0000-0002-8431-3877>

^g <https://orcid.org/0000-0003-0937-7237>

Time-series forecasting focuses on predicting future values derived from historical observations indexed in time. As most IoT applications produce time series data (for example, evolving sensor readings), the ability to accurately model and forecast these trends is critical for real-time monitoring, predictive maintenance, anomaly detection, and trend analysis (Adi et al., 2020; Cruz-Nájera et al., 2022; Mardani Korani et al., 2023). Although previous research has explored integrating ML into MDE4IoT workflows, a notable gap remains in the specialized support for advanced time-series forecasting models. Previous works, such as GreyCat (Hartmann et al., 2019), have attempted to integrate machine learning (ML) into domain models for the Internet of Things (IoT). However, they lack dedicated frameworks to address the complexities inherent in sequential data, such as capturing temporal dependencies and managing seasonality. Bridging this gap presents significant challenges, necessitating a systematic approach to modeling temporal relationships, automating feature engineering for sequential data, and seamlessly integrating these processes into Model-Driven Engineering (MDE) workflows. This paper introduces ML2+, an enhanced extension of the ML-Quadrat (ML2) framework (Moin et al., 2022a) that directly tackles these challenges by embedding comprehensive time-series forecasting capabilities into MDE4IoT. Building upon ML2, an open-source Model-Driven Software Engineering (MDSE) tool that integrates ML with IoT development, ML2+ augments the framework with robust functionalities for time-series forecasting. It leverages diverse models, including deep learning architectures like LSTM, statistical models such as ARIMA and Prophet, and hybrid approaches such as XGBoost, providing a versatile toolkit for developers to effectively handle temporal data within IoT applications. It automatically generates Python code for data preprocessing (e.g., handling missing values, creating lagged features, detecting seasonality), model training, and evaluation, thus eliminating the need for developers to have specialized expertise in time-series analysis. This automation optimizes development workflows, improves scalability, and enables developers to seamlessly integrate advanced forecasting features into their IoT solutions. By reducing manual coding, ML2+ allows developers to focus on higher-level analytical tasks rather than low-level implementation details. The framework supports configurations for multivariate analysis, seasonality detection, and stationarity checks, ensuring that generated models are well-tuned to the data characteristics. As an open-source, community-driven platform, ML2+ fosters collaboration and knowledge exchange, ultimately bridging a

key research gap in MDE4IoT and supporting sophisticated time-series analysis for a wide range of IoT systems. This paper is structured as follows: Section 2 examines the latest developments in the field, while Section 3.1.4 presents our proposed solution. Section 4 describes the validation and evaluation. Finally, Section 5 summarizes the study and outlines future work.

2 LITERATURE REVIEW

We briefly reviewed several related works in the area of MDE for the IoT (MDE4IoT). **ML-Quadrat (ML2)** (Moin et al., 2018; Moin et al., 2020; Moin, 2021; Moin et al., 2022a) integrated supervised, unsupervised, and semi-supervised learning, improving the development of IoT applications. However, it did not explicitly address time-series forecasting, requiring developers to manually manage complex temporal dependencies. Hartmann et al. introduced **GreyCat** (Hartmann et al., 2019), which integrated ML techniques into domain models for IoT. Although GreyCat enabled predictions within domain models, it did not emphasize the specialized models and preprocessing steps necessary for time-series forecasting. Similarly, **MoSIoT** (Meliá et al., 2021) incorporated learning features into the IoT scenario models, but lacked dedicated support for advanced time-series forecasting tasks. The **Stratum platform** (Bhattacharjee et al., 2019) focused on analytics management in IoT contexts, and **Hartsell et al.** (Hartsell et al., 2019) proposed model-based design techniques for cyber-physical systems. Although these approaches showcased integrated analytics or active learning methods, they did not address the unique requirements of time series forecasting, such as handling seasonality, performing stationarity checks, and performing lag feature engineering. Beyond MDE4IoT-specific tools, a range of ML frameworks (e.g., **TensorFlow** (Abadi et al., 2015), **Keras** (Chollet et al., 2015)) and workflow designers (e.g., **KNIME** (Berthold et al., 2009), **Rapid-Miner** (Mierswa et al., 2006)) offered high-level APIs and flexible interfaces for ML. However, these did not align fully with the model-driven engineering paradigm and lacked direct mechanisms to integrate time-series forecasting models into MDE4IoT artifacts. **Interoperability standards** such as PMML, PFA, and ONNX (Bai et al., 2019) enhanced model exchange but did not seamlessly incorporate temporal modeling capabilities within domain-specific modeling languages (DSMLs) for IoT. In particular, while the existing version of ML-Quadrat provided a strong

foundation for general ML tasks, it did not fully integrate time-series forecasting. This gap remained a key barrier for developers who needed to leverage forecasting within an MDE4IoT ecosystem. Our research addressed this gap by enhancing ML-Quadrat with **ML2+**. By incorporating advanced time-series forecasting models (e.g., LSTM, ARIMA, Prophet, XG-Boost), automating data preprocessing, and streamlining model training, ML2+ enabled more effective and efficient development of IoT applications. Future work will validate ML2+ through extensive IoT case studies, focusing on improvements in development time, forecast accuracy, and overall scalability.

3 PROPOSED APPROACH

We enhanced the ML-Quadrat (ML2) (Moin et al., 2022a) framework to ML2+ by integrating time-series forecasting. This extension introduces time-series modeling constructs, automated preprocessing pipelines, and advanced forecasting methods within the Model-Driven Engineering for the Internet of Things (MDE4IoT) workflow, enabling effective handling of sequential data essential for IoT applications requiring temporal pattern analysis.

3.1 The Foundational Software Model (SM)

The foundational Software Model (SM) (Moin et al., 2022a) for IoT/CPS systems is formally defined as:

$$SM = (A, \Psi, B, C) \quad (1)$$

Where:

- **A: Annotations** for external libraries and protocols.
- **Ψ : Structural Elements** (e.g., Thing, Port, Message).
- **B: Behavioral Elements** (e.g., finite-state machines or statecharts).
- **C: Configurations** that manage settings, component instantiations, and connections.

Based on *SM*, we construct the *Smart Software Model (SSM)* by incorporating a *Domain Model (DM)* that specifies machine learning (ML) tasks in behavioral elements:

$$SSM = (A, \Psi, f_B(DM), C) \quad (2)$$

Here, DM stands for Domain Model, which encompasses the relevant ML artifacts (such as algorithms,

hyperparameters, and data schemas). The function $f_B(DM)$ indicates the **behavior modifications** to the finite-state machines of SM that arise from the incorporation of ML. In practical terms, these modifications can include transitions or states that depend on the predictions of the ML model (e.g., switching to a state *warning* if a forecast sensor value exceeds a threshold). This approach ensures that the system's run-time behavior is intelligently adapted based on the results of data-driven models (Moin et al., 2022a). To integrate time-series forecasting into the SSM, we refine $f_B(DM)$ to:

$$f'_B(DM_2, DM_{TS})$$

Where:

- DM_2 : Standard ML tasks (e.g., classification or regression).
- DM_{TS} : Time-series-specific constructs (e.g., ARIMA, LSTM, Prophet).

Thus, the SSM now becomes:

$$SSM = (A, \Psi, f'_B(DM_2, DM_{TS}), C) \quad (3)$$

3.1.1 Time-Series Domain Model (DM_{TS})

The time-series domain model, DM_{TS} , contains essential artifacts for forecasting tasks, such as model architectures, parameters, features, hyperparameters, and metadata. Notably, the SSM is initially **un-trained**, meaning DM_{TS} includes constructs for training but not the trained model itself. Our implementation uses two separate datasets to train these ML models, ensuring robust and accurate forecasting.

$$DM_{TS} = (\mathcal{V}_{TS}, P_{TS}, \Phi_{TS}, H_{TS}, I_{TS}) \quad (2) \quad (4)$$

Where:

- \mathcal{V}_{TS} : **Model Architecture** (e.g., ARIMA, LSTM) defining the structure of the forecasting approach.
- P_{TS} : **Model Parameters** (e.g., ARIMA (p, d, q) , LSTM units) detailing specific configuration values per architecture.
- Φ_{TS} : **Time-Series Features** derived from historical data (e.g., lagged observations, rolling windows).
- H_{TS} : **Hyperparameters** (e.g., learning rate, epochs) guiding the training process.
- I_{TS} : **Metadata** (e.g., data frequency, forecast horizon) providing contextual information for the time-series data.

By incorporating DM_{TS} into the SSM, we enable robust time-series forecasting within MDE4IoT workflows. The following sections detail how ML2+ automates preprocessing, training, and deployment for these models, minimizing manual effort while preserving flexibility for developers and data scientists.

3.1.2 Model Validation and Code Generation

The validated and complete SSM models are converted into executable source code:

$$V(SSM) \wedge C(SSM) \implies \Delta(SSM) = \text{full_source_code} \quad (5)$$

Where:

- $V(SSM)$: Validates the SSM.
- $C(SSM)$: Checks the completeness of the SSM.
- Δ : Automates the transformation, reducing manual effort.
- Supports *statsmodels*, *xgboost*, *prophet*, *pytorch*, and integrates with *scikit-learn*, *keras-tensorflow*, *weka*.

3.1.3 Configurations Management

Configurations manage component instantiations and connections (Moin et al., 2022a):

$$C_i = (A_{C_i}, \Theta, \Xi) \quad (6)$$

where:

- A_{C_i} : Annotations for component requirements.
- Θ : Instantiated components (e.g., data streams, preprocessing units, models).
- Ξ : Communication connectors for data flow.

By maintaining abstraction, ML2+ ensures that complex IoT architectures with forecasting capabilities are manageable and maintainable.

3.1.4 Proposed Architecture Design

This study improves the framework ML-Quadrat (ML2) (Moin et al., 2022a) by integrating time-series forecasting into Internet of Things (IoT) systems. Figure 1 illustrates the metamodel of our architecture, highlighting new time series forecasting components in yellow, along with their preprocessing steps in red in the Data Analytics component, assessment methods, and model parameters. Green-marked components indicate enhancements and expanded machine learning functionalities of ML-Quadrat (ML2). Using domain-specific modeling languages (DSML) and model-driven engineering (MDE), ML2+ manages system complexity and automates development tasks such as code generation and deployment (Mardani Korani et al., 2023).

3.1.5 Abstract Syntax of the DSML

The DSML for ML2+ is built on an Ecore metamodel within the MDE4IoT framework, tailored for time-series forecasting. It defines datasets with features, labels, and temporal attributes, and includes preprocessing configurations such as imputation and outlier removal. The language supports models such as ARIMA, SARIMA, LSTM, and hybrids with customizable parameters, and incorporates evaluation metrics that include RMSE, MAE, and MSE.

3.1.6 Concrete Syntax and Model Editors

The DSML's concrete syntax is designed for user-friendliness, facilitating interaction with time-series forecasting models within the MDE4IoT framework. Developed using Xtext grammar in Eclipse, it enables structured model definitions. A cloud-based web editor has replaced the previous ML2 editor, offering features such as syntax highlighting, autocompletion, and customizable templates to enhance user experience.

3.1.7 Semantics and Model-to-Code Transformations

The semantics of the DSML define the meaning and behavior of models created using the language. It includes model-to-code transformations that convert high-level model descriptions into executable Python and Java code. These transformations automatically generate deployable scripts such as `processing.py`, `training.py`, and `predict.py`, ensuring that the generated code aligns with the model logic. Xtend-based templates streamline this process, enabling easy deployment on IoT devices and edge platforms.

3.1.8 Supported Time-Series Forecasting Methods and Techniques

ML2+ provides a comprehensive framework for time-series forecasting, preprocessing, and evaluation. It supports datasets with features, labels, and temporal configurations, using parameters like `common_period_threshold` to align multivariate IoT datasets by specifying the maximum allowable missing values. Preprocessing capabilities include imputation, outlier removal, and transformations such as `lag_features`, `rolling_windows`, and `resampling`. Temporal data handling is configurable, with options for `forecasting_horizons`, `lag_settings`, `seasonality_detection`, and `stationarity_checks`. The framework supports deep learning models, including MLP, GRU, CNN, LSTM, RNN, TCN, and Transformers, as well as

statistical models like ARIMA, SARIMA, Holt-Winters, and ETS. Machine learning models such as SVR, RFR, GBM, and XGBoost, along with hybrid approaches like ARIMA_GARCH and Prophet, are also included. Hyperparameter tuning techniques, including `grid_search`, `random_search`, and `Bayesian_optimization`, ensure optimal model performance. Model evaluation employs metrics like RMSE, MAE, and MSE, with support for context-specific performance analysis in domains such as RiverFlow prediction and DataCenterManagement. Domain-specific use cases, including RiverFlow, ServerMonitoring, and IoTDataCenter, are defined for specialized applications. The framework integrates seamlessly with widely used libraries such as `statsmodels`, `xgboost`, `prophet`, `pytorch`, `scikit-learn`, `keras-tensorflow`, and `weka`, enabling efficient data processing and accurate predictions without extensive DSML or modeling expertise.

4 VALIDATION AND EVALUATION

4.1 River Flow Forecasting

A case study was implemented to validate the proposed river flow prediction framework using actual multivariate datasets. The system leverages the ML2+ framework and the Data Analytics and Machine Learning (DAML) components to predict river discharge and support flood risk management in an IoT environment. For example, to predict river flow for the next three days at the target station Açude Ponte de Coimbra (12G/01AE), daily river flow data are collected from multiple monitoring stations, including Albufeira da Aguieira (11H/01A), Albufeira da Raiva (12H/01A), Albufeira de Fronhas (12I/01A) and Açude Ponte de Coimbra (12G/01AE). The dataset spans from January 1, 1984, to November 18, 2024, consisting of daily average effluent flow measurements (in m^3/s), provided by the Portuguese National Water Resources Information System (SNIRH)(System, ; Jesus et al., 2025). Each station transmits its data daily to a central server, which is subsequently accessed by the DAML server. The workflow, orchestrated by the DAML components, automates the critical stages of the pipeline. The data preprocessing step, executed using `DA_Preprocess`, prepares the collected multivariate data by interpolating missing values, capped at 10 consecutive gaps, to ensure data continuity. The dataset is resampled to align with a common period threshold and con-

verted into a supervised learning format by generating lagged features that capture temporal dependencies across multiple stations, essential for accurate river flow forecasting. These preprocessing steps are implemented using the `Scikit-Learn` library as part of the software model. The preprocessed data are split into 80% for training and 20% for testing, ensuring the sequential nature of the multivariate time-series data is preserved. The training process is executed using `DA_Train`, which deploys a Multilayer Perceptron (MLP) model configured with 50 neurons, 50 epochs, a batch size of 16, L2 regularization with a value of 0.01, a dropout rate of 0.2, and the `relu` optimizer with Mean Squared Error (MSE) as the loss function. This configuration ensures efficient training while maintaining model accuracy and is implemented using the `Keras` library within the ML2+ framework. Following the training phase, predictions are generated for the next three days at the target station using `DA_Predict`, a component in the ML2+ framework that retrieves the latest input features, performs predictions, and stores the forecasted values in predefined properties. The predicted river flow values are compared against a predefined threshold of $150\text{m}^3/\text{s}$. If the predicted flow exceeds this threshold, a flood alert is triggered, allowing proactive decision-making to mitigate potential risks.

4.2 Energy Consumption Forecasting

A case study was conducted to evaluate the proposed energy consumption forecasting system, leveraging real-world data from the UCI Machine Learning Repository (UCI Machine Learning Repository,). Let us assume we want to predict the total energy consumption for the next three hours in a smart home setting. The dataset spans from December 2006 to November 2010 and provides second-level measurements of electric power consumption in kilowatts (kW). The data include total energy usage as the target variable and time-indexed features, such as appliance-specific consumption (e.g., dishwasher, refrigerator), and additional contextual factors like temperature when available. In this setup, data are transmitted to a central server every second, supporting real-time monitoring and prediction. However, to align with the forecasting goal of predicting total energy consumption over the next three hours, the second-level data are aggregated into hourly intervals during preprocessing. Hourly resampling reduces noise, highlights long-term trends, and ensures computational efficiency. The aggregated data are then accessed by the DAML server, which orchestrates the workflow to automate the critical stages of

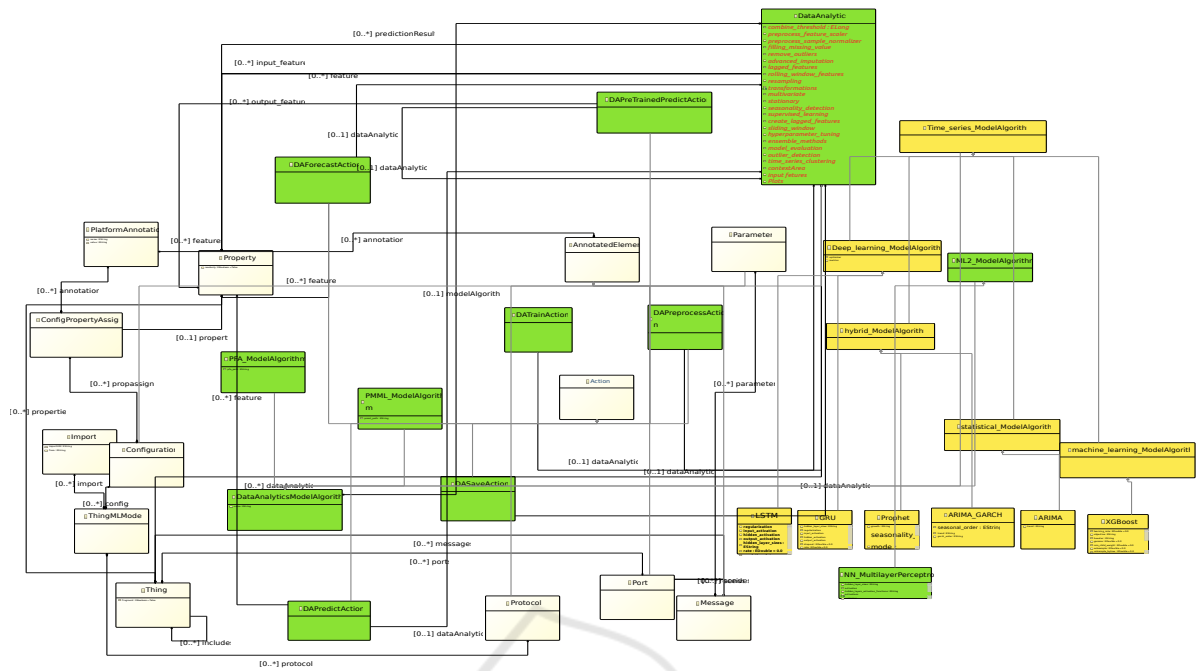


Figure 1: Meta-Model Diagram of the Enhanced ML2+ Framework, highlighting new components in yellow.

data preparation, model training, and forecasting. The workflow begins with `DA.Preprocess`, which automates preprocessing tasks, including resampling, interpolation of missing data, generation of lagged features to capture temporal dependencies, and scaling of variables to ensure consistency. During resampling, the dataset is transformed from second-level granularity to hourly intervals by taking the mean energy usage within each hour. Missing values were imputed using interpolation to ensure data continuity. Lagged versions of the use [kW] variable were generated for the last three hours ($t - 1, t - 2, t - 3$) to incorporate temporal dependencies. Features were normalized using Min-Max Scaling to bring values into the range $[0, 1]$, and outliers were removed using the Z-score method to improve model robustness. Stationarity of the time series was verified using the Augmented Dickey-Fuller (ADF) test, and non-stationary series were differenced as needed to stabilize the variance. The dataset was divided into 80% training data and 20% testing data, maintaining the temporal order of the time series to preserve sequence relationships. For model training, the `DA.Train` component was used to train two forecasting models: Holt-Winters (Triple Exponential Smoothing) and SARIMA. The Holt-Winters model was configured to capture the level, trend, and seasonality of energy usage, with additive components and a seasonal period of 24 (corresponding to daily hourly seasonality). The SARIMA model, which extends ARIMA by incorporating seasonal pat-

terns, was configured with the parameters $(1, 1, 1)$ for ARIMA and $(1, 1, 1, 24)$ for seasonal components. Seasonal decomposition was used to identify optimal seasonal parameters for the SARIMA model. Hyperparameter tuning was conducted to optimize model performance. A library such as Statsmodels was utilized within the ML2+ framework for model implementation and evaluation. Once the models were trained, the `DA.Predict` component was used to generate predictions for the next three hours. The DAML server retrieved the forecast input data, aggregated it into hourly intervals, fed them into the trained models, and stored the predicted energy usage in predefined properties. For proactive energy management, an alert threshold of 5 kW was defined. If the predicted total energy consumption exceeded this threshold, an alert was triggered, allowing real-time adjustments or interventions in the smart home environment.

4.3 Comparative Analysis and Metrics

The evaluation presented in this paper was conducted by a user with medium-level experience in Python coding and a good understanding of the writing of model instances. Forecast performance was assessed using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). **RMSE** penalizes larger errors, making it suitable for identifying significant deviations, while **MAE** provides a straightforward measure of the magnitude of the average error. For the

River Flow Forecasting use case, ML2+ achieved RMSE values of 50.75 for $t + 0$, 72.66 for $t + 1$, and 86.65 for $t + 2$, identical to those obtained through manual coding. This shows that automation using ML2+ does not compromise accuracy. Additionally, ML2+ reduced the development time from 14 to 11 hours, although the lines of code (LOC) increased from 80 to 210 due to the generation of a complete software model instance. In the **Energy Consumption Prediction** use case, RMSE values were 0.2301 for Holt-Winters and 0.1941 for SARIMA, with corresponding MAE values of 0.1882 and 0.1553. Using ML2+, the development time decreased from 11.5 hours to just 4 hours, while the LOC increased from 64 and 60 to 200 and 205 for the respective models. ML2+ automates the generation of a complete software model instance, encompassing all system elements such as Things, Ports, Messages, Properties, Statecharts, and Configurations. The user defines the model instance through a web-based editor and a template with predefined structures for Things, Ports, Messages, and workflows. This template-based approach simplifies the process by requiring the user to modify specific parameters and attributes, such as hyperparameters, model types, or message formats, rather than constructing the entire model from scratch. The built-in automation in ML2+ manages repetitive tasks such as preprocessing steps, model training, and evaluation workflows, allowing the user to focus on adapting the provided templates to their specific requirements. Consequently, ML2+ not only reduces development time but also minimizes manual effort, making it a valuable tool for users aiming to streamline the model development process.

5 CONCLUSION AND FUTURE WORK

This paper introduced ML2+, an enhanced framework that integrates comprehensive time-series forecasting functionalities into the MDE4IoT paradigm. By automating data preprocessing, feature engineering, model training, and evaluation, ML2+ addresses a crucial gap in existing MDE4IoT frameworks, allowing developers to handle sequential data without specialized time-series expertise. Evaluating river flow and energy consumption scenarios shows that ML2+ maintains predictive accuracy comparable to manual coding approaches while reducing development time. Automation of the framework and abstractions reduces the probability of human error, improves reliability, and simplifies maintenance. ML2+ thus bridges the gap between MDE and ML for time

series forecasting, positioning itself as a valuable tool for developing intelligent, scalable, and adaptable IoT applications. As an open-source, community-driven platform, ML2+ will evolve with user feedback and emerging technologies, such as advanced ML models, ensuring it remains a key enabler of intelligent IoT ecosystems. ML2+ has demonstrated its potential to automate time-series forecasting in AIoT contexts. Future improvements will focus on enhanced visualization capabilities that allow users to inspect preprocessing steps, monitor training progress, and interpret forecast results interactively. We will gather feedback from IoT developers, data scientists, and MDE practitioners to refine ML2+, ensuring that it aligns closely with user needs and workflows. Broader evaluations are planned across various domains, including smart cities, healthcare, and industrial IoT, validating ML2+ under various data conditions. Furthermore, we aim to integrate ML2+ with edge computing strategies, enabling efficient model updates on the fly without extensive code rewrites. This adaptability is vital for continuously evolving IoT environments, ensuring that the models remain accurate and relevant. Community involvement and open collaboration are also priorities. By fostering a user community that shares best practices, templates, and extensions, ML2+ can continuously evolve and remain at the forefront of MDE4IoT advancements.

ACKNOWLEDGMENTS

This work is supported by several projects, including Blockchain PT (PRR-RE-C05-i01.02: Agendas/Alianças Verdes para a Inovação Empresarial), CONNECT 22050 (Local Coastal Monitoring Service for Portugal), and ATTRACT-DIH (Digital Innovation Hub for Artificial Intelligence and High-Performance Computing, funded under the Digital European Programme, Grant 101083770).

REFERENCES

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Adi, E., Anwar, A., Baig, Z., and Zeadally, S. (2020). Machine learning and data analytics for the iot. *Neural Computing and Applications*, 32(20):16205–16233.
- Alulema, D., Criado, J., Iribarne, L., Fernández-García, A. J., and Ayala, R. (2020). A model-driven engineer-

- ing approach for the service integration of iot systems. *Cluster Computing*, 23(3):1937–1954.
- Bai, H., Breuel, T. M., et al. (2019). Onnx: Open neural network exchange. *GitHub Repository*. Available at: <https://onnx.ai/>.
- Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinel, T., Ohl, P., Sieb, C., Thiel, K., and Wiswedel, B. (2009). Knime: The konstanz information miner. In *Data Analysis, Machine Learning and Applications*, pages 319–326. Springer.
- Bhattacharjee, A., Barve, Y., Khare, S., Bao, S., Kang, Z., Gokhale, A., and Damiano, T. (2019). Stratum: A bigdata-as-a-service for lifecycle management of iot analytics applications. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1607–1612. IEEE.
- Bredereke, J., Morin, B., et al. (2013). Models@runtime to support dynamic adaptation. *Software and Systems Modeling*, 12:159–168.
- Chollet, F. et al. (2015). Keras.
- Cruz-Nájera, M. A., Treviño-Berrones, M. G., Ponce-Flores, M. P., Terán-Villanueva, J. D., Castán-Rocha, J. A., Ibarra-Martínez, S., Santiago, A., and Laria-Menchaca, J. (2022). Short time series forecasting: Recommended methods and techniques. *Symmetry*, 14(6):1231.
- Da Silva, A. R. (2015). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155.
- Guazzelli, A., Zeller, M., Lin, W.-C., and Williams, G. (2009). Pmml: An open standard for sharing models. *The R Journal*, 1(1):60–65.
- Harrand, N., Fleurey, F., Morin, B., and Husa, K. (2016). Thingml: A language and code generation framework for heterogeneous targets. *Proceedings of the ACM SIGPLAN International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 125–135.
- Hartmann, T., Moawad, A., Fouquet, F., and Le Traon, Y. (2019). The next evolution of mde: a seamless integration of machine learning into domain modeling. *Software & Systems Modeling*, 18(2):1285–1304.
- Hartsell, C., Mahadevan, N., Ramakrishna, S., Dubey, A., Bapty, T., Johnson, T., Koutsoukos, X., Sztipanovits, J., and Karsai, G. (2019). Model-based design for cps with learning-enabled components. In *Proceedings of the Workshop on Design Automation for CPS and IoT*, pages 1–9.
- Jesus, G., Mardani, Z., Alves, E., and Oliveira, A. (2025). Using deep learning for tejo river flow forecasting. *Submitted to Sensors*. Under review.
- Kirchhof, J. C., Kusmenko, E., Ritz, J., Rumpe, B., Moin, A., Badii, A., Günemann, S., and Challenger, M. (2022). Mde for machine learning-enabled software systems: a case study and comparison of montianna & ml-quadrat. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '22*, page 380–387, New York, NY, USA. Association for Computing Machinery.
- Mardani Korani, Z., Moin, A., Rodrigues da Silva, A., and Ferreira, J. C. (2023). Model-driven engineering techniques and tools for machine learning-enabled iot applications: A scoping review. *Sensors*, 23(3).
- Meliá, S., Nasabeh, S., Luján-Mora, S., and Cachero, C. (2021). Mosiot: Modeling and simulating iot healthcare-monitoring systems for people with disabilities. *International Journal of Environmental Research and Public Health*, 18(12):6357.
- Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). Rapidminer: An open source system for knowledge discovery in large data sets. In *Proceedings of the NIPS ML Open Source Software Workshop*.
- Minka, T. P., Winn, J., Guiver, J., and Knowles, D. (2018). Infer.net: A framework for running bayesian inference in graphical models. *Journal of Machine Learning Research*, 18:1–5.
- Moin, A. (2021). Data analytics and machine learning methods, techniques and tool for model-driven engineering of smart iot services. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 287–292.
- Moin, A., Challenger, M., Badii, A., and Günemann, S. (2022a). A model-driven approach to machine learning and software modeling for the iot. *Software and Systems Modeling*, 21(3):987–1014.
- Moin, A., Mituca, A., Challenger, M., Badii, A., and Günemann, S. (2022b). Ml-quadrat & driotdata: a model-driven engineering tool and a low-code platform for smart iot services. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, ICSE '22*, page 144–148, New York, NY, USA. Association for Computing Machinery.
- Moin, A., Rössler, S., and Günemann, S. (2018). Thingml+: Augmenting model-driven software engineering for the internet of things with machine learning. In *Proceedings of MODELS 2018 Workshops, Copenhagen, Denmark, October, 14, 2018*, volume 2245 of *CEUR Workshop Proceedings*, pages 521–523. CEUR-WS.org.
- Moin, A., Rössler, S., Sayih, M., and Günemann, S. (2020). From things' modeling language (thingml) to things' machine learning (thingml2). *MODELS '20*, New York, NY, USA. ACM.
- Morin, B., Barais, O., Fleurey, F., et al. (2016). Heads: A holistic approach for the development of distributed heterogeneous and adaptive systems. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 92–101. Springer.
- Open Data Group (2016). Portable format for analytics (pfa). Available at: <http://dmg.org/pfa/>.
- System, P. N. W. R. I. Snirh portal. <https://snirh.apambiente.pt/snirh/>. Accessed on [2024].
- UCI Machine Learning Repository. Individual household electric power consumption dataset. <https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption>. Accessed: [2024].