Self-Adaptive Service Selection for Machine Learning Continuous Delivery

Mostafa Hadadian Nejad Yousefi
Bernoulli Institute
University of Groningen
Groningen, The Netherlands
m.hadadian@rug.nl

Victoria Degeler
Informatics Institute
University of Amsterdam
Amsterdam, The Netherlands
v.o.degeler@uva.nl

Alexander Lazovik
Bernoulli Institute
University of Groningen
Groningen, The Netherlands
a.lazovik@rug.nl

***Abstract*—In the dynamic landscape of machine learning applications on streaming data, the constant evolution of models and input data complicates optimal model deployment. The static selection of a model risks suboptimal performance as data patterns evolve, while frequent redeployments increase operational costs. This paper proposes a self-adaptive system that autonomously selects interchangeable models for processing streaming data while balancing the tradeoff of performance and redeployment frequency. Inspired by the MAPE-K reference model, our approach utilizes an adaptive model selection control loop to continuously monitor model performance on production and experimental data. "what-if" environments are introduced to collect additional experimental data, simulating production-like scenarios. A selection algorithm that employs two distinct adaptation policies is introduced that strategically plans the selection of the most suitable module for upcoming data. Leveraging a learning-based method, we improve the efficiency of our system by recognizing the patterns of selection eliminating the need for further experimental data collection. Empirical evaluation on an energy forecasting use case spans over 16 years of data demonstrates a substantial reduction in errors up to 34% compared to the best static selection, affirming the proposed framework's effectiveness. Our findings reveal the potential to discontinue experimental "what-if" analyses with just 12% of historical data, which underlines the practicality of our adaptive strategy on a long-lasting task.**

***Index Terms*—Data Processing, Adaptive System, Service Selection, Machine Learning Development, Continuous Deployment.**

I. Introduction

Data scientists and machine learning (ML) developers are continuously developing new ML models to handle data processing tasks. However, the data used to train and evaluate models, which represent real-world conditions, is constantly changing. Therefore, developing high-quality ML solutions can be prolonged as it requires both extensive data collection and model development over numerous iterations.

Delaying deployment until a singular "best" model consistently outperforms others across all data points is inefficient and impractical for real-world use cases demanding an operational ML system. Organizations need deployable solutions as early as possible in the development cycle. This raises the key challenge of determining which model(s) should be deployed and when during ongoing development as new models are introduced and data evolves over time. This challenge intensifies in stream processing tasks where data are not immediately available, limiting the information accessible for informed real-time decision-making.

While the ideal solution would be to deploy the model with the lowest error that remains optimal forever, in practice the most accurate model can change depending on the input data. Additionally, changing the deployed model in a production environment comes with costs such as downtime and infrastructure overhead during updates. Therefore, the optimal strategy aims to balance minimizing error rates and disruptions from model changes.

The challenge of efficient model selection in machine learning and software engineering has prompted various attempts within these communities. In the machine learning realm, approaches like meta-learning and ensemble methods have been explored. Ensemble methods, while combining outputs from different models, may not consistently outperform individual models for all data inputs, and they often incur high computational costs due to training and concurrent execution of multiple models on all datasets [1]. Meta-learning, an alternative approach, aims to predict model performance based on dataset features but requires the entire dataset to be available for feature extraction, assuming the presence of relevant features correlated with model performance [2]. These methods face practical challenges, especially when dealing with a large number of datasets with diverse complexities in continuous machine learning development. Furthermore, they often overlook non-functional requirements, such as computational cost, in the decision-making process.

Meanwhile, in the realm of service computing techniques, the emphasis lies in the one-time selection of services during service composition for tasks with a focus on non-functional requirements. Despite the abundance of adaptive systems in the service community, their primary focus is not on service selection but rather on aspects such as self-healing, autoscaling, and service placement within the system [3].

The principle of "Everything as a module" (XaaM) [4] bridges the gap between machine learning and service-oriented development. XaaM views a module as a self-contained software component designed to take zero or more inputs, perform

This research has been sponsored by NWO C2D and TKI HTSM Ecida Project Grant No. 628011003.

a specific function, and provide zero or more outputs. In the context of data stream processing, we characterize modules as producers, processors, or consumers. A producer like a sensor driver generates output without requiring input. Conversely, A processor like a machine learning model processes input to produce output. Finally, a consumer like a database interface ingests input data for storage, consequently not generating a traditional output. As part of XaaM realization, this work addresses the problem of dynamically selecting the best-performing module for a given machine learning task at runtime in a stream processing context to maximize the module's performance while minimizing the frequency of module redeployments.

This paper introduces a self-adaptive system utilizing a control loop based on the widely-used MAPE-K reference model [5] to continuously monitor and select ML modules in streaming applications. Leveraging historical module performance, the system makes informed selections for production deployment and experimentation based on ongoing observations. Unlike ensemble models that incur higher costs by selecting multiple modules for production, our approach reduces costs by selecting a single module for production at each time, focusing on individual data points rather than average performance. To gain insights into non-production modules, we introduce "what-if" environments, simulating production-like scenarios with informative data samples. Rather than predicting module performance, which is challenging in streaming data, our approach focuses on understanding the pattern of optimal selection. From a service computing perspective, our work incorporates adaptive service composition, dynamically changing the ML module in production at runtime based on its performance. This work stands out in its novel integration of machine learning and service computing principles, addressing the identified problem with an effective approach.

In this paper, we present a selection algorithm equipped with two distinct adaptation policies, specifically crafted to respond to sudden shifts in system performance. Accompanying this, we introduce a learning-based method capable of detecting the inherent patterns that dictate how the selection algorithm reacts to incoming data. This method significantly lowers the computational overhead involved in running parallel modules evaluations in "what-if" scenarios, thus bolstering the efficiency of our adaptive framework.

We conducted experiments focused on forecasting hourly energy consumption demand over 16 years, implementing multiple machine learning modules within our system. Two benchmarks, random and optimal choice, were established. The optimal choice exhibited a remarkable 38% less error compared to the best static module, while the random choice performed 15% worse. Through extensive experimentation on this use case, we showcase how our adaptive selection algorithms progressively optimize the deployed pipeline. Our solution yields a significant reduction in errors, up to 34%, compared to the best static selection. These results land in close proximity to the optimal choice benchmark, thereby validating the efficacy of our proposed adaptation policy. Our experiments on the learning-based selection method illustrate its potential to stop "what-if" scenarios using just 1 to 2 years of data, depending on the chosen policy, further emphasizing the practicality of our adaptive strategy for long-running tasks.

The rest of this paper is organized as follows: Section II discusses related work. Section III outlines our proposed framework, provides a formal problem definition, introduces the adaptation policies and their respective implementation algorithms, and explains the details of the learning-based method. Section IV presents an experimental evaluation, focusing on an energy forecasting use case. Finally, Section V concludes the paper.

II. Related Work

This study resides at the convergence of machine learning methodologies and service computing paradigms, delving into both domains to establish its contextual framework.

A. Machine Learning

Our work lies at the intersection of machine learning methodologies, particularly in the domains of Meta-learning [6] and ensemble models [7]. The algorithm selection challenge, dating back to Rice in 1976 [8], recognizes the absence of a universally superior algorithm. Meta-learning prerequisites include diverse datasets, a range of algorithms, performance metrics, and suitable meta-features for characterizing datasets. Notable approaches, such as Auto-CASH [9] and MARCO-GE [10], leverage deep reinforcement learning and supervised graph embedding to predict optimal algorithms. However, these approaches face significant hurdles in extracting meaningful meta-features in such dynamic environments. Our approach shares two prerequisites with meta-learning—availability of performance metrics and a variety of algorithms—while being less restrictive and applicable to stream processing and general tasks.

Alternatively, Ensemble methods for time series [11], [12] bear similarities to model selection in stream processing. While enhancing predictive accuracy by aggregating multiple models, they do not inherently address real-time model selection challenges. Ensemble models, adjusting weights based on individual base learners' outputs, may have varying performance and act as another model rather than a selecting model. The need for constant retraining ensemble models limits adaptability in dynamic environments, and the quest for an algorithm that performs universally well on every single point, as noted by Rice [8], remains elusive. In contrast, our problem-agnostic system incorporates these models into a pool and updates selections based on real-time performance. By selecting one individual learner for production instead of running all of them concurrently, our system significantly reduces computational costs while maintaining superior performance in terms of accuracy metrics.

B. Service Computing

Self-adaptation is deeply ingrained in Service Computing, showcasing notable progress through mechanisms like auto-scaling [13], [14] and self-healing [14], along with adaptive resource allocation [15]. The software community's response to the algorithm selection problem primarily centers around service composition [16]. These approaches exhibit adaptability to diverse changes, including changes in data sources [17], and alterations in network dynamics [18]. Many of these approaches employ the MAPE-K loop [19], while some leverage Deep Reinforcement Learning techniques [20], [21]. They distinctly prioritize non-functional aspects like self-healing [22] and quality-of-service metrics [18], covering critical elements like resource utilization and latency [23].

While most contributions in this domain primarily aim to fulfill task requirements and optimize non-functional parameters [19], our work introduces a novel solution specifically tailored for the unpredictable performance of machine learning algorithms. Our approach extends continuous monitoring with "what-if" scenarios for continuous testing, recognizing the high dependence of ML applications on data.

III. Methods

The section opens with an overview of our self-adaptive system architecture and outlines its core functions. The module selection problem is then formally defined, setting the stage for exploring adaptive policies. Subsequently, it introduces two adaptation policies and their corresponding algorithms for module selection. The section concludes by presenting a novel learning method designed to efficiently reduce the costs of running concurrent experiments.

A. Self-Adaptive System

We have developed a Self-Adaptive System [24] capable of adapting to changes in data, where data represents the real world. Additionally, it can adapt to internal modifications, such as adding new modules. The architecture of our proposed self-adaptive system is depicted in Fig. 1.

Drawing upon the terminology from [25], our system is organized into three primary layers:

- **Managing System**: This layer comprises user-defined modules and integrates our proprietary adaptive module selection mechanism, ensuring optimal module selection based on evolving conditions.
- **Managed Subsystem**: Encompassing cloud environments, it includes both production and hypothetical "what-if" scenarios. It provides the foundation for module deployment.
- **Sensing Environment**: This layer interfaces with data that mirrors real-world conditions, ensuring the system's adaptability remains contextually relevant.

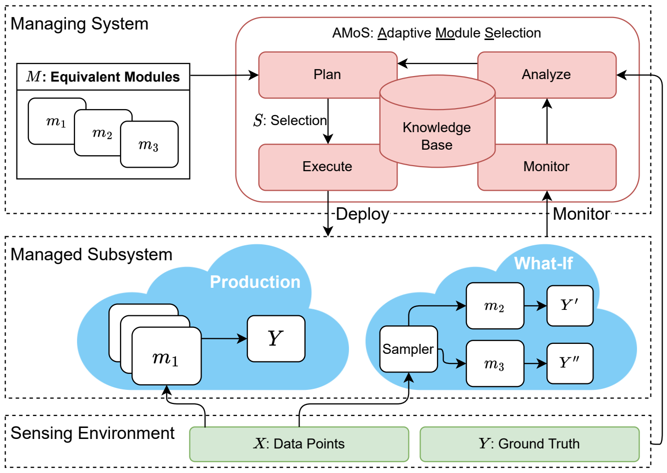

Fig. 1. Proposed Self-adaptive System

The managing system layer comprises equivalent modules that are given to the system by developers and scientists. Two modules qualify as equivalent when they are irreplaceable components in the context of software services. For instance, linear and ridge regression models trained on identical datasets to perform the same function are considered equivalent, despite potential differences in performance outcomes. This equivalence implies that both modules operate on identical input data, execute the same task—albeit using potentially different methodologies—and yield functionally compatible outputs.

The Adaptive Module Selection (AMoS) system manages module deployment across cloud infrastructures. Its key function is to employ adaptation policies, dynamically selecting the appropriate module for the production environment. Additionally, AMoS conducts experiments within a production-like "What-if" environment to enhance the evaluation of module performance, thereby improving deployment decisions. Furthermore, it continuously monitors both the cloud and the sensing environments for timely reactions to any changes encountered.

On the managed subsystem, our primary focus is on the production and "what-if" environments. In line with standard practices, the production environment is given the highest priority, receiving all data and a significant allocation of resources. Modules here are dynamically scaled to manage real-time data input.

The "What-if" environments form an experimentional framework to explore the potential impacts of deploying alternative modules in production. They address the fundamental hypothetical question, "What if a different module operates in production?" Considering operational costs, the "What-if" environments are ephemeral and provided with fewer resources than those available in the production environment, reflecting their exploratory nature.

To manage computational resources effectively, the "What-if" environments use a sampling strategy instead of experimenting on every data point. This approach balances comprehensive analysis with computational efficiency. Moreover, the evaluation does not extend to every possible module; rather, it focuses on a selection of modules considered promising alternatives to the current one in production. Furthermore, the opportunity to operate on archived data provides an understanding of how newer modules might have performed if they had been operational from the start.

This paper does not delve into "What-if" scenario generation, which encompasses decisions about which module to run on which data sample. The development of advanced sampling methods and the strategic selection of modules for these scenarios are subjects for subsequent research. Instead, the present work stands as a proof of concept for the framework, focusing on adaptation policies and module selection algorithms within the AMoS framework.

The sensing environment consists of the immediate data points captured by sensors and the delayed provision of ground truth data. Sensor data are processed in real-time by production modules. To illustrate, consider a scenario specific to predictive tasks. These tasks utilize sensor readings to predict future scenarios. Upon reaching the predicted future moment, the newly captured data serves as the ground truth, providing a benchmark to evaluate the accuracy of the modules' predictions.

In the cases where expert input for verifying predictions or labeling may be demanded for a subset of data points, potentially matching the samples used in "What-if" scenarios. Ground truth is crucial for the validation and calibration of the modules during the machine learning development process. However, the method by which ground truth is acquired is not within the scope of the framework's operations. The framework operates with the premise of accurate ground truth being available for its adaptation process.

B. AMoS: Adaptive Module Selection

Within AMoS, we utilize a control loop based on the renowned MAPE-K model which stands for Monitor, Analyze, Plan, and Execute, over a shared Knowledge base [5]. This knowledge base retains all information about the modules, their performance measures, and the current and desired state of the managed subsystem and the sensing environment.

In the *Monitor* phase, our system continuously collects data from the managed subsystem and the environment. This involves capturing performance metrics from the operational modules in production and 'what-if' environments. Our study centers on the functional requirements of the system, which are intrinsically connected to the task at hand and the underlying business logic. Aligned with XaaS [4], these functional performance metrics are integral to the system, and provided by those who understand the required tasks, e.g., developers and data scientists. They are responsible for setting the standards by which module performance is measured. Accordingly, our system is tasked with gathering this information.

Moving to the *Analyze* phase implements the evaluation methods to find discrepancies between module outputs with the ground truth. AMoS views the evaluation method as an arbitrary error function provided by the developers and data scientists. They take in the modules' outputs and ground truths as inputs and produce outputs that can be quantitatively compared. The values generated by these error functions enable us to compare the effectiveness of different modules. This analysis not only aids in pinpointing performance issues but also provides a foundation for refining the module selection process.

In the *Plan* phase, the system implements selection algorithms to apply the adaptation policies using the insights gathered from the *Analyze* phase. These selection decisions are informed by the current and historical statuses of the managed subsystem and the sensing environment. Using adaptation policies, AMoS decides whether to continue with the current module or to switch to an alternative one to minimize the error value. Moreover, the *Plan* phase identifies which what-if scenarios to run to understand the performance measures of equivalent modules better. Consequently, the choices made in the *Plan* phase determine module deployments in both production and "what-if" environments.

Finally, the *Execute* phase crafts an execution strategy based on the devised adaptation plan. It continuously observes the desired state described in the adaptation plan and reconciles it with the actual deployment in the managed subsystem.

C. Problem Description

This paper specifically focuses on processor modules that receive input and produce output. This includes machine learning models that analyze incoming data and generate predictions, normalizers that adjust and standardize data as it comes in, and post-processing modules that further process data after initial treatment to prepare it for its final form or use.

We define a module m as a function $m : \mathbb{X} \to \mathbb{Y}$ where $\mathbb{X}$ and $\mathbb{Y}$ represents the space of possible inputs and outputs, respectively. Let $M = \{m_0, m_1, m_2, ...\}$ be a sequence of equivalent modules. Two modules m_i and m_j are equivalent if they share the same domain and range. In other words

$$m_i \equiv m_j \iff \mathbb{X}_i = \mathbb{X}_j \quad \wedge \quad \mathbb{Y}_i = \mathbb{Y}_j \tag{1}$$

While equivalent modules may implement different methodologies and potentially yield different outputs, they are designed to be functionally compatible, allowing for their substitution in a given system without disrupting the flow of operations.

Consider input data as a sequence $X = (x_0, x_1, ...)$, we can define a selection sequence, $S = (s_0, s_1, ...)$, where each s_i is a module selected from M to process the corresponding data point x_i. The output resulting from this selection is represented by $\hat{Y} = (\hat{y}_0, \hat{y}_1, \hat{y}_2, ...)$, such that $\hat{y}_i = s_i(x_i)$. Alongside this, we have $Y = (y_0, y_1, y_2, ...), y_i \in \mathbb{Y}$, a sequence representing the ground truth or target values where y_i is corresponding to $\hat{y}_i$.

To assess the effectiveness of a given selection sequence, we introduce two metrics:

- Firstly, the error between our output and the target values, denoted by $err(y_i, \hat{y}_i)$, serves as an arbitrary error function where $err(y_i, \hat{y}_i) \in \mathbb{R}_0^+$.
- Secondly, the "Average Hold Time" metric, represented by the function $\mathcal{C}$ over a period $[0, \tau]$ is defined as:

$$\mathcal{C}(S, \tau) = \frac{\tau + 1}{1 + \sum_{i=0}^{\tau-1} \delta(s_i, s_{i+1})} \quad , \tag{2}$$

where the numerator is the total number of selections and the denominator is the number of changes by time τ. The $\delta(a, b)$ is a delta function used to count the changes between two consecutive selected modules and is defined as:

$$\delta(a,b) = \begin{cases} 0 & a = b \\ 1 & a \neq b \end{cases} \tag{3}$$

This metric gauges how often the selection mechanism swaps the currently deployed module with another.

The objective is to solve a multi-objective optimization problem where the aim is to discover a selection S that both minimizes the cumulative error, $\sum_i err(y_i, \hat{y}_i)$, and maximizes the average hold time $\lim_{\tau\to\infty} \mathcal{C}(S, \tau)$.

D. Adaptation Policies

The choice of the adaptation policy plays a pivotal role in module selection. Such policies should seamlessly balance accuracy with computational efficiency. Therefore, we designed two adaptation policies in this research: Rolling Average and KEEP. Each policy is uniquely designed and evaluated for its efficiency in balancing module performance and redeployment frequency.

a) Rolling Average Policy:: Our inaugural policy is designed to swiftly adapt to changes in module performance. It employs a rolling window approach, assessing the performance of available modules for each data point x_i using the most recent k data points within the window W_i^k, defined as:

$$W_i^k = \{(x_j, y_j) \,|\, i - k \leq j < i\} \tag{4}$$

The policy selects module $s_i = m$ by identifying the one that minimizes the average error $E(W_i^k, s_i)$ within the window, expressed by the inequality:

$$\forall n \in M, \quad E(W_i^k, m) \leq E(W_i^k, n) \tag{5}$$

where the average error is defined as:

$$E(W_i^k, s_i) = \sum_{(x,y)\in W_i^k} \frac{err(y, s_i(x))}{k} \tag{6}$$

The window rolls forward as the data progresses, continually reassessing modules based on their recent performance.

b) KEEP (Keeping Errors down with Enhanced Persistence) Policy:: Tailored to minimize errors while avoiding unnecessary module switches, this policy extends the principles of the rolling average. It introduces an additional feature that preserves the current module m unless its relative error to the optimal module n exceeds a predefined threshold θ. The relative error is computed as the percentage difference between the errors of the current and optimal modules. A switch to the optimal module occurs when the relative error surpasses the threshold, as expressed by the inequality:

$$\frac{|E(W_i^k, m) - E(W_i^k, n)|}{E(W_i^k, n)} > \theta \tag{7}$$

E. Adaptive Selection Algorithm

This work presents an algorithm, detailed in Alg. 1, crafted to employ the aforementioned adaptation policies. This algorithm is configurable and can be tailored through its input parameters to embody and enforce the specified policies as required.

Algorithm 1 Adaptive Module Selection

```
Require: $m$, $i$, $k$, $\theta$
 1: choice $\leftarrow m$
 2: for $s \in M$ do
 3:     $\overline{E}[s] \leftarrow E(W_i^k, s)$
 4: end for
 5: optimal-err $\leftarrow$ min($\overline{E}$)
 6: optimal-module $\leftarrow$ $\overline{E}$.indexOf(optimal-err)
 7: if $\frac{|E(W_i^k, m) - \text{optimal-err}|}{\text{optimal-err}} > \theta$ then
 8:     choice $\leftarrow$ optimal-module
 9: end if
10: return choice
```

The algorithm takes four input parameters: the currently selected module m, the current time index i, a window size k for error calculation, and a performance threshold θ. Its default action is to maintain the currently selected module (line 1 of the algorithm). Subsequently, the algorithm computes the average error $\overline{E}$, over the specified window size k for each module within the pool of candidate modules M (lines 2-4). It identifies the optimal error, the lowest average error from these computations, and the module with the optimal error (line 4-5). The subsequent step involves assessing whether the deviation between the error of the current module and the optimal error exceeds the predefined threshold θ. If this difference is greater than the threshold, the algorithm selects the module with the optimal error (lines 7-9). Otherwise, it retains the current module as the chosen one. Finally, the algorithm outputs the selected choice, which is the module to be deployed in the production environment (line 10).

F. LBS (Learning-Based Selection) Method:

The LBS method is designed to reduce computational overhead by eliminating the need to evaluate promising modules. It leverages a learning model to predict the best module choices, recognizing that adaptation policies typically select modules following a detectable pattern related to input data. This approach proves advantageous for prolonged tasks that generate enough data to identify these selection patterns.

Although the "what-if" scenarios are designed to be lightweight on sampled data using limited modules, the benefits of information gained from experimentations relative to the costs begin to drop over time. LBS addresses this issue. Upon detection of a consistent selection pattern, it can learn to make informed decisions, thus reducing the need for further experimentation.

Specifically, LBS learns from the history of module choices made under a particular policy. It replicates the outcome of a conventional selection algorithm without actually performing

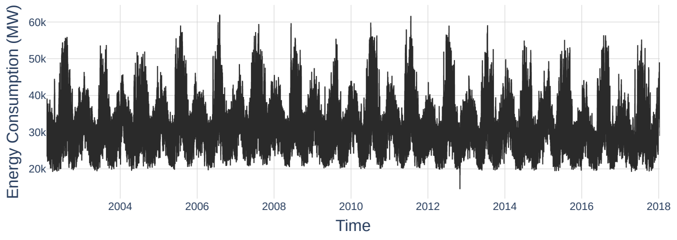

Fig. 2. Dataset Visualization

the selection computation each time, thus saving computational effort. For example, it can learn from the decisions of the selection algorithm or adhere to benchmarks such as "Best-RMSE", detailed in subsection IV-B, to select top-performing modules. This showcases its adaptability across various selection criteria.

The type of predictive model used in LBS can be adapted to the task at hand. The learning component is fed the same input as the operational modules, to predict the chosen module for each data point.

To apply LBS, we leverage the model with the best past performance from our module library. This model, already fine-tuned to reveal patterns in the input data, is further trained with the task-specific data and expected outcomes. Since this model is adept at identifying trends, it becomes a natural choice for implementing LBS, requiring minimal task-specific adjustment.

IV. Evaluation

This section evaluates the performance of our self-adaptive system using a real-world use case, adhering to established practices for assessing adaptive systems [26]. It begins by introducing the use case, detailing the experimental setup, and presenting the results from parameter adjustments in the implementation of adaptation policies.

A. Use case

This use case's primary objective is to forecast regional energy consumption using historical consumption patterns. We work with a public dataset named "PJM Hourly Energy Consumption," available on Kaggle[1]. PJM Interconnection LLC, commonly referred to as PJM, is a Regional Transmission Organization (RTO) in the United States that operates within the Eastern Interconnection grid. The dataset captures hourly power consumption from PJM's official website, presented in megawatts (MW). The problem is to predict the upcoming energy consumption based on the previous consumption pattern.

We utilized the East Region file data, `PJME_hourly.csv`. This dataset comprises two columns: `datetime`, which represents the timestamp, and `PJME_MW`, which denotes the energy consumption in megawatts (MW). A visualization of this data can be seen in Fig. 2, while its characteristics are detailed in Table I.

[1]www.kaggle.com/datasets/robikscube/hourly-energy-consumption

TABLE I
Dataset Characteristics

Duration	Records	Mean	Standard Deviation	Min	Max
2002-2018	143207	32111	6486	14544	62009

B. Experimental Setup

We simulated a stream processing environment to mirror its operation in a real-world production context. Initially, we based our approach on a dataset spanning one year. Using this first-year data, we trained our modules. Every week after that, we refined our modules, training on the most recent data to forecast the hourly energy consumption for the following week.

We operated on the premise that a singular module is actively deployed and scaled in the actual production environment. Concurrently, other modules process a subset of the primary data in "what-if" environments. This approach lets us harness insights from all modules, ensuring informed decisions when selecting the most effective module.

Our choice of modules was inspired by the top-performing Kaggle notebooks for this dataset. We selected three standout modules: Prophet[2], XGBoost[3], and Light GBM[4]. We implemented two stacking methods to gauge the efficacy of our module selection technique against ensemble approaches. These stacking methods utilized the predictions from the three aforementioned modules, with XGBoost and Light GBM serving as our primary stacking models.

To assess the relative performance of our adaptation policies, we introduced two distinct selection benchmarks:

- **Selection (Best-RMSE)**: This method provides a benchmark for the optimal module selection based on minimizing the RMSE (Root Mean Square Error). It offers insight into the potential room for improvement in our selection techniques.
- **Selection (Random)**: As a contrasting method, this approach randomly selects a module for each data record. We executed this method 1,000 times and reported the mean outcome, allowing us to gauge the significance of our selected methods compared to a random choice.

As a baseline, we introduce a benchmark termed "static," representing the current practice in machine learning development where the optimal module from a set is chosen and consistently employed. Fig. 3 showcases the modules and the benchmark results. Among the five modules, the Light GBM exhibits the best average performance, thus earning its place as the static benchmark. However, when examining its difference with the benchmark Selection (BEST-RMSE), it is evident that the optimal module shifts over time. To illustrate this, we provide a detailed view of the daily mean absolute errors of each module in Fig. 4, capturing a week from 2017. This demonstrates that,

[2]www.kaggle.com/code/robikscube/time-series-forecasting-with-prophet
[3]www.kaggle.com/code/robikscube/tutorial-time-series-forecasting-with-xgboost
[4]www.kaggle.com/code/alirezajavid1999/energy-consumption-prediction-timeseries-analysis

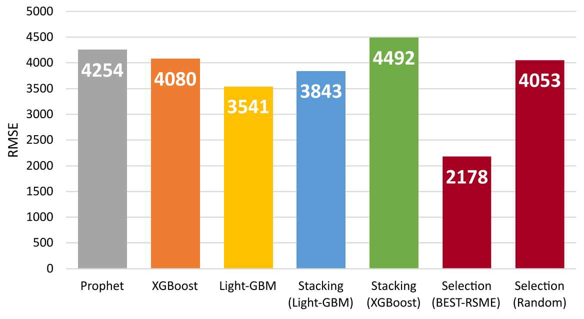

Fig. 3. Modules and benchmarks performance

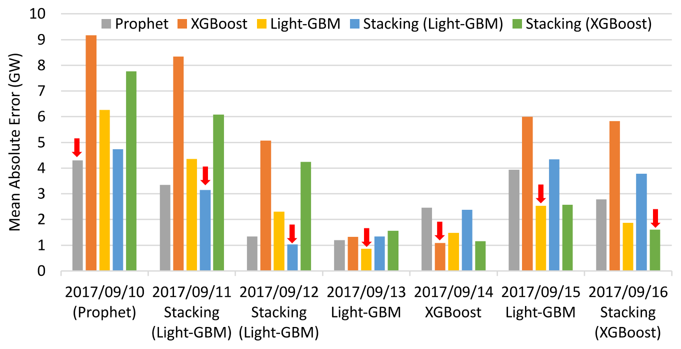

Fig. 4. Daily mean absolute error of modules from a week of 2017 in gigawatt. The red arrows point to the best module, with its name located beneath the date on the X-axis label.

despite LightGBM's average superiority and the availability of 15 years of training data, no single module consistently outperforms the others at every moment. As a result, the best module selection can differ, underscoring the importance of informed decision-making for different periods.

The selection algorithm applies the rolling average policy when the threshold (θ) is set to zero. Setting this threshold to a positive non-zero value applies the KEEP policy. For the LBS implementation, we employed the LightGBM model due to its superior performance on this data. This model was retrained using the same hyperparameters previously optimized for power prediction, but now to predict the selection sequence.

C. Results

We executed our selection algorithm using varied parameters to assess their effects. In these evaluations, we incorporated both the RMSE (Root Mean Square Error) and "Average Hold Time." For a clear depiction of these trends, we employed dual-axis plots. The left axis, highlighted in orange, represents the RMSE, while the right axis, in blue, shows the average hold time in hours. The X-axis denotes the adjustable parameter specific to each adaptation policy, particularly, windows size k and threshold θ for Rolling Average and KEEP policies, respectively.

Fig. 5.a delineates the performance of the rolling average policy for different window sizes. The optimal RMSE is achieved when using a window size of one. This observation implies that decisions based on the most recent data record frequently yield the best results. It indicates a strong correlation between consecutive data points, resulting in RMSE values closely approximating the best achievable performance. Conversely, enlarging the window size tends to decrease the frequency of module changes, consequently lowering the expenses linked to redeployment.

We set the window size to one for the KEEP policy evaluation since it performs the best according to previous experiments. The findings are illustrated in Fig. 5.b. As anticipated, an increase in the threshold results in fewer changes, but this is at the expense of a deteriorating RMSE. Notably, this policy can offer enhanced performance at comparable change frequencies with respect to the rolling average policy.

To better facilitate comparison of these two policies, Fig. 6 presents a two-dimensional plot of RMSE versus average hold time using the same data points presented in Fig. 5. Optimal performance is achieved when points lie towards the bottom right corner, as this represents lower RMSE and higher rates of change. For instance, at a 5% threshold, the average hold time stands at 9.4 hours with an RMSE of 2506. In contrast, using the rolling average approach to achieve a similar RMSE of around 2500 necessitates redeployment every 5 hours using a 4-hour window size.

We tested our Learning-Based Selection (LBS) method against two benchmarks: the Best-RMSE and the KEEP adaptation policy as shown in Fig. 7 and Fig. 8, respectively. For the KEEP policy, we set the window size $k = 1$ and the performance threshold $(\theta = 5\%)$, as these parameters typically provide satisfactory results. The primary variable in our experiments was the point in time when we began to apply LBS, meaning the quantity of data available for training LBS's model selection framework. In the illustrations of results, dashed lines illustrate benchmark values. It is important to note that the goal of the LBS model is to learn the selection pattern rather than directly predict RMSE and Average Hold Time. These metrics are consequential to the accuracy of LBS's module selection.

As depicted in Fig. 7, for the Best-RMSE benchmark, two years of data from running "what-if" scenarios can deliver an RMSE close to 2500, with an average hold time of ∼3.1,

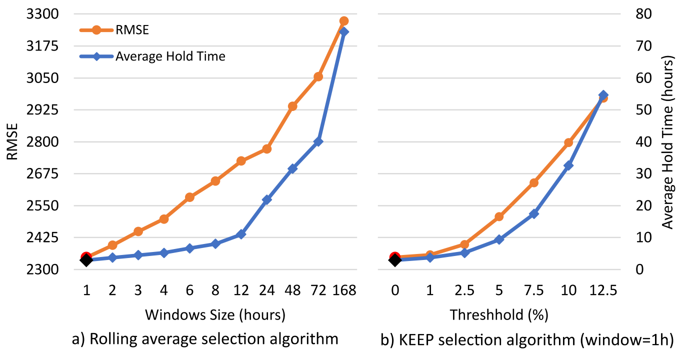

Fig. 5. Rolling average and KEEP selection policies performance: In subfigures a) and b), the red circle and black diamond represent the same data points.

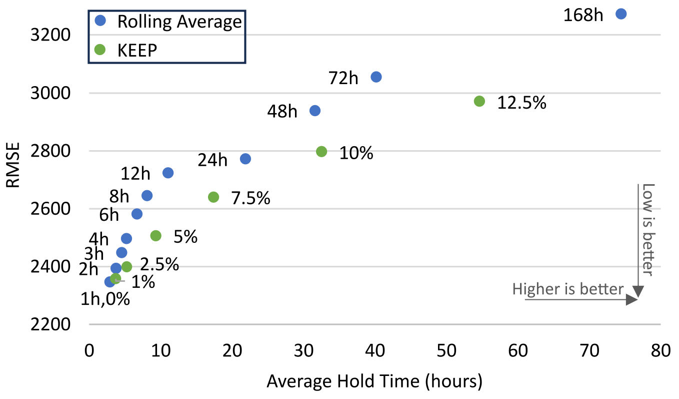

Fig. 6. Rolling average and KEEP selection algorithms comparisons. Optimal performance occurs towards lower RMSE and higher rates of change.

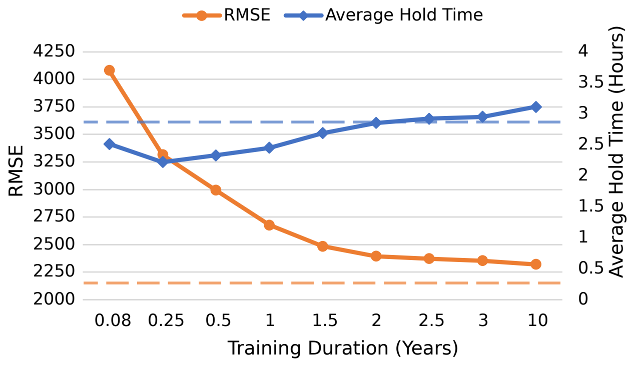

Fig. 7. LBS performance trained on Best-RMSE selection benchmark over varying training spans.

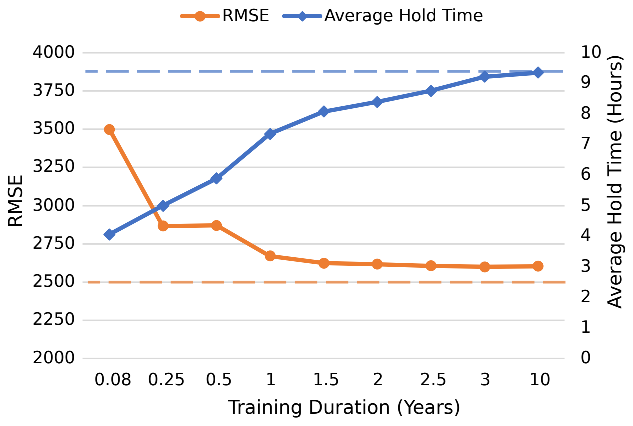

Fig. 8. LBS performance trained on KEEP policy over varying training spans.

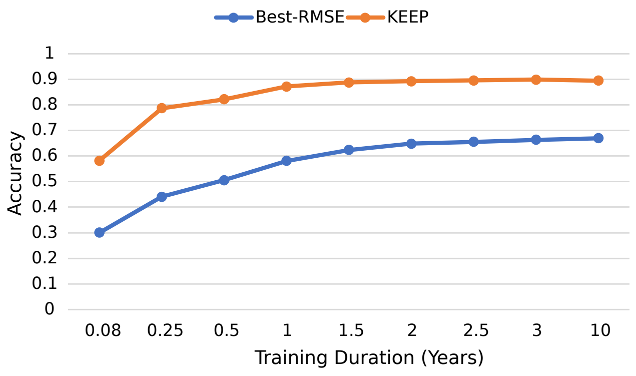

Fig. 9. LBS accuracy of predicting the selection sequence over varying training spans.

roughly equal to the benchmark. A similar trend is observed in Fig. 8 for the KEEP policy, where, using the LBS method eliminates the need for additional experiments while providing comparable results after approximately one year. There's an inherent performance trade-off: more extensive training typically enhances performance. However, as previously mentioned, the incremental benefits may not justify the costs of continuous what-if scenario testing. Despite this, the decision-making process tends to improve in both RMSE and average hold time as more data becomes available.

Furthermore, the LightGBM model's performance, utilized within the LBS framework, underpins this observation. Fig. 9 shows that accuracy trends correlate with RMSE and average hold time improvements. In the initial stages, with limited data equivalent to just one month (or 0.08 years), the LBS's selections are nearly random, but accuracy improves as it is trained on more data up to the point where further experiments bring little to no additional value for the selection task.

Moreover, it's particularly noteworthy that the KEEP policy, which is characterized by less frequent switching between modules, achieves a higher accuracy rate in comparison to the Best-RMSE benchmark, which tends to oscillate more frequently. This indicates that a consistent adaptation policy like KEEP can leverage LBS to minimize switching and experimentation expenses, albeit potentially compromising performance metrics such as RMSE.

V. Conclusion

In the evolving Machine Learning (ML) realm, deploying modules effectively for streaming data remains challenging due to the continuous evolution of both models and input data. To tackle this challenge, we introduce a self-adaptive system designed to dynamically select machine learning modules in streaming data processing. This system manages the delicate balance between module performance and redeployment costs. Guided by the MAPE-K model, our approach monitors ML module performance on both production and experimental data, selecting the most suitable module based on observed performance metrics. The inclusion of a "what-if" scenario mechanism enables continuous evaluation of available modules. We designed a selection algorithm to implement two adaptation policies intelligently to determine the optimal module for the incoming data. Utilizing a learning-based approach, our system's efficiency is enhanced by identifying selection patterns, consequently discontinuing additional experimental data gathering. The algorithm and the learning-based method are complementary for optimizing performance, redeployment costs, and computational power.

The efficacy of our system in fluctuating streaming scenarios was demonstrated through an energy consumption forecasting use case. The employment of the KEEP adaptation policy was found to be superior to both static module selection and ensemble methods. This policy offers flexibility, allowing users to tailor the system's responsiveness to balance performance against the increased costs associated with more frequent redeployment. The approach is enhanced with a Learning-Based Selection (LBS) method method. When the pattern of optimal choices remains stable or a suboptimal decision isn't critical, LBS stands out as a recommendation to reduce computational expenses. Future directions include refining selection algorithms, delving deeper into the integrated KEEP-LBS methodology, optimizing retraining periods, investigating sampling mechanisms, and expanding the system's applicability across diverse domains and use cases.

REFERENCES

[1] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[2] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, pp. 1–25, 2009.

[3] D. Weyn, I. Gerostathopoulos, N. Abbas, J. Andersson, S. Biffl, P. Brada, T. Bures, A. D. Salle, P. Lago, A. Musil, J. Musil, and P. Pelliccione, "Preliminary results of a survey on the use of self-adaptation in industry," in *2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2022, pp. 70–76.

[4] M. Hadadian Nejad Yousefi, V. Degeler, and A. Lazovik, "Empowering machine learning development with service-oriented computing principles," in *"Service-Oriented Computing"*, Springer. "Springer International Publishing", 2023.

[5] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, 2006.

[6] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, 2022.

[7] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[8] J. R. Rice, *The Algorithm Selection Problem*, ser. Advances in Computers. Elsevier, 1976, vol. 15.

[9] T. Mu, H. Wang, C. Wang, Z. Liang, and X. Shao, "Auto-cash: A meta-learning embedding approach for autonomous classification algorithm selection," *Information Sciences*, vol. 591, pp. 344–364, 2022.

[10] N. Cohen-Shapira and L. Rokach, "Automatic selection of clustering algorithms using supervised graph embedding," *Information Sciences*, vol. 577, pp. 824–851, 2021.

[11] J. Gastinger, S. Nicolas, D. Stepić, M. Schmidt, and A. Schülke, "A study on ensemble learning for time series forecasting and the need for meta-learning," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.

[12] P. Montero-Manso, G. Athanasopoulos, R. J. Hyndman, and T. S. Talagala, "Fforma: Feature-based forecast model averaging," *International Journal of Forecasting*, vol. 36, no. 1, pp. 86–92, 2020.

[13] J. D. Pereira, R. Silva, N. Antunes, J. L. M. Silva, B. De França, R. Moraes, and M. Vieira, "A platform to enable self-adaptive cloud applications using trustworthiness properties," in *2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2020, pp. 71–77.

[14] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, "Self-managing cloud-native applications: Design, implementation, and experience," *Future Generation Computer Systems*, vol. 72, pp. 165–179, 2017.

[15] S. Ghari, "Devops for digital business: Optimizing the performance and economic efficiency of software products for digital business," in *2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2022, pp. 53–57.

[16] R. R. Aschoff and A. Zisman, "Proactive adaptation of service composition," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012, pp. 1–10.

[17] E. Eryilmaz, M. A. Khan, F. Trollmann, and S. Albayrak, "Adaptive service selection for enabling the mobility of autonomous vehicles," in *Ambient Intelligence*, I. Chatzigiannakis, B. De Ruyter, and I. Mavrommati, Eds. Cham: Springer International Publishing, 2019, pp. 203–218.

[18] D. Zhao, Z. Zhou, P. C. K. Hung, S. Deng, X. Xue, and W. Gaaloul, "Ctl-based adaptive service composition in edge networks," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1051–1065, 2023.

[19] M. Filho, E. Pimentel, W. Pereira, P. H. M. Maia, and M. I. Cortés, "Self-adaptive microservice-based systems - landscape and research opportunities," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2021, pp. 167–178.

[20] H. Wang, M. Gu, Q. Yu, Y. Tao, J. Li, H. Fei, J. Yan, W. Zhao, and T. Hong, "Adaptive and large-scale service composition based on deep reinforcement learning," *Knowledge-Based Systems*, vol. 180, pp. 75–90, 2019.

[21] J.-W. Liu, L.-Q. Hu, Z.-Q. Cai, L.-N. Xing, and X. Tan, "Large-scale and adaptive service composition based on deep reinforcement learning," *Journal of Visual Communication and Image Representation*, vol. 65, p. 102687, 2019.

[22] Y. Wang, S. Wang, B. Yang, B. Gao, and S. Wang, "An effective adaptive adjustment method for service composition exception handling in cloud manufacturing," *Journal of Intelligent Manufacturing*, pp. 1–17, 2022.

[23] S. Kumar, T. Chen, R. Bahsoon, and R. Buyya, "Datesso: Self-adapting service composition with debt-aware two levels constraint reasoning," in *2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2020, pp. 96–107.

[24] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 3, pp. 54–62, 1999.

[25] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, *On Patterns for Decentralized Control in Self-Adaptive Systems*. Berlin, Heidelberg: Springer, 2013, pp. 76–107.

[26] I. Gerostathopoulos, T. Vogel, D. Weyns, and P. Lago, "How do we evaluate self-adaptive software systems?: A ten-year perspective of seams," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2021, pp. 59–70.