Meta-Learning for Time Series-based Tasks in Software Engineering

Federico Di Menna, Luca Traini, and Vittorio Cortellessa

***Abstract*—Time series analysis is widely employed in software engineering (SE) for various critical activities, including capacity planning, anomaly detection, and performance testing. However, effectively incorporating time series analysis into SE processes can require extensive experimentation or specialized expertise, which may not often be readily accessible to software engineers. Meta-learning offers a promising avenue to reduce human intervention by automating several aspects of the time series analysis process, such as selecting the most appropriate algorithm. Although this approach has shown promising results across various time series domains, its applicability and potential impact within SE contexts still remain largely unexplored.**

In this paper, we report the first empirical evidence on the effectiveness of meta-learning for time series–based SE tasks. First, we introduce MALIAS, a framework inspired by the meta-learning principles, which leverages over 700 time series features, a random forest model, and best practices such as dimensionality reduction and hyper-parameter tuning, to automatically select and run the best-performing algorithm for a given software time series. Then, we instantiate this framework for three distinct time series-based SE tasks–forecasting of software telemetry data, anomaly detection in online software systems, and steady-state detection in software performance testing–and we conduct an extensive evaluation across six different real-world datasets. Results show that MALIAS algorithm selection yields net improvements, over individual algorithms, up to +17.7 percentage points (*pp*) for forecasting, up to +20*pp* for anomaly detection, and up to +19.8*pp* for steady-state detection. Furthermore, when compared to other automated approaches, such as established AutoML frameworks, MALIAS consistently outperforms them across all the considered tasks with very high statistical significance ($p<0.0001$), while showing competitive training and testing times, despite heterogeneous results. These findings highlight the potential of meta-learning for time series analysis in SE, paving the way for its broader adoption in both research and practice.

***Index Terms*—Meta-Learning, Software Time Series, AIOps, Anomaly Detection, Time Series Forecasting, Software Performance Testing**

I. Introduction

Time series are extensively used for modeling a wide range of phenomena, spanning fields from climatology to economics and beyond [6]. Due to its versatility, the field of time series analysis has significantly grown over recent decades with the introduction of countless algorithms designed for tasks such as forecasting, classification, and anomaly detection [7]–[9]. These advancements have also started to gain attention within the Software Engineering (SE) domain [4], [10]–[16]. In particular, researchers have begun to employ time series analysis across a variety of SE tasks, often related to several critical software quality assurance activities. For example, time series forecasting has been employed to predict future software telemetry data [12]–[14], thus supporting crucial activities such as resource capacity planning [10], [11], [17], [18], software self-adaptation [19], [20] and more [21]–[23]. Time series anomaly detection has been utilized to automatically identify unexpected issues in online software systems and prevent major failures [15], [24]–[26]. Time series classification have been employed to address the inherent trade-off between result quality and execution time in software performance testing [4], [27]–[29].

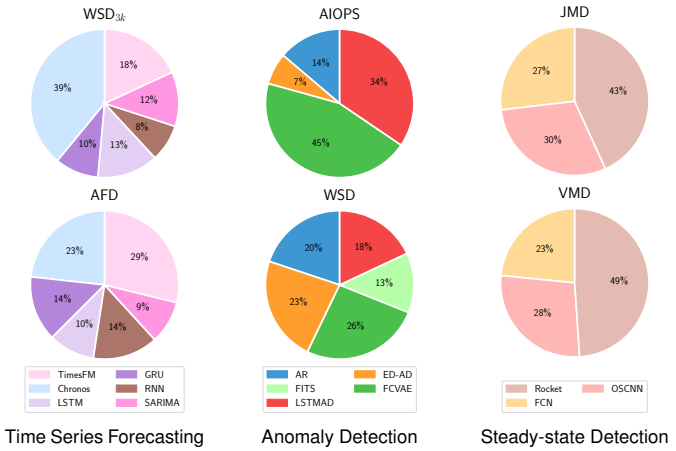


Fig. 1. Distribution of best-performing algorithms on six software time series datasets (WSD_{3k} [1], AFD [2], AIOPS [3], WSD [1], JMD [4], VMD [5]) spanning three different SE tasks (Time Series Forecasting, Anomaly Detection, Steady-state Detection). Each pie chart shows the percentage of instances in which each algorithm ranked first in performance compared to the other algorithms.

Despite these developments, the effective application of time series analysis in SE remains fundamentally constrained by several critical challenges. One of the main difficulties lies in selecting the most appropriate algorithm for the target time series data. Indeed, the effectiveness of time series analysis algorithms often greatly varies depending on the characteristics of the data. This issue is particularly pronounced in SE contexts, where time series may represent diverse software metrics (*e.g.,* response times, software faults, CPU utilization) whose behavior is often highly variable [30] and influenced by external factors such as workload dynamics [31]. We observed clear evidence of this phenomenon in our previous empirical studies [12], [13], where we found that SARIMA [32]–a well-established algorithm for time series forecasting–performed suboptimally on most types of software telemetry data, whereas it proved to be particularly effective on a

The authors are with the Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, 67100 L'Aquila, Italy (e-mail: federico.dimenna@graduate.univaq.it, {luca.traini, vittorio.cortellessa}@univaq.it).

few specific types. Figure 1 provides further evidence of this problem. The figure illustrates how frequently different algorithms achieve the top performance across six time series datasets spanning three distinct SE tasks: forecasting software telemetry data, anomaly detection in software systems, and steady-state detection in performance testing. Each pie chart shows the proportion of time series instances within a dataset for which each algorithm achieved the best performance. The figure clearly demonstrates that no single algorithm consistently achieves the best performance in more than 50% of the cases. Consequently, even if one selects the algorithm that is on average the best for a specific SE task, it will still lead to suboptimal outcomes in more than half of the time series instances. This situation underscores the importance of carefully choosing the most suitable algorithm. However, this process may require specialized expertise and/or extensive experimentation, which may not be readily available to software engineers seeking to incorporate time series analysis into their SE processes.

Over the years, the data science literature has proposed various approaches to automating the selection of algorithms for time series analysis. These approaches typically fall under the umbrella of meta-learning [33], leveraging descriptive features extracted from time series data to predict the most suitable algorithm. While such approaches have demonstrated effectiveness across several time series domains [33]–[36], their efficacy within SE remains largely underexplored. Indeed, the application of meta-learning has the potential to provide substantial benefits across a wide range of SE activities. Yet, there is still little empirical evidence on whether–and to what extent–it can improve time series-based SE tasks.

In this work, we aim to fill this knowledge gap by conducting the first empirical investigation on the effectiveness of meta-learning in time series–based SE tasks. To this end, we design MALIAS (MetA LearnIng for softwAre time-Series), a meta-learning framework that dynamically selects the best algorithm based on the characteristics of the time series and the SE task at hand. MALIAS first extracts about 700 features from each time series and employs these features to train a random forest model that predicts the performance of different algorithms. This process follows best practices in machine learning, such as dimensionality reduction and hyper-parameter tuning. The predicted algorithm performance are subsequently employed in the inference phase to automatically select and run the best algorithm for each given time series. We evaluate MALIAS on 8,561 real-world software time series from six publicly available datasets. For forecasting, we selected datasets aimed at predicting future telemetry data of online software systems [1], [2]. For anomaly detection, we utilized datasets containing telemetry data from online software systems with labeled time steps marking known software issues [1], [3]. Lastly, for classification, we leveraged datasets addressing the steady-state detection problem [5], [37], which is commonly used to enhance the cost-effectiveness of software performance testing [4], [28], [38]. Results show that, when compared to the best performing algorithm for each dataset, MALIAS provides net improvements of +17.7 percentage points (*pp*) and +9.5*pp* for time series forecasting, +20.7*pp* and +3.1*pp* for anomaly detection, and +9.8*pp* and +19.8*pp* for steady-state detection. Our results also highlight that a MALIAS instance trained in one domain cannot be readily transferred to another one as a pre-trained solution, as it requires an initial training phase on a set of time series collected from the target domain. Furthermore, MALIAS outperforms two well-known AutoML frameworks with very high statistical significance ($p < 0.0001$). However, regarding training and testing times, its efficiency varies across tasks, achieving the fastest execution only on time series forecasting training and on steady-state detection testing.

Our main contributions are as follows:

- A first empirical evidence of the effectiveness of meta-learning for time series-based SE tasks, based on an evaluation of MALIAS across six real-world time series datasets, spanning three different SE tasks.
- A comparative evaluation of MALIAS with two established AutoML frameworks in terms of effectiveness and efficiency.
- Source code, experimental scripts, and results of MALIAS publicly available [39] to support both developers and researchers.

II. Background and Related Work

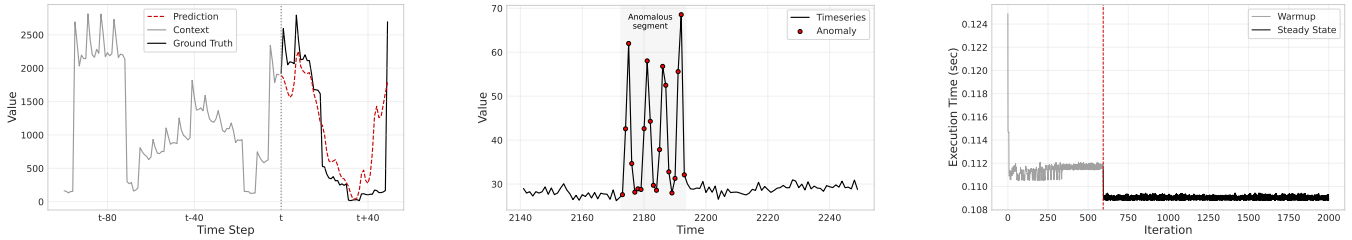
A. Time Series-based Tasks in SE

The concept of time series has been widely employed in SE domain due to its flexibility in modeling software-related phenomena that evolve over time. Time series analysis tasks can be broadly divided in three main categories: *time series forecasting*, *anomaly detection*, and *time series classification*. Below, we report examples of application of time series analysis in SE for each of these three categories.

1) Time Series Forecasting (TSF): TSF aims at predicting future values of a temporally ordered sequence by learning patterns and structures inherent to the historical observations, typically accounting for temporal dependencies, trends, seasonality, and noise within the data. TSF has been extensively applied in the SE domain, with methods ranging from ARIMA models [10], [17], [21] to neural networks [12], [13], [23], [26]. A large body of research in this area focuses on forecasting telemetry data from online software systems—such as software performance metrics (*e.g.,* memory/CPU usage, response time) [12]–[14], [23], [26] or system workloads [17], [18], [20]—to support activities like capacity planning [17], [18], self-adaptation [19], and software rejuvenation [23].

Beyond software telemetry data, TSF has also been used to model the temporal evolution of software systems in terms of growth [40], [41], reliability [21], [22], bug report trends [10], and technical debt accumulation [11].

In this work, we focus on the prediction of software telemetry data as a case study to evaluate MALIAS, since forecasting telemetry data is one of the most common applications of TSF in SE. Figure 2a illustrates a real example of TSF applied to predicting the workload of an Azure serverless function [2]. The figure shows the number of function invocations over time at ten-minute intervals. A TSF model (Chronos [42]) is employed to generate predictions for the next 50 future values,



(a) Time Series Forecasting of workload in a serverless environment [2].

(b) Time Series Anomaly Detection of software telemetry data from AIOps dataset [3].

(c) The steady-state phenomenon in the *fasta* benchmark [5].

Fig. 2. Illustrative examples of practical scenarios involving time series-based tasks in software engineering.

namely from the time step "$t+1$" to "$t+50$". The grey line denotes the model's input context, the red dashed line indicates the forecasted values, and the ground truth is depicted in black.

2) Time Series Anomaly Detection (TSAD): Anomaly detection involves identifying data points, patterns, or segments within a time-ordered sequence that significantly deviate from the expected behavior or underlying structure of the previously encountered data in the time series. Detecting anomalous data points within a set of observations is a critical capability in SE [26], [43]. This is particularly important in modern online software systems, where software undergoes continuous code changes [44] and is subject to highly variable workloads [31], making such systems more susceptible to unexpected issues [45], [46]. As a result, the adoption of anomaly detection techniques for software telemetry data has significantly increased in recent years, enabling the early identification of faults [24] and performance degradations [26], [47]. Applications span over different domains such as spacecraft software [25], digital IT infrastructures [24], and microservices-based systems [48]. This growing interest has even led to the development of specialized TSAD benchmarks tailored for SE, such as TimeSeriesBench [15]. TimeSeriesBench employs a curated collection of software telemetry datasets with known anomalies, accompanied by associated algorithms and a standardized process for evaluation and comparison. According to this benchmark, state-of-the-art approaches for TSAD in SE are predominantly based on deep learning models [15].

Figure 2b illustrates examples of known anomalies in software telemetry data from the AIOps dataset [3]. The anomalous data points are highlighted in red.

3) Time Series Classification (TSC): TSC refers to categorizing entire sequences or segments of time series data into predefined classes based on learned temporal patterns and characteristics. A typical application of TSC in software engineering is the so-called *steady-state detection* problem in performance testing [5], [37].

This problem is particularly relevant in software that run on virtual machines that support Just-In-Time (JIT) compilation, such as Java, where performance measurements during the early iterations of test executions are often unstable due to runtime warm-up effects of JIT compilation. Such variability makes these early measurements unsuitable for rigorous performance evaluation.

Figure 2c illustrates this phenomenon using the *fasta* benchmark from the computer language benchmarks game [5], where the x-axis denotes successive benchmark executions and the y-axis indicates its execution time. In this example, the system reaches a steady state around the 600th iteration.

To detect the onset of steady-state behavior at runtime, researchers have developed a variety of techniques based on TSC. These approaches classify incoming segments of performance measurements as either "steady" or "non-steady," using either statistical heuristics [28], [29] or deep learning models [4], [27], and are typically employed to dynamically halt warm-up iterations at runtime during performance testing.

B. Time Series and Meta-Learning

Several previous studies have empirically demonstrated that there is no universally best algorithm for time series analysis [9], [12], [13], [49]–[51]. As a result, a common practice is to rely either on the domain expertise of data analysts or on extensive empirical evaluation of available models to identify the algorithm that yields the best performance. However, expert knowledge is not always readily accessible, and exhaustive experimentation can be impractical due to time and cost constraints.

In light of these challenges, researchers have begun exploring automated approaches for selecting the best-performing models, leveraging the principles of meta-learning [33]–[35], [52]–[59]. These approaches typically involve extracting descriptive features from time series data and training a predictor to identify the most suitable algorithm based on those features. For instance, Prudêncio *et al.* [34] introduced a method in which a meta-learner is trained to solve a classification problem, predicting the best algorithm for TSF. Shahoud *et al.* [35] proposed DSTMS, a descriptive schema for energy time series data, designed to support effective algorithm selection for energy forecasting activities. Smith and Miles [33] formulated the algorithm selection task originally proposed by Rice [60] as a learning problem, demonstrating its applicability across various domains, including TSF. Talagala *et al.* [36] proposed the FFORMS framework, which classifies forecasting models based on over 30 time series features (*e.g.,* trend, seasonality, autocorrelation).

Building on the intuition of these works, this paper introduces MALIAS, a meta-learning approach specifically tailored

to the SE domain. To the best of our knowledge, none of the previous studies have explored meta-learning for time series-based tasks in SE.

III. Approach

We present MALIAS (MetA LearnIng for softwAre time-Series), a meta-learning framework designed to select and run the most effective algorithm for a given time series-based SE task.

A. Overview

MALIAS leverages a large number of time series features—which we refer to as *meta-features*—to select the most suitable algorithm for a given time series. To make this selection, MALIAS *learns* how each algorithm performs across individual time series based on their corresponding performance metrics, which we refer to as *target metrics*. These target metrics may vary depending on the task. For instance, they may be error-based in the case of TSF tasks (*e.g.,* SMAPE), or accuracy-based in the case of TSC tasks (*e.g.,* F_β score). At its core, MALIAS employs a *Random Forest* regressor model that, given the set of meta-features for a time series, predicts a vector of target metric values $\vec{m} = \langle m_1, m_2, \ldots, m_k \rangle$, where each m_i denotes the predicted target metric for candidate algorithm i on the provided time series. This prediction vector is then used to select the best-performing algorithm for the given SE task.

Figure 3 provides an overview of our approach, which consists of two stages: (i) an initial training stage and (ii) an application stage. The training stage begins with a set of training time series and a pool of k candidate algorithms. MALIAS executes each of the k algorithms on the training time series to obtain their respective target metrics. Simultaneously, it extracts *meta-features* from each time series. These meta-features, along with the corresponding metrics, are used to train a Random Forest model that learns to predict the vector of target metric values $\vec{m}$. This process yields a trained *meta-learner* capable of estimating algorithm performance based on the time series *meta-features*.

The application stage represents the practical usage scenario of MALIAS. Given a previously unseen time series, MALIAS first extracts the corresponding *meta-features* and then uses them as input to the meta-learner to predict the performance of the k candidate algorithms—namely, the vector of *target metrics* $\vec{m}$. Afterwards, MALIAS selects the best algorithm based on the predicted target metrics $\vec{m}$ and executes it on-the-fly. Depending on the specific SE task, the relevance of each *meta-feature* might change based on the relationship with the task *target metric*. For this reason, the proposed framework is task-dependent, namely a specific instance of MALIAS is uniquely dedicated to a specific SE task.

B. Meta-Features and Target Metrics

MALIAS collects hundreds of *meta-features* for each time series. Table I provides an illustrative example subset of the extracted features types. The actual features used in our work are specific instantiations of those types, created by selecting particular parameter values. For instance, a specific *fft_coefficient* feature can be defined by setting the *coefficient* parameter to `10` and the *attr* parameter to `abs`, thereby returning the real component of the complex value. As shown in the table, these features spread from the basic statistical properties (*e.g.,* quantile, autocorrelation) to more complex ones, such as frequency domain features [61]. The extraction process results in up to 771 distinct features[1] for each time series. Considering the substantial amount of extracted *meta-features*, we decide to apply a feature selection step for narrowing the selection to the most relevant *meta-features* for each specific task. Specifically, the feature selection technique is composed of two sequential filters. The initial filter aims to mitigate multicollinearity by removing redundant features that exhibit high similarity. In order to do that, the Pearson correlation coefficient is calculated among all the pairs of features, and those that have a correlation coefficient greater than *0.8* are ignored. We set the filtering cutoff value to 0.8 based on the commonly accepted convention that correlation coefficients exceeding 0.8 indicate probable collinearity [62], [63]. We refer to this step as *ρ-filter*. The second filtering step is performed using a *Recursive Feature Elimination* (RFE) [64] strategy, leveraging the feature importance score of the Random Forest model. The resulting features constitute the final set of *meta-features* that MALIAS will use to choose the best algorithm for the time series.

The *target metrics* represent the output of the Random Forest regressor integrated within MALIAS, namely the vector $\vec{m} = \langle m_1, m_2, \ldots, m_k \rangle$. MALIAS supports vectors of varying sizes depending on the number of k candidate algorithms available for the task. Additionally, the policy for recommending the best algorithm changes depending on the type of target

TABLE I
An illustrative subset of extracted feature types by name and description.

Feature type	Description
fft_coefficient	Coefficients of the one-dimensional discrete Fourier transform of the time series.
change_quantiles	Aggregated measure of changes in the time series within a corridor defined by lower and upper quantiles.
agg_linear_trend	Divide the time-series into equal chunks, compute an aggregate value for each chunk, then fit a linear regression to those chunk-averages against the chunk-index (0, 1, ...).
symmetry_looking	Boolean flag that indicates whether the distribution of the time-series "looks symmetric" by checking if the difference between its mean and median is small compared to its overall range.
large_standard_deviation	Boolean indicator of whether the standard deviation of the series is large relative to its range.
autocorrelation	Computes the correlation between the time series and its lagged (shifted) copy at a given lag.
energy_ratio_by_chunks	Ratio of the energy (sum of squares) contained in a chunk of the time series to the energy of the whole series, across several segments.
quantile	A selected quantile (e.g., 10th, 20th percentile) of the time series values, summarizing the distribution.
mean	The arithmetic average of the time series values, summarizing central tendency.
standard_deviation	The standard deviation of the time series values, summarizing dispersion around the mean.
	...

[1] https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

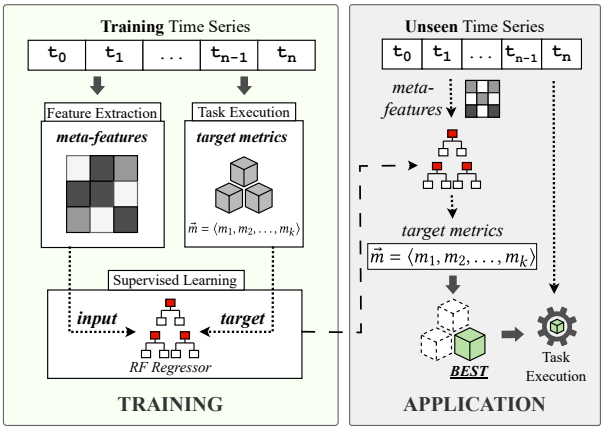


Fig. 3. MALIAS Overview.

metric (and corresponding task). For instance, in the case of TSF and error-based metrics, MALIAS selects the algorithm i that minimizes m_i. Conversely, for TSC/TSAD and accuracy-based metrics, MALIAS selects the algorithm i that maximizes m_i.

IV. Goal and Research Questions

The *goal* of the study is to investigate whether—and to what extent—meta-learning can support algorithm selection for time-series-based SE tasks. In other words, we want to understand whether learning the relationship between time series *meta-features* and algorithm *target metrics* can aid such selection. The *context* is represented by three distinct time series-based SE tasks, involving a total of six different datasets, and 14 time series analysis algorithms. The study addresses the following **research questions (RQs)**:

RQ$_1$ *Does MALIAS effectively select TSF algorithms that improve forecasting of software telemetry data?* We want to understand whether, and to what extent, MALIAS yields improvements over individual TSF algorithms in forecasting software telemetry data. If MALIAS does not outperform these baselines, its practical utility may be limited.

RQ$_2$ *Does MALIAS effectively select TSAD algorithms that improve anomaly detection in software systems?* We investigate whether, and to what extent, MALIAS outperforms individual TSAD algorithms in detecting software anomalies.

RQ$_3$ *Does MALIAS effectively select TSC algorithms that improve steady-state detection in software performance testing?* We examine whether, and to what extent, MALIAS provides better steady-state detection than individual TSC algorithms.

RQ$_4$ *To what extent does MALIAS generalize across datasets of different domains?* We investigate to what extent MALIAS generalizes to datasets from domains different from those on which it was trained. This reflects a scenario in which MALIAS is first pre-trained on data from one domain and then deployed in a zero-shot setting to automatically select the most suitable algorithm for a time series from a previously unseen domain.

RQ$_5$ *What is the impact of the meta-features on MALIAS effectiveness?* First, we present an ablation study to validate our feature selection strategy. Subsequently, we investigate the most important meta-features employed by MALIAS for algorithm selection and quantify their impact.

RQ$_6$ *How does MALIAS perform relative to other automated machine learning approaches?* We aim to assess how MALIAS performs in comparison with other similar approaches that automatically select or integrate algorithmic predictions. Specifically, we compare MALIAS against two established AutoML frameworks and three variants of MALIAS using different meta-learners.

In the subsequent sections, we first outline the study context (Section V), which describes in detail the three distinct time series-based SE tasks used to evaluate MALIAS, including the datasets, algorithms, and the evaluation metrics (*i.e.,* target metrics) used for each task[2]. We then present the study setup (Section VI), which defines the general procedure used to train and evaluate MALIAS, the metrics used to quantify its improvements over the baselines, and the methodologies employed to answer each RQ. Finally, we discuss the results (Section VII).

V. Study Context

We evaluate MALIAS across three SE tasks, each representing a distinct type of time series analysis: time series forecasting (TSF), time series anomaly detection (TSAD), and time series classification (TSC). Specifically, MALIAS is assessed on (i) forecasting of software telemetry data (TSF), (ii) anomaly detection in online software systems (TSAD), and (iii) steady-state detection in software performance testing (SSD), which is framed as a TSC task.

A. Time Series Forecasting (TSF)

Datasets. For this task, we employ two publicly available, real-world datasets of software telemetry data: (i) *Azure Functions Dataset (AFD)* [2] and (ii) *Web Service Dataset (WSD)* [1]. AFD contains time series data representing the number of Azure Functions[3] invocations recorded for a 14-day period, aggregated at one-minute intervals. We initially preprocessed the AFD time series to aggregate the data into ten-minutes intervals, thereby reducing the length of each time series to 2,016 data points and consequently decreasing the computational cost of the experiments. Additionally, we focus on a subset of 288 functions exhibiting elevated workload variability, identified by examining the distribution of AFD time series values. Additional technical details are included

[2]Note that each task involves a different set of datasets, evaluation metrics, and algorithms. For instance, a TSC algorithm cannot be used to forecast software telemetry data because it outputs a class label rather than a continuous numerical value.

[3]https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview.

in our replication package [39]. The second dataset (WSD) contains time series data that represent software telemetry data collected from three top-tier Internet companies providing large-scale web services (*e.g.,* Baidu, Sogou, and eBay). This dataset has been widely adopted in previous time series-related SE studies [15], [65], [66]. The WSD dataset also includes synthetic time series; however, we chose to exclude them from our evaluation, as our goal is to assess the TSF capability of MALIAS solely on real-world software telemetry data. As a result of this filtering step, we obtain a total of 210 time series. Additionally, to mitigate the computational cost of our experiments, we truncate each time series to the first 3,000 values. For clarity, we refer to this reduced version of the dataset as WSD_{3k}.

Algorithms. We consider six candidate algorithms for TSF to evaluate MALIAS, covering the three most common classes of TSF models: classical statistical methods, recurrent neural networks (RNNs), and time series foundation models [13]. As representative of classical statistical model, we select SARIMA [32], given its widespread popularity and adoption in recent studies [12], [13], [67], [68]. From the RNN class, we include : Fully-Connected RNN (FC-RNN) [69], *Long Short-Term Memory* (LSTM) [70] and *Gated Recurrent Unit* (GRU) [71]. They are specifically designed to address time dependencies among input data, making them suitable for tasks involving sequential data. Moreover, recently introduced pretrained foundation models have demonstrated impressive effectiveness and flexibility, with the possibility to be used as zero-shot forecasters. From this category, we select two state-of-the-art time series foundation models: *TimesFM* [72] and *Chronos* [42]. Summarizing, our model selection for TSF considers six distinct models: SARIMA, FC-RNN, LSTM, GRU, TimesFM, and Chronos.

Evaluation. For this task, MALIAS is evaluated in performing multi-step-ahead univariate time series forecasting. We analyze the effectiveness of each forecasting model using the rolling origin evaluation technique [50], where the testing portion of each time series is partitioned into smaller overlapping sliding windows (*i.e.,* forecasting segments), and each model is subsequently tasked with generating predictions for all such segments. Each time series is initially partitioned into training, validation, and testing subsequences, using an 80-10-10 split strategy. The training set (TS_{train}) is used to fit the model, the validation set (TS_{val}) is used to tune the parameters of the SARIMA model (and is ignored for the other models), and the testing set (TS_{test}) is used to evaluate model performance—specifically, to obtain the *target metrics*. The prediction horizon is set to $H = 50$ future time steps. The length of the input window (ℓ) varies according the specific category of considered model [12], [13]; the input window for RNN-based models is set to match the forecasting horizon ($\ell = 50$ in our setting), while the approaches that do not require any training, and instead perform zero-shot predictions (SARIMA, Chronos and TimesFM), use a maximum input sequence length of 512 time steps. Testing set windowing produces 102 50-length windows for AFD and 201 windows for WSD time series.

We evaluate each forecasting algorithm by adopting the Symmetrical Mean Absolute Percentage Error (SMAPE) [12], [50] on each testing set (TS_{test}). SMAPE values range from 0% to 200%, where 0% and 200% indicates perfect and worst forecasting accuracy, respectively. For each time series, SMAPE is calculated at each forecasting step h ($0 \leq h < H$), and the final score of each model in obtained by averaging over all H steps (mean SMAPE). Hence, this results in a mean SMAPE value for each time series and forecasting model, which together make up the *target metrics* of MALIAS.

B. Time Series Anomaly Detection (TSAD)

Datasets. We conduct TSDA experiments using two software time series datasets: (i) the *Web Service Dataset (WSD)*, which was also used in the TSF task, and (ii) *AIOPS* [3]. Both datasets consist of software time series with labeled data points annotated as anomalies. From the WSD dataset, we consider exclusively real-world software time series, excluding synthetic ones. We also exclude time series without anomalies, yielding a total of 161 WSD time series out of 510 initial time series. The resulting time series have a varying length ranging from 30,806 to 43,327 data points. The AIOPS dataset contains anomaly-labeled time series concerning software telemetry data, including a total of 29 time series collected from big companies (*e.g.,* Sougo, eBay, Baidu, and Tencent) each involving thousands of data points as well. Although the *AIOPS* dataset is composed of relatively small number of series, the length of each series included in *AIOPS* dataset ranges from 17,568 to 299,053 data points. In our view, this dataset represents an interesting complementary scenario, characterized by a small number of monitored components observed over an extended period, thereby yielding a large number of data points within each time series. For this reason, we decided to include it in our study. Both WSD and AIOPS datasets are widely employed in the TSAD SE literature [15], [73], [74]. Each time series of WSD and AIOPS is pre-partitioned into equal-sized 50-50% parts, one used for training (TS_{train}) and one for testing (TS_{test}).

Algorithms. We select five TSAD algorithms, each representing a distinct methodological family, as classified by Liu *et al.* [65]. We include LSTM-AD_α [75], a specific adaptation of LSTM-based model for performing the anomaly detection task on sequential data, and Autoregressive (AR) model [76], both categorized as *prediction-based* models. We also consider EncDec-AD [77], an LSTM-based encoder-decoder approach belonging to the *reconstruction-based* category, and the Frequency-enhanced Conditional Variational Autoencoder (FC-VAE) [78] model classified within the *VAE-based* family. Ultimately, we include FITS model [79], a lightweight $10k$-parameters model for time series analysis, categorized as a *general-purpose time series* model.

Evaluation. The experimental setting of the TSAD task is based on the standard procedure proposed in the *TimeSeries-Bench* [15] benchmark. The effectiveness of each TSAD algorithm is evaluated on each testing instance TS_{test} using the Area Under the Precision-Recall Curve (AUC-PR) [80], which serves as the *target metric* for this task. AUC-PR is

widely adopted in TSAD tasks within SE [15], particularly because it effectively handles the typically imbalanced nature of such datasets. It captures the trade-off between precision and recall across varying classification thresholds. The AUC-PR value ranges from 0 to 1, with higher values indicating better performance in identifying positive class instances–*i.e.,* anomalies. To account for inherent detection latency in real-world applications, we adopt the refined version of the AUC-PR metric by applying a 3-step delay adjustment provided by the benchmark framework [15]. This adjustment is a latency-aware evaluation strategy that considers an anomaly to be successfully detected if it is identified within three time steps following its occurrence.

C. Steady-State Detection (SSD)

Datasets. For the SSD task, we leverage two datasets from prior related studies: the dataset introduced by Barrett et al. [5], referred to as Virtual Machines Dataset (VMD), and the dataset employed by Traini et al. [4], referred to as Java Microbenchmark Dataset (JMD). To the best of our knowledge, these constitute the most comprehensive publicly available datasets for steady-state detection. VMD includes 3,660 time series collected from 46 microbenchmarks encompassing 8 distinct JIT-equipped Virtual Machines. The microbenchmarks cover a variety of programming languages, including Java, C, PHP and Python, and their executions are repeated over two different hardware configurations and two operating systems. Each time series is composed of 2,000 data points, and is annotated with the specific time step at which the steady-state is reached—marking the completion of the warm-up phase. We removed the time series extracted using PyPy, as the majority did not reach steady-state. JMD is the most extensive publicly available datasets of Java Microbenchmark Harness (JMH) performance measurements [37], containing measurements from 586 JMH microbenchmarks spread across 30 established open-source Java software systems. The dataset includes 10 time series for each microbenchmark, each one involving 3,000 consecutive microbenchmark iterations annotated with the specific iteration at which the steady state is attained.

We preprocess both VMD and JMD datasets to frame this task as a binary time series classification problem. Namely, each time series has been divided in smaller overlapping segments. Each segment is labeled as *unsteady* (negative class) if it contains at least one warm-up iteration; otherwise, it is marked as *steady* (positive class). In addition, we only included in the evaluation the time series that achieved steady state behavior, yielding a total number of 2,654 time series from VMD and 5,219 time series from JMD. To mitigate measurement redundancy across segments, we adopt a segmentation strategy with an adaptive step size [4]. Furthermore, we employ a customized resampling strategy designed to balance the number of steady and unsteady segments while maintaining a representative distribution across all microbenchmarks [4].

Algorithms. For the model selection, we reuse three state-of-the-art TSC algorithms that have been employed in a previous study for SSD [4], [27]: (i) Fully-Convolutional Network (FCN) [53], (ii) Omni-Scale Convolutional Neural Network (OSCNN) [81] and Rocket [82]. Both FCN and OSCNN are neural networks; the first one includes several stacked convolutional layers, while OSCNN is based on the Omni-Scale block introduced by Tang et al. [81]. Rocket combines convolutional kernels with a cross-validated Ridge Classifier [83] to perform the classification.

Evaluation. We perform the SSD experiments based on the procedure highlighted in prior work [4]. Specifically, the process combines an initial training phase, involving a supervised learning process over a set of training time series, and a subsequent inference stage over the testing set, employing a 5-fold cross-validation process. The splitting strategy accounts for data leakage prevention: for the JMD dataset, it ensures that the segments pertaining to the same benchmark are always located within the same fold, while for VMD, the time series related to the same VM are always located in the same fold. We evaluate the TSC algorithms using the F_β score, which serves as the *target metric* for SSD. We set the β value to 0.2 to prioritize precision over recall. This choice is motivated by the need to penalize more false positives than false negatives, as false positives have shown to have a more detrimental effect in the practical usage of SSD in performance testing [4].

VI. Study Setup

In this section, we detail the experimental procedure used for evaluating MALIAS, along with the adopted metrics and implementation details.

A. Experimental Procedure

We evaluate the effectiveness of MALIAS on the three time series-based tasks introduced in the previous section. For each task, we use two datasets and configure MALIAS to include all candidate algorithms outlined in Section V. Notably, the pool of candidate algorithms is fixed for each task. For each dataset, the procedure involves an initial training stage followed by the application stage. We emphasize that the operational procedure of MALIAS is consistent across all the datasets and tasks. In that, the time series within each dataset are partitioned into training and testing (*i.e.,* application) subsets. More specifically, the splitting strategy relies on a 5-fold cross-validation process, where each dataset is evenly partitioned into five disjoint subsets. In each iteration, four folds are used for training, while the remaining fold is used for testing the model. We iterate this process five times, ensuring that each fold is used exactly once for testing. This cross-validation setup eliminates bias toward a specific training subset by systematically varying the training data, ensuring that all samples are used for testing across different folds. Additionally, 20% of the time series are randomly selected within the four non-test folds to form a validation set. This validation set is employed for performing the feature selection and the hyperparameter tuning. Notably, the number of time series instances varies substantially across datasets—ranging from 29 (AIOPS) to 5,219 (JMD)— to allow evaluating the effectiveness of MALIAS under a wide range of data availability conditions.

Before starting the MALIAS training and evaluation process, we executed the k candidate algorithms for each of three tasks over their respective full datasets, to collect a k-length vector $\vec{m}$ of *target metrics* for each time series. Afterwards, we extract the *meta-features* for each time series instance.

In each cross-validation iteration, *target metrics* and *meta-features* from the training folds are used to train the Random Forest model, while the testing fold provides the data for MALIAS evaluation.

As described in Section V, both TSF and TSAD experiments involve partitioning each time series instance into a training portion (TS_{train}) and a testing portion (TS_{test}). In line with the practical use of MALIAS, we extract the time series *meta-features* exclusively from TS_{train}. In contrast, the *target metrics* (SMAPE or AUC-PR) represents the algorithms' performance on TS_{test}.

For SSD tasks, when evaluating a time series, the extraction of the *meta-features* is performed on another time series belonging to the same microbenchmark. This experimental design choice aims to mirror a realistic usage scenario of SSD, where a microbenchmark is first executed once to gather its time series *meta-features* and, subsequently, MALIAS leverages these *meta-features* to automatically select the best algorithm for all future executions of that microbenchmark. In contrast, the *target metrics* (F_β scores) represent the performance of algorithms on the segments derived from the evaluated time series, as described in Section V.

During the training stage, the supervised learning process of the Random Forest model (meta-learner) is structured as follows: for each time series, the *meta-features* serve as the input to the meta-learner, while the corresponding *target metrics* are normalized using a *min-max* strategy and concatenated to form the output vector $\vec{m}$, which serves as the target output of the meta-learner. In the application phase, the inference process follows a similar procedure: given a set of *meta-features* extracted from an unseen time series, the meta-learner predicts the corresponding vector $\vec{m}$ containing the *target metric* values for each algorithm. MALIAS then selects the algorithm with the most favorable predicted *target metric*. For TSAD and SSD, MALIAS selects the algorithm with the highest target metric values, as these correspond to accuracy-based scores (*i.e.,* AUC-PR and F_β). For TSF, MALIAS selects the algorithm with the lowest target metric values, since these correspond to error-based scores (*i.e.,* SMAPE).

Our experimental design prevents data leakage through strict separation of series used for training MALIAS and target test series. Specifically, each dataset is partitioned into three non-overlapping subsets: a training subset for the Random Forest fitting, a validation subset for driving the feature selection and hyperparameter tuning steps, and a test subset for evaluating MALIAS predictions. This partitioning is enforced across a 5-fold cross-validation procedure, ensuring that each time series serves as a test instance exactly once. Thus, feature selection and hyperparameter optimization steps are implemented within MALIAS and automatically executed in the same way across all the datasets based on the validation set performance metrics. As a direct consequence, applying the same process to different datasets yields different selected features and model hyperparameters. Our evaluation procedure rigorously reflects a real-world usage scenario in which a set of in-domain historical time series is sufficient to train and deploy MALIAS, which automatically performs feature extraction and selection as well as hyperparameter optimization.

Implementation details of our experimental procedure are provided in Appendix A.

B. Evaluation Metrics

We assess the effectiveness of MALIAS by focusing on two complementary aspects: (i) the frequency with which MALIAS yields improved or degraded performance compared to other time series analysis algorithms, by reporting the corresponding *Improvement Rates*, and (ii) the magnitude of performance deviation by comparing the *Target Metrics* and the *Oracle Performance Gap*. In the following sections, we provide a detailed explanation of each of these evaluation metrics.

Improvement Rates. The first measure concerns the frequency with which MALIAS achieves better performance than other time series analysis algorithms. In that, we compute the following evaluation metrics for each pairwise comparison:

- *Improvement (%)*: This measure represents the percentage of time series instances in which MALIAS outperforms the compared algorithm.
- *Regression (%)*: This measure represents the percentage of time series instances in which MALIAS provides worse performance than the compared algorithm.
- ***Net Improvement (pp)***: The arithmetic difference between the Improvement and the Regression percentages in percentage points (pp). This value reflects the net success frequency provided by MALIAS.

Target Metrics. We report summary statistics of the *target metric* values for our approach and other time series analysis algorithms. As detailed in Section V, we use the SMAPE score for TSF, the AUC-PR values for TSAD, and the F_β score for SSD. Specifically, we present the mean values of these metrics using bar plots, along with the .95 confidence intervals (CIs). To assess whether the observed differences between MALIAS and the other algorithms are statistically significant, we apply the Wilcoxon signed-rank pairwise test with Holm-Bonferroni correction. Statistical significance levels are annotated on the bar plots using the following notation: $p \leq 0.05$ (*), $p \leq 0.01$ (**), $p \leq 0.001$ (***) and $p \leq 0.0001$ (****).

Oracle Performance Gap (OPG). We also assess the relative performance gap of MALIAS compared to an oracle-based approach [84], which always selects the optimal algorithm for each time series instance. The concept of an oracle is commonly used in the evaluation of dynamic ensemble selection methods [85], [86]. If MALIAS perfectly selects the best algorithm for each time series instance, then it will match the oracle performance. It is worth noting that the oracle performance represents an upper bound for MALIAS in the case of TSAD and SSD tasks, and a lower bound in the case of TSF. We refer to this measure as the OPG score, which quantifies the relative performance loss with respect

to the oracle. Let S denote the performance achieved by MALIAS (*i.e.,* mean SMAPE for TSF, mean AUC-PR for TSAD, and mean F_β for SSD), and let S^* represent the ideal performance of the oracle. We define the OPG score as: $\frac{|S^*-S|}{S^*} \times 100$. This measure (where lower values are better) reflects how closely MALIAS approaches the optimal performance achievable within the given experimental setup.

C. Methodologies for the Research Questions

In the following, we describe the methodologies used to answer each research question.

RQ$_1$. To answer this RQ, we execute MALIAS following the cross-validation strategy outlined in Section VI-A. Specifically, at each iteration, we first perform feature selection and hyperparameter tuning using the training and validation folds. We then evaluate the predictions generated by MALIAS on the test fold using SMAPE. The feature selection procedure results in 39 *meta-features* for the AFD dataset and 20 for the WSD_{3k} dataset. After obtaining the final set of SMAPE values, we compute: (i) the *Improvement Rates* of MALIAS relative to the six considered TSF algorithms—SARIMA, FC-RNN, LSTM, GRU, TimesFM, and Chronos; (ii) the statistical significance of the differences in the *Target Metrics* using the Wilcoxon signed-rank test; and (iii) the *Oracle Performance Gap (OPG)*.

RQ$_2$. The evaluation process follows the same cross-validation strategy employed in RQ_1. The feature selection process leads to 9 *meta-features* for AIOPS, and 112 for WSD dataset. The anomaly detection capabilities of MALIAS are evaluated using AUC-PR. The evaluation is carried out by analyzing the improvement rates of MALIAS over five baseline TSAD algorithms—LSTM-AD_α, AR, EncDec-AD, FC-VAE, and FITS—along with the mean AUC-PR values and OPG scores.

RQ$_3$. To evaluate MALIAS for SSD, we once again adopt cross-validation, where the task performance is evaluated using F_β score. The feature selection step eventually yields 8 *meta-features* for JMD and 5 for VMD. We compare the performance of MALIAS with that of three SSD algorithms, namely Rocket, FCN and OSCNN. The final comparison is made by considering the improvement rates, mean F_β scores, and the OPG scores.

RQ$_4$. To answer this research question, we train MALIAS on the *meta-features* and *target metrics* of one dataset and evaluate its performance on another dataset related to the same task. Specifically, as our experimental setup includes two datasets per task, we first train MALIAS on one dataset and subsequently test it on the other. For instance, in the TSF task, we train MALIAS on the AFD dataset and evaluate it on WSD_{3k}, and vice-versa. We follow the same procedure for all three time series-based tasks: TSF, TSAD, and SSD. We refer to this variant of our approach as MALIAS $_{gen}$. After obtaining the SMAPE values (for TSF), AUC-PR values (for TSAD), and F_β scores (for SSD) of MALIAS $_{gen}$, we compute the *net improvements* compared to two baselines: (i) a standard instance of MALIAS (i.e., trained on the training set from the same dataset), and (ii) the best-performing baseline algorithm for the specific task and dataset. As in earlier research questions, net improvement is defined as the percentage difference between instances where MALIAS $_{gen}$ outperforms the baseline and those where it underperforms.

RQ$_5$. To address this research question, we first validate our feature selection strategy through an ablation study against four alternative approaches: (i) using the complete feature set, (ii) *k*-random feature selection, (iii) *k*-best based on the F-test for regression, and (iv) *k*-best based on Mutual Information Regression (MIR) [87], with *k* denoting the feature retention threshold set to the mean cardinality of feature subsets identified by our strategy.

Subsequently, we analyze the individual contributions of different meta-features. We use the Random Forest feature importance scores computed during training as a proxy for their impact on MALIAS algorithm selection. Given the large number of considered meta-features, we restrict the analysis to the most influential ones. We start by discussing the five topmost meta-features for each dataset, reporting the mean importance score (across the k folds) and their standard deviations. Subsequently, for each dataset, we isolate the most important meta-feature and analyze how MALIAS selection rate of the algorithms varies across its values. Finally, we visually report few exemplary instances to discuss the specifics of MALIAS in the software engineering context and clarify why it is more effective than the static selection.

RQ$_6$. To answer this RQ, we compare MALIAS against automated machine learning approaches. We selected the baselines based on two specific criteria: (i) the approaches should automatically leverage multiple algorithms to produce their predictions, and (ii) they should support all three time series-based tasks considered in our study, namely time series forecasting, time series anomaly detection, and time series classification, consistent with the capabilities of MALIAS. We therefore consider widely-used AutoML frameworks that support time series analysis, as they constitute a practical *plug-and-play* solution for real-world use. Specifically, we performed a systematic exploration of GitHub repositories, by querying with the keyword "*AutoML*" and limiting the results to the top 20 most-starred projects with more than 500 forks. Subsequently, we manually inspected the results and selected the suitable candidates. Our main inclusion criterion in the selection process was the native availability of dedicated APIs for manipulating time series data, as well as the capability of supporting all the three tasks that are considered in MALIAS evaluation. Frameworks were included only if actively maintained (*i.e.,* not archived). The complete list of AutoML frameworks we examined is available in the replication package. Following this screening process, two relevant open-source AutoML frameworks were identified as baselines: (i) *AutoGluon*[4] and (ii) *FLAML*[5]. These frameworks represent our best choice among the possibilities, considering our selection criteria. They both have been also employed in previous works concerning time series analysis [88], [89].

[4]https://auto.gluon.ai/stable/index.html
[5]https://microsoft.github.io/FLAML/

TABLE II
IMPROVEMENT RATES OF MALIAS OVER BASELINE TSF ALGORITHMS

MALIAS *vs.* Model	Azure Functions Dataset (AFD) Improvement (%)	Regression (%)	***Net Improvement (pp)***	Web Service Dataset (WSD_{3k}) Improvement (%)	Regression (%)	***Net Improvement (pp)***
MALIAS *vs. Chronos*	40.3%	22.6%	**+17.7**	14.3%	4.8%	**+9.5**
MALIAS *vs. TimesFM*	44.4%	23.6%	**+20.8**	68.6%	25.7%	**+42.9**
MALIAS *vs. GRU*	51.4%	28.8%	**+22.6**	78.1%	18.6%	**+59.5**
MALIAS *vs. RNN*	60.1%	31.6%	**+28.5**	79.0%	18.6%	**+60.5**
MALIAS *vs. LSTM*	64.9%	33.0%	**+31.9**	76.7%	18.1%	**+58.6**
MALIAS *vs. SARIMA*	81.6%	17.7%	**+63.9**	78.6%	19.0%	**+59.5**

AutoGluon uses models bagging and stack ensembling, leveraging the predictions of each individual algorithm as extra features in the fitting stage and, at prediction time, it averages the predictions of all these individual models to generate the final prediction scores. FLAML selects a set of candidate learning algorithms and defines a corresponding hyperparameter search space for each one. It then performs training and validation across these candidates and selects the model that achieves the best evaluation score for prediction. Although FLAML supports stacked ensembling with multiple estimators, this option is disabled in its default configuration. Based on the target task, the AutoML frameworks select the appropriate data preprocessing and validation strategies, such as stratified sampling for classification and temporal splitting for forecasting. Complete experimental pipelines of both AutoGluon and FLAML, including their configuration and runtime computation, are provided in Appendix B.

As a complementary analysis, we expanded the comparison by including additional baselines that more closely reflect the behavior of MALIAS. Specifically, we introduced three variants of MALIAS that use different meta-learning algorithms in place of the Random Forest model: (i) Decision Tree ($MALIAS_{DT}$), (ii) K-Nearest Neighbors ($MALIAS_{KNN}$), and (iii) Linear Regression ($MALIAS_{LR}$). The feature selection procedure has been aligned with the methodology used for MALIAS. Hyperparameters have been tuned through a GridSearch and based on the validation score. Complete hyperparameters search space for each baseline is reported in our replication package [39].

We compare MALIAS with all the baselines using one representative dataset per task. Specifically, we employed the Azure Functions Dataset (AFD) for TSF, the AIOPS dataset for TSAD and Virtual Machines Dataset (VMD) for SSD. As for the previous research questions, we report, for each approach, the improvement rates along with the distribution of the metric scores for each task: SMAPE for TSF, AUC-PR for TSAD, and F_β for SSD. For the comparison with AutoML frameworks, we also analyze the time spent by each approach in carrying out the training and testing phases for each time series.

VII. RESULTS

In this section, we present our research questions, describe the methodology used to address them, and discuss the results.

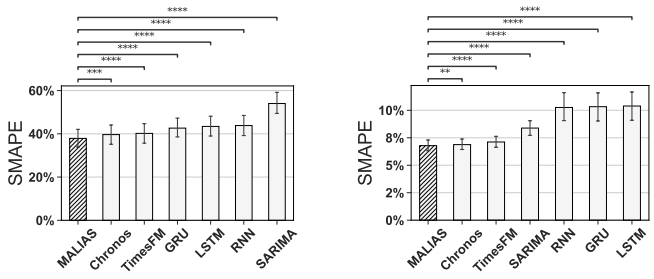
A. RQ_1: Does MALIAS effectively select TSF algorithms that improve forecasting of software telemetry data?

Table II shows the improvement rates of our approach relative to baseline TSF algorithms. The results clearly demonstrate the advantage of adopting MALIAS for TSF of software telemetry data. MALIAS improves SMAPE across a significant portion of the evaluated time series instances, with improvements observed in 40.3% to 81.6% of instances for AFD, and 14.3% to 79% of instances for WSD_{3k}. However, it also leads to regressions in up to 33% of instances for AFD and up to 25.7% of instances for WSD_{3k}. Nevertheless, MALIAS ultimately achieves a substantial net improvement over all baseline algorithms, ranging from +17.7 percentage points (*pp*) to +63.9 *pp* in AFD, and from +9.5 *pp* to +60.5 *pp* in WSD_{3k}.

TABLE III
MALIAS SELECTION RATE IN TSF DATASETS.

Dataset	#TS	Selection Rate (%) *Chronos*	*TimesFM*	*GRU*	*RNN*	*LSTM*	*SARIMA*
AFD	288	**37.2** %	<u>31.9</u> %	19.8 %	8.3 %	2.1 %	0.7 %
WSD_{3k}	210	**81.0** %	<u>5.7</u> %	3.3 %	2.4 %	5.2 %	2.4 %

The highest is reported in **bold**, the second-highest is <u>underlined</u>.



(a) Azure Functions Dataset (AFD). (b) Web Service Dataset (WSD_{3k}).

Fig. 4. Mean SMAPE for MALIAS and TSF algorithms; annotations indicate Wilcoxon signed-rank test results.

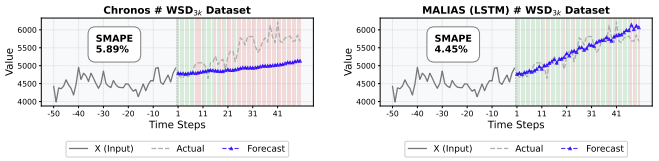
Figure 4 reports the mean SMAPE of MALIAS and TSF algorithms, along with their 0.95 confidence intervals. We observe that MALIAS achieves the lowest mean errors on both datasets, reporting a mean SMAPE of 37.89% for AFD and 6.78% for WSD_{3k}. The results of the Wilcoxon signed-rank test highlight a very high statistical significance in the differences between the SMAPE values produced by MALIAS and those by other TSF algorithms, with $p < 0.001$ across all MALIAS-baseline comparisons. This result indicates a

consistent improvement over all baseline TSF algorithms on both datasets.

To better understand these improvements, we analyze the algorithm selections made by MALIAS. Table III presents the selection rates of algorithms chosen by MALIAS across both TSF datasets. We observe that the distribution of selection rates differs significantly between the two datasets. For the AFD dataset, the selections are more diverse: Chronos is the most frequently selected model (37.2%), followed by TimesFM (31.9%) and GRU (19.8%). In contrast, the WSD_{3k} dataset exhibits a more skewed selection pattern, with Chronos being selected 81% of the times. All other models are chosen less than 10% of the times. Interestingly, even though MALIAS predominantly selects Chronos, it still achieves an overall improvement of +9.5 *pp* over Chronos (as shown in Table II).

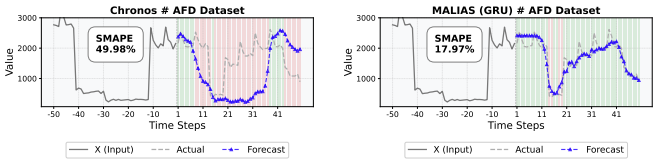
We also observe an alignment between the algorithm selection rates by MALIAS and the distribution of best-performing models depicted in Figure 1. For instance, the top-3 best-performing TSF models on the AFD dataset match the three most frequently selected algorithms by MALIAS. Similarly, for WSD_{3k}, the best-performing model, Chronos, is also the one most frequently selected by MALIAS. However, we also notice a significant mismatch between these percentages. For example, MALIAS selects Chronos 81% of the time, even though it is the best-performing model in just 39% of cases. This discrepancy potentially explains the OPG values of 5.05% for WSD_{3k} and 16.86% for AFD shown in Table IV. These results highlight that MALIAS has not reached oracle-level performance, suggesting there remains room for improvement.

To illustrate why MALIAS selection improves forecasting accuracy, we present two representative cases comparing the forecasts generated by both MALIAS and Chronos, as the latter represents the best-performing TSF algorithm on average. We isolate a single forecasting window in each plot for a more interpretable visualization. Figure 5 illustrates an example from WSD_{3k} in which MALIAS automatically selects LSTM as the most suitable TSF algorithm. As shown in the figure, the predictions produced by LSTM better capture the evolution of the series than those produced by Chronos, reducing the SMAPE from 5.89% to 4.45%. Similarly, Figure 6 presents an example from the AFD dataset in which the model selected by MALIAS (GRU) accurately captures the underlying trend, achieving a substantial SMAPE reduction from 49.98% to 17.97% compared with Chronos. Unlike a traditional approach based on a single algorithm, which would likely use Chronos because of its superior average performance, MALIAS automatically selects the model best suited to each individual time series, even when that model performs suboptimally on average across the dataset. These examples also evidence that, although zero-shot foundation models appear highly promising for forecasting, as also evidenced in previous studies [13], simpler models with domain-specific training procedures still perform better in certain settings.



(a) Chronos Forecasts (b) MALIAS Forecasts

Fig. 5. Visual comparison of Chronos vs MALIAS (choosing LSTM) for generating forecasts (Example of Window #1 from WSD_{3k} dataset). Red background indicates forecasts where the percentage absolute error is higher than 5%. Instance ID=`83`.



(a) Chronos Forecasts (b) MALIAS Forecasts

Fig. 6. Visual comparison of Chronos vs MALIAS (choosing GRU) for generating forecasts (Example of Window #30 from AFD dataset). Red background indicates forecasts where the percentage absolute error is higher than 30%. Instance ID=`5c063c...5250`.

Answer to RQ_1: MALIAS algorithm selection consistently improves the performance of individual TSF algorithms, with very high statistical significance ($p < 0.001$). MALIAS yields a net improvements of +17.7 *pp* and +9.5 *pp* when compared to the best-performing TSF baseline.

B. RQ_2: Does MALIAS effectively select TSAD algorithms that improve anomaly detection in software systems?

Improvement rates for TSAD are presented in Table V. We observe that the percentages of improvements are always higher than regressions. Specifically, for the AIOPS dataset, MALIAS delivers a net improvement ranging from +20.7 *pp* (*vs.* FCVAE) to +93.1 *pp* (*vs.* FITS). The percentages of regressions, which range from 3.4% to 27.6%, are indeed considerably lower than those of improvements, which range from 34.5% to 96.6%. In the WSD dataset, we observe improvements ranging from 23% to 59% and regressions from 19.9% to 34.8%. As a result, MALIAS achieves net improvements between +3.1 *pp* and +31.7 *pp* on WSD instances.

TABLE IV
OPG OF MALIAS ACROSS TASKS.

Task	Target Metric	*MALIAS OPG (%)*	
TSF	SMAPE	*AFD* 16.86 %	*WSD_{3k}* 5.05 %
TSAD	AUC-PR	*AIOPS* 5.15 %	*WSD* 7.86 %
SSD	F_β	*VMD* 2.63 %	*JMD* 3.24 %

TABLE V
IMPROVEMENT RATES OF MALIAS OVER BASELINE TSAD ALGORITHMS

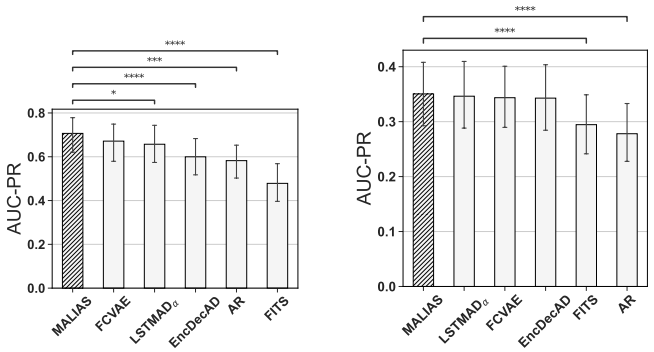
MALIAS *vs.* Model	AIOPS Dataset			Web Service Dataset (WSD)		
	Improvement (%)	Regression (%)	***Net Improvement (pp)***	Improvement (%)	Regression (%)	***Net Improvement (pp)***
MALIAS *vs. LSTMAD*$_\alpha$	34.5%	3.4%	**+31.0**	23.0%	19.9%	**+3.1**
MALIAS *vs. FCVAE*	48.3%	27.6%	**+20.7**	41.0%	31.7%	**+9.3**
MALIAS *vs. AR*	75.9%	10.3%	**+65.5**	57.1%	25.5%	**+31.7**
MALIAS *vs. EncDecAD*	86.2%	13.8%	**+72.4**	45.3%	34.8%	**+10.6**
MALIAS *vs. FITS*	96.6%	3.4%	**+93.1**	59.6%	28.0%	**+31.7**

TABLE VI
MALIAS SELECTION RATE IN TSAD DATASETS.

Dataset	#TS	Selection Rate (%)				
		LSTM-AD$_\alpha$	*FCVAE*	*AR*	*EncDecAD*	*FITS*
AIOPS	29	**62.1%**	<u>24.1%</u>	13.8%	0%	0%
WSD	161	**52.8%**	<u>22.4%</u>	11.8%	8.7%	4.3%

The highest is reported in **bold**, the second-highest is <u>underlined</u>.

As shown in Table VI, the model most frequently selected by MALIAS is LSTM-AD$_\alpha$, although the FCVAE model is also chosen in over 20% of the instances in both datasets. Notably, this task involves the smallest number of time series—190 in total. Despite the limited size of the training and validation datasets, MALIAS consistently produced effective algorithm recommendations, outperforming all baseline TSAD models. One possible explanation for this result is that the large number of data points within each time series (up to 299k in this case) enables the extraction of statistically robust and informative features, partially compensating for the limited number of training instances.



(a) AIOPS Dataset (b) Web Service Dataset (WSD)

Fig. 7. Mean AUC-PR for MALIAS and TSAD algorithms; annotations indicate Wilcoxon signed-rank test results.

Figure 7 presents the mean AUC-PR scores for MALIAS and the baseline TSAD algorithms. On the AIOPS dataset, MALIAS achieves the highest mean AUC-PR score (0.71). Among the baseline algorithms, the most accurate for AIOPS is FCVAE, with a mean AUC-PR of 0.67. MALIAS outperforms all the TSAD baselines with statistical significance ($p < 0.05$), with the sole exception of FCVAE. On the WSD dataset, MALIAS also achieves the highest AUC-PR (0.350), although the performance differences with the baselines are less pronounced. For example, the best-performing TSAD baseline, LSTM-AD$_\alpha$, achieves an AUC-PR of 0.346. Statistically significant improvements are observed only over FITS and AR ($p < 0.0001$). Table IV presents the OPG score achieved by MALIAS, showing a gap of 5.15% in AIOPS instances and a gap of 7.86% for WSD.

Answer to RQ$_2$: MALIAS algorithm selection consistently improves the performance of individual TSAD algorithms, delivering a net improvement of +20.7 *pp* for AIOPS and +3.1 *pp* for WSD against the best-performing baseline.

C. RQ$_3$: Does MALIAS effectively select TSC algorithms that improve steady-state detection in software performance testing?

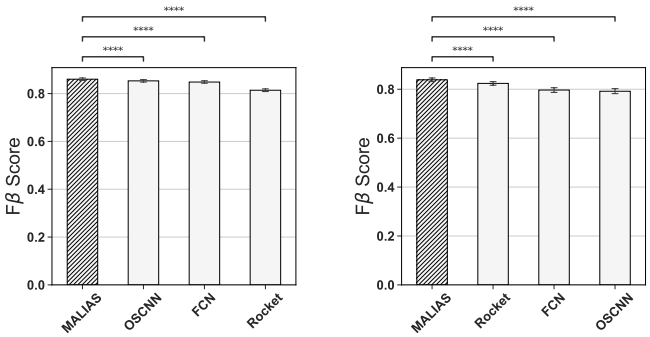
Table VII shows the improvement rates achieved by MALIAS compared to each baseline SSD algorithm. In both the VMD and JMD scenarios, MALIAS consistently delivers higher improvement percentages than regressions. Specifically, for the VMD dataset, improvements range from 24.9% to 61.3% of the time series instances, whereas regressions are observed in only 15% to 19.5% of instances. This results in net improvements ranging from +9.8 *pp* to +41.8 *pp*. Similarly, for the JMD dataset, improvements span from 33.2% to 59.2%, with net improvements between +19.8 *pp* and +28 *pp*. These findings highlight the consistent effectiveness of MALIAS for the SSD task.

Figure 8 reports the mean F$_\beta$ score for each approach in JMD and VMD datasets. We notice that MALIAS always achieves the best mean F$_\beta$ score (0.860 for JMD and 0.839 for VMD) and that it outperforms all the other baseline SSD algorithms with very high statistical significance ($p < 0.0001$). For Table IV, we also find that MALIAS nearly approaches oracle-level performance with OPG of only 2.63% for VMD and 3.23% for JMD.

Concerning the selection rates reported in Table VIII, we observe that Rocket is the most frequently selected model for the VMD dataset (58.1%). This aligns well with the results shown in Figure 1, where Rocket emerges as the best-performing SSD algorithm in 49% of the time series instances. Interestingly, for the JMD dataset, we notice a different trend: OSCNN becomes the most frequently chosen model by MALIAS (53.7%), despite Rocket remaining the most frequent best-performing model on JMD, as indicated in Figure 1. This discrepancy can be attributed to the learning process being guided by the target metric values rather than by success frequency. Nonetheless, even in the JMD dataset, MALIAS still provides a substantial net improvement over Rocket (+19.8 *pp*), as well as better F$_\beta$ scores, with very high statistical significance ($p < 0.0001$).

TABLE VII
IMPROVEMENT RATES OF MALIAS OVER BASELINE SSD ALGORITHMS.

MALIAS *vs.* Model	Virtual Machines Dataset (VMD) Improvement (%)	Regression (%)	***Net Improvement (pp)***	Java Microbenchmark Dataset (JMD) Improvement (%)	Regression (%)	***Net Improvement (pp)***
MALIAS *vs. Rocket*	24.9%	15.0%	**+9.8**	39.4%	19.6%	**+19.8**
MALIAS *vs. OSCNN*	49.5%	19.3%	**+30.2**	33.2%	12.6%	**+20.6**
MALIAS *vs. FCN*	61.3%	19.5%	**+41.8**	59.2%	31.2%	**+28.0**



(a) Java Microbenchmark Dataset (JMD)

(b) Virtual Machines Dataset (VMD)

Fig. 8. Mean F_β scores for MALIAS and SSD algorithms; annotations indicate Wilcoxon signed-rank test results.

TABLE VIII
MALIAS SELECTION RATE IN SSD DATASETS.

Dataset	#TS	Selection Rate (%) *Rocket*	*OSCNN*	*FCN*
JMD	5,219	<u>40.1</u> %	**53.7** %	6.2 %
VMD	2,654	**58.1** %	<u>26.9</u> %	15.0 %

The highest is reported in **bold**, the second-highest is <u>underlined</u>.

Answer to RQ$_3$: MALIAS algorithm selection consistently improves the performance of individual SSD algorithms with very high statistical significance ($p < 0.0001$), delivering net improvements of +9.8 *pp* (VMD) and +19.8 *pp* (JMD) when compared to the best-performing baseline.

D. RQ$_4$: To what extent does MALIAS generalize across datasets of different domains?

TABLE IX
NET IMPROVEMENTS OF MALIAS $_{gen}$.

Task	**Testing Dataset**	**Training Dataset**	*MALIAS $_{gen}$ vs. best-performing model (pp)*	*MALIAS $_{gen}$ vs. MALIAS (pp)*
TSF	WSD_{3k}	AFD	-6.2 (*vs. Chronos*)	-15.7
TSF	AFD	WSD_{3k}	-0.7 (*vs. Chronos*)	-18.1
TSAD	AIOPS	WSD	-10.3 (*vs. FCVAE*)	-44.8
TSAD	WSD	AIOPS	-6.8 (*vs. LSTMAD$_\alpha$*)	-8.1
SSD	JMD	VMD	+0.8 (*vs. Rocket*)	-19.6
SSD	VMD	JMD	-3.8 (*vs. Rocket*)	-13.8

Table IX reports the net improvements of MALIAS $_{gen}$ compared to both a standard MALIAS instance and the best-performing baseline algorithm. When compared to the standard MALIAS instance, we observe that MALIAS $_{gen}$ consistently yields negative net improvements. This indicates that the proportion of time series instances where MALIAS $_{gen}$ improves MALIAS performance is lower than those where it worsens it. Specifically, net improvements range from -8.1 to -44.8 *pp*. This suggests that the relationship between *meta-features* and *target metrics* is closely tied to the characteristics of a specific dataset. Compared against the best-performing baseline algorithm, we observe slightly better net improvements, though they remain predominantly negative. The only exception is the JMD dataset, where MALIAS $_{gen}$ yields a marginal net improvement of +0.8 *pp*. In all other cases, MALIAS $_{gen}$ results in negative net improvements ranging from -0.7 to -10.3 *pp*. While these results highlight a limitation in the ability of MALIAS to transfer the learned knowledge across different datasets, this does not affect its effectiveness once an initial training phase has been performed using in-domain software time series, as observed in the other RQs. We consider this lack of generalizability across domains an important finding, as it suggests that the domain-specific characteristics heavily influences the learning phase.

Answer to RQ$_4$: The prediction capabilities of MALIAS do not generalize across datasets from different domains. MALIAS $_{gen}$ exhibits lower performance compared to both the standard MALIAS instance and the best-performing baseline.

E. RQ$_5$: What is the impact of the meta-features on MALIAS effectiveness?

Table X reports the net improvement differences obtained comparing our feature selection with the four alternative strategies. We set the retention threshold $k = 30$ for the *k-** alternative strategies to approximate the average number of meta-features retained by our selection strategy. Notably, negative values indicate an accuracy degradation compared with our feature selection strategy. Results exhibit a mixed pattern of gains and losses depending on the dataset and selection strategy. Overall, all alternatives introduce a net performance degradation, ranging from -0.1 to -19.2 *pp*, suggesting that the proposed feature selection strategy—composed of *ρ-filter* and RFE—leads to superior aggregate performance across all evaluated scenarios. Our feature selection strategy also yields substantial speedup in the application cost of MALIAS; for example, when executing MALIAS on the JMD dataset (SSD) using the optimal hyperparameter configuration, applying our feature selection leads to a 96% reduction in average mean training time per fold and a 24% reduction in average testing time.

TABLE X
ABLATION STUDY USING DIFFERENT FEATURE SELECTION PROCEDURES FOR MALIAS. REPORTED VALUES REPRESENT NET IMPROVEMENTS OVER OUR FEATURE SELECTION STRATEGY; DIFFERENCES < 0.1 ARE DENOTED BY "–".

	Net Improvement Diff.						
	TSAD		SSD		TSF		
	AIOPS	WSD	VMD	JMD	AFD	WSD_{3k}	***Overall***
Complete Set	-11.0 ⇓	0.8	0.4	0.4	-4.0 ⇓	-	**-13.5** ⇓
k-random	-11.1 ⇓	1.1	-	-3.1 ⇓	-3.6 ⇓	-2.6 ⇓	**-19.2** ⇓
k-best$_{Fregr}$	-	3.5	0.3	-5.8 ⇓	-3.0 ⇓	-2.4 ⇓	**-7.4** ⇓
k-best$_{MIR}$	-1.4 ⇓	7.5	0.1	0.3	-1.9 ⇓	-4.8 ⇓	**-0.1** ⇓

Next, we discuss the most influential meta-features affecting MALIAS selection. Figure 10 reports the mean importance scores–across the k folds–of the five topmost meta-features, along with their standard deviation, for each dataset across the three time series-based tasks. Overall, we notice that the meta-features vary across datasets. For instance, Figure 10a shows that AFD dataset is impacted by Augmented Dickey–Fuller (ADF) statistics, which are not present in the top 5 meta-features of WSD_{3K}. Similar patterns can be observed in Figures 10b and 10c. This heterogeneity becomes even more pronounced when examining all the datasets without regard to task. For example, the `Longest Strike Above Mean` meta-feature appears exclusively in AIOPS dataset. These results suggest that MALIAS is not able to find a universal rule that consistently holds across datasets within each task, thereby supporting and explaining the observed limitations in generalization reported in RQ_4. Although no task exhibits identical top five meta-feature sets across the two considered datasets, some shared characteristics can nevertheless be observed. For example, the `Longest Strike Below Mean` meta-feature heavily impacts both AFD and WSD_{3K}. Figure 9 reports the selection rates of the SSD algorithms

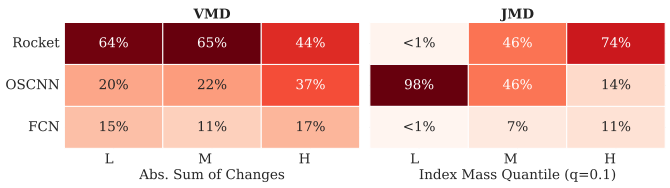


Fig. 9. Heatmaps showing how the value distribution of the top-ranked feature relates to its selection rate across algorithms for each SSD dataset. Colors intensity reflects the percentage values reported in the map.

while varying the values of the most important meta-feature according to the importance score of the Random Forest. Notably, such meta-feature is different for each dataset and it is automatically detected by MALIAS during the training process. As meta-feature values may follow markedly different distributions, we discretize each meta-feature distribution in the dataset into three ranges—L (low), M (medium), and H (high)—by partitioning it into three equal-frequency quantile bins. We observe that, based on the meta-feature values, MALIAS distributes the selection across models differently. For instance, for VMD, the results suggest that Rocket is more suitable than all the others when the `Abs. Sum of Changes` assumes low (L) and medium (M) values. For JMD, we observe that OSCNN is predominantly selected when `Index Mass Quantile (q=0.1)` is low (L), while for high values (H) the selection rate of Rocket is considerably the highest.

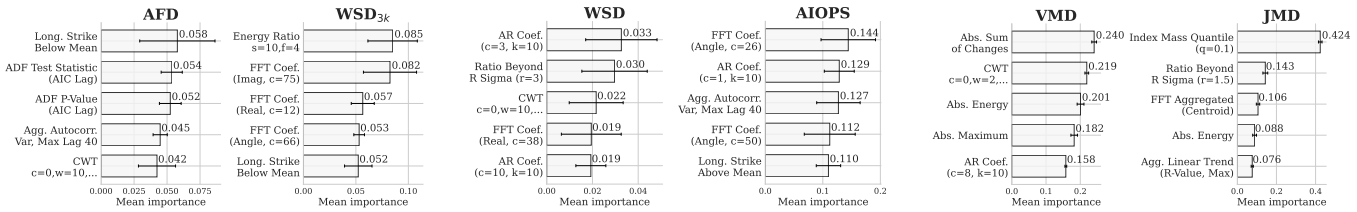
For completeness, our replication package [39] reports selection rates for all the other datasets considered in this study.

By examining the impact of MALIAS meta-features, we can also discuss its specifics in SE context while clarifying why it is more effective than a fixed-algorithm strategy. For TSF, Figure 10a illustrates that the `Longest Strike Below Mean`—defined as the longest below-mean subsequence—heavily impacts the selection on both AFD and WSD_{3K}, suggesting that it is a relevant characteristics to properly decide which algorithm to use. Longer below-mean subsequences may indicate substantial changes in the series behavior, a common phenomenon in runtime metric fluctuations of real-world software systems under constantly varying load [31].

As highlighted in Figure 9, in the context of SSD, the most important meta-features are `Absolute Sum of Changes` (ASC), for VMD, and `Index Mass Quantile (q=0.1)` (IMQ), for JMD (see Figure 9). ASC represents the sum over the absolute value of consecutive changes in the series. High warm-up spikes in performance tests may increase cumulative changes over time due to measurement instability [37]. Figure 11 illustrates two representative practical examples from the VMD dataset: *N-body* exhibits fewer warm-up iterations but higher spikes, resulting in larger sum of changes (5.81), whereas the *Fannkuch Redux* benchmark shows a longer warm-up phase with lower peaks, resulting in a total sum of changes of 0.70. Notably, MALIAS correctly selects the best model—*i.e.,* OSCNN for *N-body* and Rocket for *Fannkuch Redux*—in these cases, in line with the selection trend shown in Figure 9.

Concerning JMD, IMQ is mainly used for driving the selection (Figure 9). IMQ is defined as the (normalized) position i in a time series where a given percentage (in this case 10%) of the total data lies to the left. Intuitively, this meta-feature is expected to capture differences among warm-up phases characterized by distinct fluctuation patterns. To evidence this concern, we report two distinctive examples from JMD dataset in Figure 12, where warm-up behaviors are very different. *Log4j2* execution clearly shows higher execution times recorded after the warm-up phase, leading to a IMQ of 0.46. Instead, *JGraphT* reports a relatively stable trend with intermittent spikes distributed throughout the entire series. In this case, the IMQ is 0.1, corresponding to the first 10% of the series. MALIAS succesfully selects the best model in both the cases, consistently with the rates reported in Figure 9.

These examples emphasize how MALIAS specifically fits the SE context, cleverly leveraging the series meta-features based on the domain and the task. Notably, we did not inject any explicit information related to the specific task in addition to the historical series and target metrics. All the knowledge is directly learned from the target metrics (*e.g.,* binary labels for SSD) provided during the training phase, which inherently encode the objective of the task in the specific SE context.



(a) Time Series Forecasting (b) Time Series Anomaly Detection (c) Steady State Detection

Fig. 10. Top-5 meta-features importance scores computed by MALIAS across folds for each dataset, with bar length representing the mean (μ) and error whiskers denoting the standard deviation (σ).

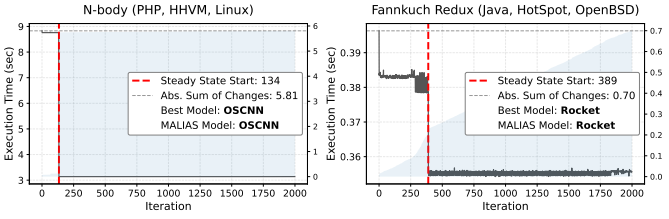


Fig. 11. ASC values across different warm-up behaviors. In the filled area, we report how the step-by-step changes cumulatively increase along the series. The red dashed line denotes the ground-truth steady-state index.

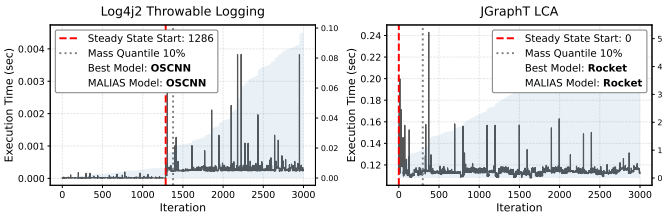
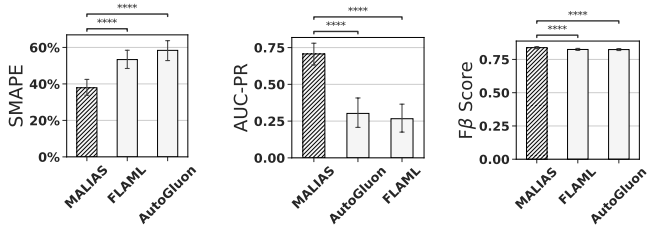


Fig. 12. IMQ values across different warm-up behaviors. In the filled area, we report how the step-by-step changes cumulatively increase along the series. The red dashed line denotes the ground-truth steady-state index.

Answer to RQ$_5$: The most influential meta-features affecting MALIAS selection largely vary across datasets and time series-based tasks, motivating the observed limitation in generalization reported in RQ$_4$. By examining how MALIAS distributes the model selection based on the meta-feature values in the SSD task, we found some interesting connections between the most important meta-features and the specific software behavior.

F. RQ$_6$: How does MALIAS perform relative to other automated machine learning approaches?

Table XI reports the improvement rates achieved by MALIAS in comparison with the AutoML baselines. When performing the TSF task, MALIAS improved by +73.6*pp* over AutoGluon and +66 pp over FLAML. In the TSAD task, MALIAS reported all wins over the 29 timeseries in the AIOPS dataset compared with AutoGluons (+100*pp*), and a net improvement of +93.1*pp* compared with FLAML. In the SSD task, the net improvements achieved by MALIAS are smaller but consistently positive, with gains of +16.8*pp* compared with AutoGluon and +11.3*pp* with FLAML on the VMD dataset. Notably, the SSD experiment reveals that the number of regressions—defined as the time series where MALIAS



(a) TSF - AFD (b) TSAD - AIOps (c) SSD - VMD

Fig. 13. Mean scores for MALIAS and AutoML frameworks; annotations indicate Wilcoxon signed-rank test results.

TABLE XI
COMPARISON BETWEEN MALIAS AND AUTOML FRAMEWORKS.

MALIAS *vs.*	Improvement (%)	Regression (%)	***Net Improvement (pp)***
	TSF - Azure Functions Dataset (AFD)		
AutoGluon	86.8%	13.2%	**+73.6**
FLAML	83%	17%	**+66**
	TSAD - AIOPS Dataset		
AutoGluon	100%	0%	**+100**
FLAML	96.6%	3.4%	**+93.1**
	SSD - Virtual Machines Dataset (VMD)		
AutoGluon	51.1%	34.2%	**+16.8**
FLAML	47%	35.7%	**+11.3**

underperforms compared with the baselines—exceeds 30% of the dataset, which represents a substantially higher rate with respect to the other tasks. Overall, MALIAS outperformed both AutoML baselines across all the three tasks. Figure 13 presents bar plots with the distribution of performance metrics for each task. In TSF and TSAD tasks, it is evident that the baseline scores are considerably lower than the ones obtained with MALIAS. However, in the SSD task, all the three approaches achieved a mean F_β score over 80%, thus suggesting similar performance. Nonetheless, as annotated in the same figure, the Wilcoxon test returned very high statistical significance ($p < 0.0001$) among all the comparisons.

The principal differences between MALIAS and the AutoML baselines lie in their training procedures and algorithm selection strategies. In this setting, MALIAS is trained over a pool of time series drawn from the same domain on which it is subsequently evaluated (*i.e.,* same dataset), whereas the AutoML baselines select the optimal model solely based on the time series under analysis. Additionally, MALIAS uses time series characteristics and metric values for carrying out the selection, while the AutoML frameworks primarily rely on

multiple algorithm predictions for this purpose. The findings suggest that utilizing cross-series information within the same domain is an effective approach to guide algorithm selection, and that this process can be effectively driven in light of the characteristics of the target series.

TABLE XII
COMPARISON OF TRAINING AND TESTING TIMES BETWEEN MALIAS AND AUTOML.

Task	Training time			Testing time		
	MALIAS	**AutoGluon**	**FLAML**	**MALIAS**	**AutoGluon**	**FLAML**
TSF	**158**	391.42	200.58	11.65	**4.16**	29.26
TSAD	1,733	**720.76**	1,791.11	39.49	2.11	**1.81**
SSD	34	**1.19**	38.07	**0.05**	0.27	0.09

Times are reported in seconds, averaged per time series.

Table XII reports training and testing times for MALIAS, AutoGluon and FLAML. In the TSF task, the results indicate that MALIAS attains a faster training time, averaging 158 seconds per time series, corresponding to a 59% decrease relative to AutoGluon (391 s) and a 21% decrease relative to FLAML (201 s). In carrying out TSAD, AutoGluon achieved the fastest training time, with an average of 721 seconds per time series. In percentages, this reduces by 58% the MALIAS training time (1,733 s) and by 60% the FLAML training time (1,791 s). For SSD, AutoGluon remains the fastest option with an average training time per time series of 1.19 seconds, decreasing by 95% the time required by MALIAS (34 s) and by 97% the training time required by FLAML (38 s). Overall, FLAML reports training times very close to those of MALIAS, likely due to the time budget configuration, which was derived from MALIAS average training duration. Table XII also reports the testing times. Across the evaluated tasks, all the approaches exhibit heterogeneous testing performance. In the TSF task, it records a runtime of 12 seconds, higher than AutoGluon (4 s) but considerably lower than FLAML (29 s). For TSAD, MALIAS required 39 seconds per time series, which is longer than both baselines; however, in the SSD task, it achieved the lowest inference time (0.05 s), outperforming both AutoGluon (0.27 s) and FLAML (0.09 s). These results suggest that the efficiency of each approach is task-dependent, and that the runtime behavior is influenced by the characteristics of the underlying dataset or task configuration, rather than by a consistent computational advantage. In summary, during the training stages MALIAS proved to be the most efficient for the TSF task, while AutoGluon reported significantly lower times in TSAD and SSD scenarios. In the testing phase, each approach appeared to be the most efficient in at least one specific task. However, the net improvements and scores indicate that the actual effectiveness of MALIAS is significantly higher than that of the baseline methods.

To complement our analysis, Table XIII compares the proposed RF-based MALIAS (*i.e.,* MALIAS$_{RF}$) with three baselines using alternative regression models in terms of net improvements (percentage points). Results are reported for the same subset of datasets used for AutoML analysis. For instance, MALIAS$_{KNN}$ in the AIOps dataset (TSAD) reports a degradation in terms of net improvements of -19.4 *pp* compared to MALIAS$_{RF}$. Overall, we observe that MALIAS$_{RF}$ constitutes a robust alternative compared to all the other strategies. Results for MALIAS$_{DT}$ implementations report degradations for all the datasets, ranging from -2 to -35.9 *pp*. For MALIAS$_{KNN}$, performance deteriorates across all datasets except VMD in SSD, where a marginal improvement of +0.8 *pp* is observed. Similarly, MALIAS$_{LR}$ exhibits performance degradation on all scenarios. Overall, these results indicate that MALIAS$_{RF}$ consistently provides the strongest performance among the evaluated alternatives, making it the most effective choice in this setting.

TABLE XIII
COMPARISON OF THREE ALTERNATIVE MALIAS IMPLEMENTATIONS AND THE RF-BASED ONE. FOR EACH DEGRADATION COMPARED TO THE RF-BASED IMPLEMENTATION, WE REPORT A RED DOWNARROW.

	Net Improvement Diff.		
MALIAS$_{RF}$ *vs.*	**TSF – AFD**	**TSAD – AIOPS**	**SSD – VMD**
MALIAS$_{DT}$	-14.8 ⇓	-35.9 ⇓	-2 ⇓
MALIAS$_{KNN}$	-9.3 ⇓	-19.4 ⇓	0.8
MALIAS$_{LR}$	-3.4 ⇓	-7 ⇓	-6.8 ⇓

Answer to RQ$_6$: In terms of effectiveness, MALIAS consistently outperforms both AutoML baselines (AutoGluon and FLAML) across all the tasks, with results demonstrating very high statistical significance ($p < 0.0001$). However, regarding training and testing times, its efficiency varies across tasks, achieving the fastest execution only in the *TSF training* and *SSD testing* settings.

VIII. PRACTICAL IMPLICATIONS

In this study, we empirically demonstrate the effectiveness of meta-learning in supporting time series analysis for software engineering activities. We emphasize that all datasets used in this work come from real-world systems operating under natural conditions, ensuring that our findings reflect authentic behavior rather than artifacts of simulation or synthetic data. Although our results indicate that an instance of MALIAS trained in one domain cannot be directly transferred to another, the approach remains promising for practical use. In particular, our findings show that, after being trained on in-domain software time series (*i.e.,* same dataset), MALIAS can be successfully applied to previously unseen time series for each of the tasks considered in our study. This suggests that its applicability in real-world SE environments is feasible. In such environments, organizations often collect several time series representing different software components or runtime characteristics (*e.g.,* memory usage or response time). A subset of these time series could be used to train the meta-learner of MALIAS. The resulting MALIAS instance could then be applied to select the most suitable time series analysis algorithm for previously unseen software time series collected within the same IT infrastructure. Moreover, MALIAS does not require manual tuning or specialized expertise, as its training and application pipeline is fully automated, including feature selection and hyperparameter tuning.

This section discusses how the proposed approach can be integrated into real-world software engineering processes for the three time series-based tasks considered in this study.

A. Forecasting of Software Telemetry Data

Time series forecasting is widely adopted for monitoring real-world applications during post-deployment [12], [13], [26]. Based on our results, meta-learning proves highly beneficial for enhancing the forecast quality of software telemetry data. To operate, MALIAS requires a set of training time series to fit the meta-learner. In real-world scenarios, where software systems are instrumented through Application Performance Monitoring (APM) tools, obtaining such data is straightforward due to the large number of software components that are continuously monitored [24], [90]. For instance, in a microservices system, an organization could use the telemetry data collected from a predefined subset of microservices to train MALIAS and subsequently apply the trained instance to automatically select the most suitable TSF algorithm for forecasting the telemetry data of the remaining microservices. Periodic retraining on recent data can help mitigate concept drift caused by system evolution. We envision that the application of meta-learning can provide substantial benefits in software engineering activities that involve forecasting software telemetry data. For instance, more accurate forecasting may lead to better-informed capacity planning [18] and more effective self-adaptation in software systems [19].

B. Anomaly Detection in Software Systems

MALIAS has also demonstrated strong effectiveness in time series anomaly detection, where its capability to dynamically select the most appropriate algorithm has enhanced the identification of software anomalies. The practical application of MALIAS to TSAD follows a process similar to that described for TSF. The main difference is that, for TSAD, MALIAS requires labeled anomalies in the training time series, which may entail some initial manual labeling effort. However, organizations often maintain records of historical software anomalies in dedicated repositories [91], which can be used for this purpose. Moreover, many conventional TSAD approaches already rely on supervised learning and therefore assume the availability of labeled anomalies. Overall, our findings suggest that meta-learning may provide practical benefits in real-world software anomaly detection.

C. Steady-State Detection in Software Performance Testing

Prior work in software performance testing has demonstrated that the use of deep learning models can support the detection of the steady state of performance, thus enabling a reasonable balance between result quality and testing time [4], [27]. However, even in such cases, selecting the optimal model remains a challenging task. As shown in Figure 1, different models may offer varying levels of accuracy depending on the characteristics of the performance measurements. Our experimental results indicate that meta-learning can enhance the accuracy of steady-state detection, suggesting its potential for deployment in real-world software performance testing scenarios. In such scenarios testing suites often comprise hundreds of performance tests [45], [92], [93], practitioners can therefore select a representative subset of these tests to collect the time series required to train MALIAS. This process first involves executing each selected test for a sufficiently large number of iterations (*e.g.,* 2,000 [5]) and then applying existing offline steady-state detection techniques [5], [37], [94] to derive ground-truth steady-state labels for the resulting time series. These labeled time series can subsequently be used to train MALIAS. The training process may be performed once or repeated periodically using recent data to mitigate concept drift caused by codebase evolution. During practical use of MALIAS on an unseen performance test, a practitioner should first run a pilot test execution or leverage past test results to obtain a representative time series from which meta-features can be extracted. These meta-features can then be provided to MALIAS, which selects the most suitable SSD algorithm to be used in future test executions for dynamically terminating warm-up iterations at runtime. As such, meta-learning can help achieve a better trade-off between the quality of test results and testing time [38].

IX. Threats To Validity

Construct Validity. A primary threat concerns the choice of *meta-features* and time series analysis algorithms employed in the study. We mitigate this threat by using large number of time series features (771) and by considering algorithms of various kinds. Specifically, the selected pool of algorithms ensures heterogeneity of models within each task, spanning time series foundation models, neural networks, other ML–based approaches, and statistical techniques. The choice of the target metrics also affects the outcomes achieved in this work. However, our selection is grounded in the practical usage scenario of each task and is aligned with common literature practices [4], [15], [50].

Some of the time series-based tasks considered in this study may support more complex software engineering activities. For instance, forecasting software telemetry data can support broader activities, such as system self-adaptation and capacity planning. In this work, however, we do not evaluate the impact of MALIAS on such activities, as doing so would require a dedicated and comprehensive study accounting for the different components involved and their potential influence on the overall outcome (*e.g.,* the planner and actuator components of a self-adaptive system [95]). We leave this investigation as an interesting direction for future work.

Internal Validity. The implementation of each candidate algorithm introduces potential limitations and affects the accuracies reported in this study. We aimed to mitigate this threat by relying on widely employed ML libraries and by adhering the implementation guidelines and tools established in prior work [4], [12], [13], [15]. Another concern involves the different operational and data access approaches of AutoML frameworks relative to MALIAS. For instance, AutoGluon trains multiple models and combines them to produce the final predictions, whereas FLAML trains multiple models but selects only one for inference. By contrast, MALIAS uses a Random Forest model (meta-learner) trained on external time series from the same domain to select a single algorithm, which is then used to train the model and generate predictions for the

target time series. It is clear that an inherent trade-off exists between the methodologies; although MALIAS exploits cross-series information offline, deployment trains only the selected model, excluding cross-model information. AutoML frameworks retrain all candidates per series, potentially increasing deployment costs but enabling cross-model exploitation. Such methodological differences prevent us to perfectly align the two experimental procedures. We tried to mitigate this concern in RQ_6 by expanding the comparative analysis to include additional baselines that more closely reflect the behavior of MALIAS.

External Validity. The findings reported in this paper may not generalize to time series extracted from other domains or employed for other tasks, as they might have different characteristics and the impact of the meta-feature might change accordingly. To mitigate this threat, our evaluation encompasses two different datasets of real-world time series for each time series-based SE task considered, for a total of six datasets. Notably, the selected datasets are heterogeneous in terms of measured components and application domain. For example, JMD includes performance tests across 30 open-source projects, while AIOPS contains telemetry data from several top-tier Internet companies.

Additionally, MALIAS has been evaluated using univariate time series, and its capacity to model high-dimensional coupled features has yet to be established.

X. Conclusion

In this paper, we introduce MALIAS, a meta-learning framework evaluated across three distinct SE tasks using six publicly available datasets of real-world software time series. MALIAS builds on established meta-learning paradigms, while the novelty of our work lies in providing the first empirical evidence of the effectiveness of meta-learning for time series-based software engineering tasks. Results show that MALIAS consistently outperforms both the best-performing models and two well-established AutoML frameworks across all considered tasks. This improvement can be attributed to MALIAS ability to automatically identify the most suitable time series analysis algorithm given the specific characteristics of the data. The implementation of MALIAS is publicly available, facilitating reproducibility and enabling future research and practitioners to reuse and adapt our approach. Notably, the design of MALIAS is task-agnostic, which enables its extension to time series-based tasks beyond those considered in this study.

Despite the significant improvements delivered by MALIAS, our study highlighted challenges in generalizing its predictive capability across datasets from different domains. Specifically, our findings indicate that MALIAS cannot effectively function as a pre-trained approach in zero-shot settings and requires domain-specific training for optimal performance. This limitation opens opportunities for future research to explore meta-learning techniques tailored explicitly for transfer learning in time series-based SE tasks. We encourage further investigation into approaches that can address these generalization challenge, which will drive our future research agenda.

References

[1] S. Zhang, Z. Zhong, D. Li, Q. Fan, Y. Sun, M. Zhu, Y. Zhang, D. Pei, J. Sun, Y. Liu, H. Yang, and Y. Zou, "Efficient kpi anomaly detection through transfer learning for large-scale web services," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 8, pp. 2440–2455, 2022.

[2] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 205–218.

[3] A. Competition, "Kpi dataset," https://github.com/NetManAIOps/KPI-Anomaly-Detection, 2018.

[4] L. Traini, F. Di Menna, and V. Cortellessa, "Ai-driven java performance testing: Balancing result quality with testing time," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24, 2024, p. 443–454.

[5] E. Barrett, C. F. Bolz-Tereick, R. Killick, S. Mount, and L. Tratt, "Virtual machine warmup blows hot and cold," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, oct 2017.

[6] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. John Wiley & Sons, 2015.

[7] Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," *ACM Comput. Surv.*, vol. 57, no. 1, Oct. 2024.

[8] J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International Journal of Forecasting*, vol. 22, no. 3, pp. 443–473, 2006, twenty five years of forecasting.

[9] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[10] R. Krishna, A. Agrawal, A. Rahman, A. Sobran, and T. Menzies, "What is the connection between issues, bugs, and enhancements? lessons learned from 800+ software projects," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. ACM, 2018, p. 306–315.

[11] M. Robredo, N. Saarimäki, M. Esposito, D. Taibi, R. Peñaloza, and V. Lenarduzzi, "Evaluating time-dependent methods and seasonal effects in code technical debt prediction," *Journal of Systems and Software*, vol. 230, p. 112545, Dec. 2025.

[12] F. D. Menna, L. Traini, and V. Cortellessa, "Time series forecasting of runtime software metrics: An empirical study," in *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024*, 2024, pp. 48–59.

[13] F. Di Menna, L. Traini, and V. Cortellessa, "Forecasting software runtime metrics: A comparative study of classical statistical, neural network, and foundation models," *Journal of Systems and Software*, vol. 240, p. 112937, Oct. 2026.

[14] Y. Syu, J.-Y. Kuo, and Y.-Y. Fanjiang, "Time series forecasting for dynamic quality of web services: An empirical study," *Journal of Systems and Software*, vol. 134, pp. 279–303, 2017.

[15] H. Si, J. Li, C. Pei, H. Cui, J. Yang, Y. Sun, S. Zhang, J. Li, H. Zhang, J. Han, D. Pei, and G. Xie, " TimeSeriesBench: An Industrial-Grade Benchmark for Time Series Anomaly Detection Models ," in *IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2024, pp. 61–72.

[16] F. Di Menna and L. Traini, "Performance regressions prediction using time series classification: A case study," in *Companion of the 17th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE Companion '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 7–11.

[17] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[18] J. Kumar and A. K. Singh, "Performance assessment of time series forecasting models for cloud datacenter networks' workload prediction," *Wireless Personal Communications*, vol. 116, no. 3, pp. 1949–1969, 2021.

[19] A. Bauer, M. Züfle, N. Herbst, A. Zehe, A. Hotho, and S. Kounev, "Time series forecasting for self-aware systems," *Proceedings of the IEEE*, vol. 108, no. 7, pp. 1068–1093, 2020.

[20] V. R. Messias, J. C. Estrella, R. Ehlers, M. J. Santana, R. C. Santana, and S. Reiff-Marganiec, "Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure," *Neural Computing and Applications*, vol. 27, no. 8, pp. 2383–2406, 2016.

[21] A. Amin, L. Grunske, and A. Colman, "An approach to software reliability prediction based on time series modeling," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1923–1932, 2013.

[22] H. Lu, W. Kolarik, and S. Lu, "Real-time performance reliability prediction," *IEEE Transactions on Reliability*, vol. 50, no. 4, 2001.

[23] K. Jia, X. Yu, C. Zhang, W. Hu, D. Zhao, and J. Xiang, "Software aging prediction for cloud services using a gate recurrent unit neural network model based on time series decomposition," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 3, pp. 580–593, 2023.

[24] F. Di Menna, V. Cortellessa, M. Lucianelli, L. Sardo, and L. Traini, "Radig-x: a tool for regressions analysis of user digital experience," in *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2024*, 2024, pp. 359–370.

[25] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. ACM, Jul. 2018, p. 387–395.

[26] G. Zhao, S. Hassan, Y. Zou, D. Truong, and T. Corbin, "Predicting performance anomalies in software systems at run-time," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 3, apr 2021.

[27] A. Trovato, L. Traini, F. Di Menna, and D. Di Nucci, "Amber: Ai-enabled java microbenchmark harness," in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2025, pp. 762–766.

[28] C. Laaber, S. Würsten, H. C. Gall, and P. Leitner, "Dynamically reconfiguring software microbenchmarks: reducing execution time without sacrificing result quality," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 989–1001.

[29] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous java performance evaluation," *SIGPLAN Not.*, vol. 42, no. 10, p. 57–76, Oct. 2007.

[30] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, "Taming performance variability," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USENIX Association, 2018, p. 409–425.

[31] D. Ardelean, A. Diwan, and C. Erdman, "Performance analysis of cloud applications," Renton, WA, pp. 405–417, Apr. 2018. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/ardelean

[32] G. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.

[33] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys*, vol. 41, no. 1, p. 1–25, Jan. 2009.

[34] R. B. Prudêncio and T. B. Ludermir, "Meta-learning approaches to selecting time series models," *Neurocomputing*, vol. 61, p. 121–137, Oct. 2004.

[35] S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer, "Descriptive statistics time-based meta features (dstmf): Constructing a better set of meta features for model selection in energy time series forecasting," in *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, ser. APPIS 2020. ACM, 2020.

[36] T. S. Talagala, R. J. Hyndman, and G. Athanasopoulos, "Meta-learning how to forecast time series," *Journal of Forecasting*, 2023.

[37] L. Traini, V. Cortellessa, D. Di Pompeo, and M. Tucci, "Towards effective assessment of steady state performance in java software: are we there yet?" *Empirical Software Engineering*, vol. 28, no. 1, p. 13, 2022.

[38] T. Kalibera and R. Jones, "Rigorous benchmarking in reasonable time," in *Proceedings of the 2013 International Symposium on Memory Management*, ser. ISMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 63–74.

[39] "Malias replication package," 2026. [Online]. Available: https://github.com/SpencerLabAQ/MALIAS

[40] E. Fuentetaja and D. Bagert, "Software evolution from a time-series perspective," in *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, ser. ICSM '02. USA: IEEE Computer Society, 2002, p. 226.

[41] W. Turski, "The reference model for smooth growth of software systems revisited," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 814–815, 2002.

[42] A. F. Ansari, L. Stella, A. C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor, J. Zschiegner, D. C. Maddix, H. Wang, M. W. Mahoney, K. Torkkola, A. G. Wilson, M. Bohlke-Schneider, and B. Wang, "Chronos: Learning the language of time series," *Transactions on Machine Learning Research*, 2024, expert Certification.

[43] S. Kong, M. Lu, J. Ai, and S. Wang, "Detection software content failures using dynamic execution information," *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 141–147, 2021.

[44] J. Rubin and M. Rinard, "The challenges of staying together while moving fast: an exploratory study," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. ACM, May 2016.

[45] L. Traini, "Exploring performance assurance practices and challenges in agile software development: An ethnographic study," *Empirical Software Engineering*, vol. 27, no. 3, p. 74, 2022.

[46] K. Veeraraghavan, J. Meza, D. Chou, W. Kim, S. Margulis, S. Michelson, R. Nishtala, D. Obenshain, D. Perelman, and Y. J. Song, "Kraken: Leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Nov. 2016, pp. 635–651.

[47] F. Di Menna, L. Traini, and V. Cortellessa, "Leveraging time series foundation models to detect performance anomalies in software systems," in *Companion of the 17th ACM/SPEC International Conference on Performance Engineering*. ACM, May 2026, p. 268–276.

[48] Z. Zeng, Y. Zhang, Y. Xu, M. Ma, B. Qiao, W. Zou, Q. Chen, M. Zhang, X. Zhang, H. Zhang, X. Gao, H. Fan, S. Rajmohan, Q. Lin, and D. Zhang, "Traceark: Towards actionable performance anomaly alerting for online service systems," in *IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice*, 2023, pp. 258–269.

[49] F. Collopy and J. S. Armstrong, "Rule-based forecasting: Development and validation of an expert systems approach to combining time series extrapolations," *Manage. Sci.*, vol. 38, no. 10, p. 1394–1414, Oct. 1992.

[50] A. Bauer, M. Züfle, S. Eismann, J. Grohmann, N. Herbst, and S. Kounev, "Libra: A benchmark for time series forecasting methods," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '21. Association for Computing Machinery, 2021, p. 189–200.

[51] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *International Journal of Forecasting*, vol. 38, no. 4, pp. 1346–1364, 2022.

[52] C. Lemke and B. Gabrys, "Meta-learning for time series forecasting and forecast combination," *Neurocomputing*, vol. 73, no. 10–12, p. 2006–2016, Jun. 2010.

[53] X. Wang, K. Smith-Miles, and R. Hyndman, "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series," *Neurocomputing*, vol. 72, no. 10–12, p. 2581–2594, Jun. 2009.

[54] M. Abdallah, R. Rossi, K. Mahadik, S. Kim, H. Zhao, and S. Bagchi, "Autoforecast: Automatic time-series forecasting model selection," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, ser. CIKM '22, 2022, p. 5–14.

[55] M. Abdallah, R. A. Rossi, K. Mahadik, S. Kim, H. Zhao, and S. Bagchi, "Evaluation-free time-series forecasting model selection via meta-learning," *ACM Trans. Knowl. Discov. Data*, vol. 19, no. 3, Mar. 2025.

[56] S. Wang, M. Rußwurm, M. Körner, and D. B. Lobell, "Meta-learning for few-shot time series classification," in *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, 2020, pp. 7041–7044.

[57] A. Abanda, U. Mori, and J. A. Lozano, "Time series classifier recommendation by a meta-learning approach," *Pattern Recognition*, vol. 128, p. 108671, Aug. 2022.

[58] J. Narwariya, P. Malhotra, L. Vig, G. Shroff, and T. V. Vishnu, "Meta-learning for few-shot time series classification," in *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, ser. CoDS COMAD 2020, 2020, p. 28–36.

[59] P. Pilyugina, S. Medvedeva, K. Mosievich, I. Trofimov, A. Kostromina, D. Simakov, and E. Burnaev, "A large-scale empirical study of aligned time series forecasting," *IEEE Access*, vol. 12, p. 131100–131121, 2024.

[60] J. R. Rice, *The Algorithm Selection Problem*. Elsevier, 1976, p. 65–118.

[61] K. Yi, Q. Zhang, W. Fan, S. Wang, P. Wang, H. He, N. An, D. Lian, L. Cao, and Z. Niu, "Frequency-domain mlps are more effective learners in time series forecasting," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt,

and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 76 656–76 679.

[62] N. Shrestha, "Detecting multicollinearity in regression analysis," *American Journal of Applied Mathematics and Statistics*, vol. 8, no. 2, p. 39–42, Jun. 2020.

[63] D. S. Young, *Handbook of Regression Methods*. Chapman and Hall/CRC, Oct. 2018.

[64] B. F. Darst, K. C. Malecki, and C. D. Engelman, "Using recursive feature elimination in random forest to account for correlated variables in high dimensional data," *BMC Genetics*, vol. 19, no. S1, Sep. 2018.

[65] W. Liu, W. Zhang, Y. Jiang, S. Chang, and S. Zhang, "Learning triple-view representation discrepancy for multivariate time series anomaly detection with multi-scale patching," in *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*, 2024, pp. 560–567.

[66] Y. Jing, J. Wang, J. Qi, Q. Qi, B. He, Z. Zhuang, N. Wu, and J. Liao, "Diner: Interpretable anomaly detection for seasonal time series in web services," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2248–2260, 2024.

[67] G. O. Ferreira, C. Ravazzi, F. Dabbene, G. C. Calafiore, and M. Fiore, "Forecasting network traffic: A survey and tutorial with open-source comparative evaluation," *IEEE Access*, vol. 11, p. 6018–6044, 2023.

[68] A. P. Jegannathan, R. Saha, and S. K. Addya, "A time series forecasting approach to minimize cold start time in cloud-serverless platform," in *2022 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, Jun. 2022, p. 325–330.

[69] E. C. Mengüç and N. Acır, "Kurtosis-based crtrl algorithms for fully connected recurrent neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6123–6131, 2018.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[72] A. Das, W. Kong, R. Sen, and Y. Zhou, "A decoder-only foundation model for time-series forecasting," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[73] M. Zhang, X. Shi, J. Huang, L. Su, and Y. Zhang, "Dyntrackr: A robust two-stage framework with attribute enhancement for kpi anomaly detection," in *IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Oct. 2024, p. 169–176.

[74] Z. Yu, C. Pei, S. Zhang, X. Wen, J. Li, G. Xie, and D. Pei, "Autokad: Empowering kpi anomaly detection with label-free deployment," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Oct. 2023, p. 13–23.

[75] P. Malhotra, L. Vig, G. M. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *The European Symposium on Artificial Neural Networks*, 2015.

[76] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. John Wiley & Sons, Dec. 1987.

[77] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," 2016. [Online]. Available: https://arxiv.org/abs/1607.00148

[78] Z. Wang, C. Pei, M. Ma, X. Wang, Z. Li, D. Pei, S. Rajmohan, D. Zhang, Q. Lin, H. Zhang, J. Li, and G. Xie, "Revisiting vae for unsupervised time series anomaly detection: A frequency perspective," in *Proceedings of the ACM Web Conference 2024*, ser. WWW '24. ACM, May 2024, p. 3096–3105.

[79] Z. Xu, A. Zeng, and Q. Xu, "FITS: Modeling time series with $10k$ parameters," in *The Twelfth International Conference on Learning Representations*, 2024.

[80] K. Boyd, K. H. Eng, and C. D. Page, *Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals*. Springer Berlin Heidelberg, 2013, p. 451–466.

[81] W. Tang, G. Long, L. Liu, T. Zhou, M. Blumenstein, and J. Jiang, "Omni-scale CNNs: a simple and effective kernel size configuration for time series classification," in *International Conference on Learning Representations*, 2022.

[82] A. Dempster, F. Petitjean, and G. I. Webb, "Rocket: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.

[83] D. E. Hilt and D. W. Seegrist, *Ridge, a computer program for calculating ridge regression estimates*. Upper Darby, Pa, Dept. of Agriculture, Forest Service, Northeastern Forest Experiment Station, 1977, 1977, vol. no.236. [Online]. Available: https://www.biodiversitylibrary.org/item/137258

[84] L. Kuncheva, "A theoretical study on six classifier fusion strategies," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, p. 281–286, 2002.

[85] R. M. Cruz, R. Sabourin, and G. D. Cavalcanti, "Meta-des.oracle: Meta-learning and feature selection for dynamic ensemble selection," *Information Fusion*, vol. 38, p. 84–103, Nov. 2017.

[86] P. R. G. Cordeiro, G. D. C. Cavalcanti, and R. M. O. Cruz, "Dynamic ensemble algorithm post-selection using hardness-aware oracle," *IEEE Access*, vol. 11, p. 86056–86070, 2023.

[87] B. Frénay, G. Doquire, and M. Verleysen, "Is mutual information adequate for feature selection in regression?" *Neural Networks*, vol. 48, p. 1–7, Dec. 2013.

[88] C. O'Leary, C. Lynch, and F. G. Toosi, "A comparative analysis of automated machine learning libraries for electricity price forecasting," *Applied Computer Systems*, vol. 29, no. 2, p. 43–52, Dec. 2024.

[89] C. Gonçalves, P. Pereira, and P. Cortez, "Argus: A large language model reasoning system for machine learning task identification," in *Proceedings of the 41st ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '26. ACM, Mar. 2026, p. 1692–1699.

[90] T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang, "Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: An experience report," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 2016, pp. 1–12.

[91] C. Bansal, S. Renganathan, A. Asudani, O. Midy, and M. Janakiraman, "Decaf: Diagnosing and triaging performance issues in large-scale cloud services," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '20. Association for Computing Machinery, 2020, p. 201–210.

[92] D. Daly, "Creating a Virtuous Cycle in Performance Testing at MongoDB," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '21. New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 33–41.

[93] K.-T. Rehmann, C. Seo, D. Hwang, B. T. Truong, A. Boehm, and D. H. Lee, "Performance Monitoring in SAP HANA's Continuous Integration Process," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 4, pp. 43–52, Feb. 2016.

[94] M. Beseda, V. Cortellessa, D. Di Pompeo, L. Traini, and M. Tucci, "A kernel-based approach for accurate steady-state detection in performance time series," *Future Generation Computer Systems*, vol. 177, p. 108233, Apr. 2026.

[95] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, May 2015, pp. 13–23.

APPENDIX A
ADDITIONAL IMPLEMENTATION DETAILS

We rely on established ML libraries—such as *TensorFlow* and *HuggingFace*—to implement TSF algorithms. The TSAD experiments are implemented based on the *EasyTSAD* suite [15], which offers a configurable experimental framework encompassing all the considered anomaly detection algorithms and evaluation metrics. The SSD experiments are implemented reusing the data extraction, training and testing scripts provided by previous related work [4], [27], [37], utilizing the same experimental setting and parameters. Concerning the MALIAS implementation, the feature extraction module is based on *TsFresh*[6], resulting in up to 771 features for each time series. The random forest regressor, along with the feature selection and hyper-parameter tuning phases, are implemented using the *scikit-learn* library. Further implementation details are available in the replication package [39]. The total amount of time required for conducting the entire experimental process—including the algorithms execution and MALIAS evaluation—is approximately one week. The complete experimentation was performed on a a Linux server equipped with 40 Intel(R) Xeon(R) 2.30GHz CPUs and 78Gb of RAM.

APPENDIX B
AUTOML FRAMEWORKS EXPERIMENTAL SETUP

Experimental Configuration. Given the extensive configurability of AutoGluon and FLAML, we preserved their default settings to approximate a realistic *out-of-the-box* usage scenario. At the same time, the experimental setup must ensure a meaningful and balanced comparison between MALIAS and the selected AutoML frameworks, which inherently support a broader range of algorithms and more powerful optimization capabilities. AutoGluon includes a set of pre-defined validated configurations—termed *presets*—designed to achieve varying performance levels; we employed the `medium_quality` preset to ensure a model pool comparable to that of MALIAS. Under this setting, the framework exploited eight models for TSF and eleven models for TSAD and SSD, excluding the stacked ensemble. As the *presets* parameter inherently controls the computational effort, no explicit time budget was imposed on the training process for this framework. AutoGluon natively supports the TSF task through the `TimeSeriesPredictor` class, whereas the TSAD and SSD tasks were executed using the `TabularPredictor` class with the task type set to `binary`. FLAML does not support a *presets*-like parameter, so we handled the computational cost and the search space using the time budget, defined as the maximum amount of seconds granted for executing the selection process over a single series. For each task, the FLAML training time budget was set to the average training time spent by MALIAS, computed per time series. FLAML automatically adjusts its model selection and hyperparameter optimization strategies according to the specified time budget. For the TSF task, FLAML was configured with the `ts_forecast` task type to learn from past data and predict the next time horizon. Given the use of rolling-origin cross-validation, predicting multiple testing windows required refitting FLAML on the same time series, which aligns with how the framework handles sequential forecasts. However, performing multiple fittings from scratch would create an unfair comparison with the other approaches that do not require such repetition. Therefore, we adopt a minimal configuration for this refitting process, restricting its time budget to one second and applying refitting solely to the best estimator selected by FLAML during the training phase. Additionally, we disable FLAML's internal validation during this stage. The TSAD and SSD tasks are performed by setting the `classification` parameter as task type, using the time series window data points as individual features. The evaluation metric used for algorithm selection was automatically determined by each AutoML framework according to the task type.

To clarify the experimental configuration differences between MALIAS and AutoML frameworks, Figure 14 illustrates how each of them accesses data and use candidate models during training and inference phases. The figure shows that, while AutoML frameworks require training all candidate models for each target series, MALIAS leverages an initial offline training to reduce the computational cost during application, requiring only the selected model to be trained during its practical usage. In contrast, AutoGluon and FLAML both train the entire candidate pool each time they are used on a new time series.

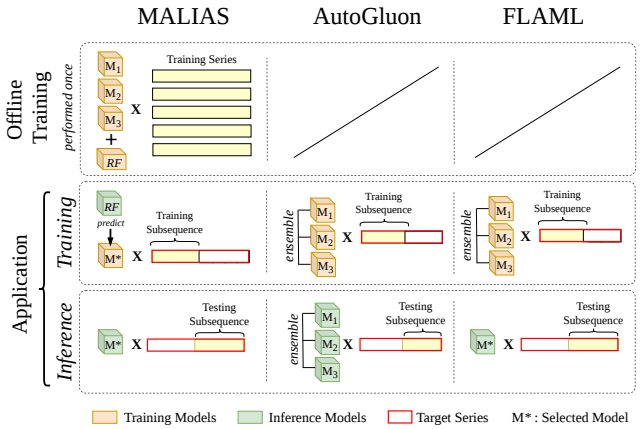


Fig. 14. Illustration of how MALIAS and AutoML frameworks leverage time series data and candidate models to generate predictions in our experimental setting. The example demonstrates a hypothetical time series task evaluated across three candidate models.

Execution Times. The training procedure of MALIAS requires executing all candidate algorithms over the *training pool* of time series, plus the fitting process of the selected algorithm for the *target* time series. To shed light on its efficiency within the context of meta-learning, we compare its training and testing time with those of the selected AutoML frameworks. In that, Section VII-F reports the average training and testing times per time series for each approach and dataset. For each k-fold iteration, the training cost of MALIAS is normalized by the number of time series within the corresponding

[6]https://tsfresh.readthedocs.io/en/latest/.*Accessed on July 4, 2025.*

testing fold, yielding the unit cost associated with a single series.

We computed the training time of MALIAS (T^M_{train}) as:

$$T^M_{\text{train}} = \frac{T_{\text{ext}_F} + T_{\text{M-fit}_F}}{k_{test}} + T_{\text{fit}_A}$$

where k_{test} is the number of time series included in the testing fold of the dataset, T_{ext_F} indicates the average time required to extract all the features and target metrics across the training folds, $T_{\text{M-fit}_F}$ refers to the total time spent for the feature selection, hyper-parameter tuning and the training of the meta-learner embedded in MALIAS, while T_{fit_A} is the average time required for training the algorithm selected by MALIAS when applied to a specific time series. T_{ext_F} and $T_{\text{M-fit}_F}$ are computed in average over the five cross-validation iterations. The testing time of MALIAS (T^M_{test}) is defined as the average time required by the chosen algorithms for generating the predictions when applied to a single time series, following the evaluation process described in Section V. In particular: for TSF, the testing time corresponds to the duration required for generating the testing windows for each time series (*i.e.,* 102 50-length windows in the AFD dataset); for TSAD, it corresponds to the time needed to compute anomaly scores for the testing portion of the time series (50% of the AIOPS series data points); and for SSD, it is defined as the time required to classify segments of length 100 for each microbenchmark execution (fork) in the VMD dataset.

AutoGluon and FLAML, instead, were applied independently to each time series, with training and testing times computed as the average across all series.

Training and testing times for MALIAS (T^M_{train} and T^M_{test}), together with the corresponding times for AutoGluon and FLAML, are reported in Table XII in Section VII-F.