# Robust and Accurate Period Inference using Regression-Based Techniques

Şerban Vădineanu
Delft University of Technology, Netherlands

Mitra Nasri
Eindhoven University of Technology, Netherlands

***Abstract*—With the growth in complexity of real-time embedded systems, there is an increasing need for tools and techniques to understand and compare the observed runtime behavior of a system with the expected one. Since many real-time applications require periodic interactions with the environment, one of the fundamental problems in guaranteeing their temporal correctness is to be able to infer the periodicity of certain events in the system. The practicability of a period inference tool, however, depends on both its accuracy and robustness (also its resilience) against noise in the output trace of the system, e.g., when the system trace is impacted by the presence of aperiodic tasks, release jitters, and runtime variations in the execution time of the tasks. This work (i) presents the first period inference framework that uses regression-based machine-learning (RBML) methods, and (ii) thoroughly investigates the accuracy and robustness of different families of RBML methods in the presence of uncertainties in the system parameters. We show, on both synthetically generated traces and traces from actual systems, that our solutions can reduce the error of period estimation by two to three orders of magnitudes w.r.t. state of the art.**

***Index Terms*—real-time systems, period inference, regression-based machine learning, robustness.**

## I. Introduction

The rapid growth of software size and complexity in real-time embedded systems has posed imminent challenges to the ability to debug systems, identify runtime deviations from the correct service [1], and detect (and evade) security attacks at runtime [2]. This raises an urge for tools and techniques to understand (or infer) the runtime behavior of a system from its observable outputs such as the traces of output messages, task executions, actuations, etc. without impacting the system itself or, in some cases, without being able to access the source code or the internal parts of the system.

In this paper, we focus on developing a tool for inferring the timing properties of a system. Such a tool can be used to **(i)** find time-bugs during the development phase (e.g., to check if activities happen with the expected frequency or period), **(ii)** detect timing anomalies and security attacks that leave a trace on the observable timing profile of the system during the operation phase (e.g., such as those explained in [2]–[4] to spot anomalies in the regularity of an activity in the system), and **(iii)** diagnosing the system after applying a patch or an upgrade during the maintenance phase (e.g., to check if a data-consumer application still performs periodically after installing an upgrade on the data-producer application).

Since many real-time applications require periodic interactions with the environment [5], one of the primary use cases of a timing inference tool is to infer the periodicity of events from a system's output traces. What makes this very first step challenging is that the observable timing traces are typically obtained from the components' interfaces and hence are impacted by the internal structure of the application, operating system, hardware platform, and their interactions. For instance, consider an execution trace that indicates the time intervals during which a certain task has occupied the processor. It is easy to infer the period if the task exclusively runs on top of dedicated hardware. It becomes harder if the task is one of the low-priority tasks in a set of periodic tasks running on top of a *real-time operating system* (RTOS) with a preemptive fixed-priority scheduling (FP) policy because then the task's execution intervals are affected by the interference from the higher-priority periodic tasks. Finally, it becomes much harder if the latter system also includes high-priority aperiodic or event-driven activities (such as interrupt services), sporadic tasks, release jitters, and deadline misses. A timing inference tool, therefore, must be *robust* against these interferences, dynamic behavior, and uncertainties; otherwise, it might not be able to address true challenges faced by real systems and hence becomes useless in practice. Furthermore, it must be accurate, else it will not be helpful to find time bugs or to detect deviations from the expected periodicity.

**Related work.** Iegorov et al. [6] are among the few pioneers who proposed a solution for the problem of inferring periods from execution traces. They created an algorithm which identifies the time intervals between consecutive jobs and computed the period as the mode of the intervals' distribution. However, their method performs poorly when the tasks have runtime execution-time variation and/or the true period of the task under analysis does not divide all other smaller periods in the task set, i.e., it is not *harmonic* with the rest of the tasks. Young et al. [7] use a *fast Fourier transformation* to infer the periodicity of messages sent on a *controller area network* (CAN) in order to detect security attacks that impact the timing of the messages. Their problem, however, is only a subset of ours since CAN applies a non-preemptive fixed-priority policy and messages have typically a fixed size with a low runtime variation on the message length.

Data-driven methods such as $k$-nearest neighbors and dynamic time-warping algorithms have been used in reverse engineering real-time systems to identify tasks from their runtime power traces [8]. In addition, *long short-term memory* (LSTM) neural networks have been used to reconstruct traces affected by noise [9]. However, to the best of our knowledge, no study so far has utilized *regression-based machine learning* (RBML) methods to infer the timing properties of real-time systems. We not only provide the first such solution, but also extensively investigate the accuracy and robustness of various families of RBML for this problem.

Finding the periodicity of a signal is a well-studied problem in signal processing research [10]–[18]. *Periodogram* [10] and *circular autocorrelation* [18] are among the widely used methods to find a plausible set of periods for a signal. However, as we will see in the experimental section, these methods perform poorly when used on signals generated from preempted tasks. Nonetheless, despite their limitations, we found them to be helpful to generate an initial set of candidate periods and hence will use them only in the first step of our solution to extract features from execution traces.

**This paper.** We consider the problem of inferring a task's period from a timed-sequence of zeros and ones (called a *binary projection*) that shows when the task was occupying the resource (see Sec. II). We consider a single processing resource (it can be a CPU, a network link, a CAN bus, etc.) that is governed by a work-conserving *job-level fixed-priority* (JLFP) scheduling policy. We assume *no prior knowledge* about the number of other tasks in the system and their parameters, execution model (preemptive or non-preemptive), runtime execution-time variations, and release jitters.

Our framework uses two signal-processing techniques, i.e., periodogram and circular autocorrelation, to extract features from the binary projection, treat and reduce the size and the number of features, and then use them to train a set of RBML methods (in Sec. III). This work **(i)** presents the first period inference framework[1] that utilizes RBML methods, and **(ii)** thoroughly investigates the accuracy and robustness of different families of RBML methods in the presence of uncertainties in the system parameters, noise resulted from aperiodic tasks in the input data or missed jobs.

Our results show that RBML methods infer tasks' periods with an average error of 0.4% (for periodic tasks with or without execution-time variation), 1.1% (for periodic tasks with release jitter), and 0.4% (for task sets with a mixture of periodic, sporadic, and aperiodic tasks) while the state of the art [4] has an average error of 1160%, 1950%, and 156%, respectively. On case studies from actual systems [19,20], the error of our (best) solution was below 1.7%. Sec. V provides insight on the strengths and weaknesses of different families of RBML methods for the problem of period inference.

## II. System Model and Problem Definition

We assume a system with a single (processing) resource (such as a CPU core, I/O or CAN bus, or a link on the network). The resource can be occupied/used by a set of tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, scheduled by a work-conserving JLFP scheduling policy on the resource, i.e., only the highest-priority job among the ready jobs can be dispatched on the resource, where a job is an instance of a task in $\tau$. JLFP policies include widely implemented/used scheduling algorithms in real-time systems such as the earlier-deadline first (EDF), fixed-priority (FP), and first-in-first-out (FIFO) scheduling policies. Furthermore, we assume no restriction on whether each task executes preemptively or non-preemptively.

A task in $\tau$ can be activated periodically, sporadically, or aperiodically. A periodic or sporadic task is identified by $\tau_i = (C_i^{min}, C_i^{max}, T_i, D_i, \sigma_i)$, where $C_i^{min}$ and $C_i^{max}$ are the *best-case* and *worst-case execution times* (BCET and WCET), $T_i$ is the period, $D_i$ is the relative deadline (which is assumed to be equal to the period), and $\sigma_i$ is the maximum release jitter of the task. If the task is sporadic, its period indicates the minimum-inter arrival time between its activations. An aperiodic task is identified by a 3-tuple $\tau_j = (C_j^{min}, C_j^{max}, D_j)$, where $C_i^{min}$ and $C_i^{max}$ are the BCET and WCET and $D_j$ is the relative deadline of the task, respectively. We further assume that all timing parameters are positive integer values in $\mathbb{N}^+$ with the exception of $C_i^{min}$ and $\sigma_i$ that can be 0. The total utilization of the system is denoted by $U$ and is the sum of the utilization of all periodic and sporadic tasks, i.e., $U = \sum u_i$, where $u_i = C_i^{max}/T_i$. The *hyperperiod* of a task set, denoted by $H$, is the least common multiple of the periods.

A task $\tau_i$ generates an infinite number of instances, called jobs, during the life-time of the system. We use $J_{i,k}$ to denote the $k$-th job of a task $\tau_i$. The priority of a job $J_{i,j}$ is denoted by $p_{i,j}$ and is determined by the scheduling policy. We assume that at any time instant $t$, either one of the tasks in $\tau$ or the *idle task*, denoted by $\tau_0$, is running on the resource.

A trace $\mathcal{T} = ([t_s, t_e], \langle \epsilon_1, \epsilon_2, \ldots, \epsilon_N \rangle)$ represents a time-ordered sequence of symbols that represents a schedule generated by the JLFP scheduler for the task set $\tau \cup \{\tau_0\}$ from the time $t_s$ to $t_e$. Each symbol $\epsilon_i \in \mathcal{T}$ is an identifier (index) of a task that was occupying the resource at time $i$, where $i \in \{t_s, t_{s+1}, \ldots, t_e\}$. Hence, $\epsilon_i \in \tau \cup \{\tau_0\}$. The length of a trace is $|\mathcal{T}| = t_e - t_s$. Figs. 1(a) and (b) show a schedule of a task set with 4 tasks and the equivalent trace of that schedule. In practice, one may utilize operating system commands such as *top* and *trace* commands or the Linux trace toolkit to obtain a trace of a certain task. There is no need to distinguish a preemption from starting a new job or a self-suspension, etc. On a CAN bus, one can obtain projections by observing the messages transferred on the bus to form a trace.

To formally define problems considered in the paper, we need to introduce two other notions that are tied to a trace: *binary projection* and *ternary projection*. A binary projection for task $\tau_i$ is a sequence of zeros and ones that represents the times at which a job of the task was occupying the resource in the trace. Fig. 1(c) shows the binary projection of the task $\tau_3$.

**Definition 1.** *A* binary projection *of a trace $\mathcal{T}$ for task $\tau_i$, denoted by $P_i^B = \langle p_1, p_2, \ldots, p_{|\mathcal{T}|} \rangle$, is a time-ordered sequence of elements $p_k$, where*

$$p_k = \begin{cases} 1, & \epsilon_k = i \\ 0, & \textit{otherwise} \end{cases} . \tag{1}$$

A ternary projection (Fig. 1(d)) for a task contains resource-

[1]Available at: https://github.com/SerbanVadineanu/period_inference

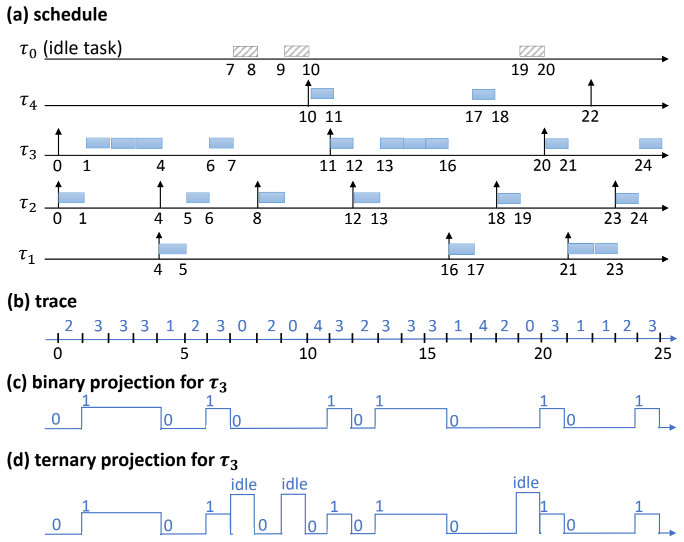

Fig. 1: A task set with one aperiodic task ($\tau_1$ with $C_1^{max} = 2$), two sporadic tasks ($\tau_2$ and $\tau_4$ with $C_2^{max} = 1$ and $C_4^{max} = 2$) and one periodic task with release jitter ($\tau_3$ with $C_3^{max} = 4$) scheduled by a FP policy (assuming $p_i = i$). **(a)** shows a schedule, **(b)** shows the trace of the schedule, and **(c)** shows the binary projection of task $\tau_3$.

idle intervals in addition to what is stored in a binary projection.

**Definition 2.** *A* ternary projection *of a trace $\mathcal{T}$ for task $\tau_i$, denoted by $P_i^T = \langle p_1, p_2, \ldots, p_{|\mathcal{T}|}\rangle$, is a time-ordered sequence of elements $p_k$, where*

$$p_k = \begin{cases} 1, & \epsilon_k = i \\ idle, & \epsilon_k = 0 \\ 0, & otherwise \end{cases} . \tag{2}$$

Next, we define the two versions of the *period inference* (PI) problem:

**Problem 1.** *Find the period of $\tau_i$ from its projection $P_i^B$.*

**Problem 2.** *Find the period of $\tau_i$ from its projection $P_i^T$ provided that $P_i^T$ does not include a deadline miss from $\tau_i$.*

It is worth noting that the only input to the Problems 1 and 2 is a projection. Since a projection is just a sequence of limited symbols ('0', '1', and 'idle'), it does not contain any information about the scheduling policy or tasks' parameters (such as the execution times, release jitter, periods, etc.). Moreover, the projections themselves do not contain information about whether or not there are tardy jobs (i.e., jobs that have completed after their deadline) in the original trace.

## III. Regression-based Period Mining

This section first introduces the challenges of the period inference (PI) problem and then presents our solution.

**Challenges.** As mentioned earlier, the PI problem has a long history in signal processing. Methods such as periodogram [10] and circular autocorrelation (or autocorrelation for short) [18] have been applied to infer periodicity of a signal and shown to work well in the presence of small (or standard) noise.

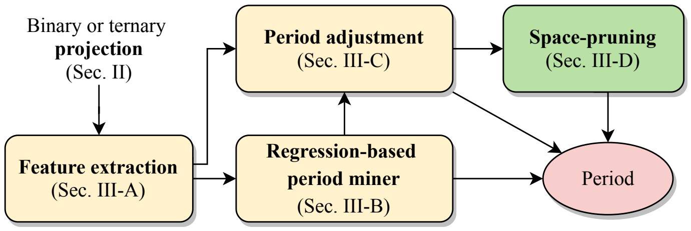
Binary or ternary projection (Sec. II)
Period adjustment (Sec. III-C)
Space-pruning (Sec. III-D)
Feature extraction (Sec. III-A)
Regression-based period miner (Sec. III-B)
Period

Fig. 2: Our period inference framework. The edges denote the information flow between our algorithms.

However, they do not perform well (see Sec. IV) when applied to the PI problem because: **(i)** they may generate many period candidates most of which are irrelevant, **(ii)** although they assign a weight (called *power*) to each candidate, there is no direct relation between the weight and the true period, **(iii)** they cannot cope with preemptions well because they perceive each preemption as a new occurrence of the event under analysis (which adds a significant amount of noise to their inputs), and **(iv)** the true period is not necessarily among their generated candidates (specially in the autocorrelation method).

We then decided to look into the learning-based methods that could work well on the PI problem. We specially focused on those whose decision logic is *explainable* and traceable by a human. Therefore, we deliberately avoided using deep neural networks for the problem or for the feature extraction. However, this raised the next challenge: how to extract meaningful and helpful features from a projection? A starting point could be to use the whole binary projection as a feature and let the machine-learning method figure out the period. However, that could lead to two major issues: **(i)** dimensionality problems with the feature space, and **(ii)** having inputs with varying-length. High-dimensional feature spaces typically lead to sparse data which in turn reduces the efficiency and increases the runtime of model learning [21]. Moreover, most machine-learning methods require a fixed input size which implies that the input projection must be cut (unanimously for all projections of all training and testing task sets). However, since task sets have different hyperperiods, putting a predetermined cut-off threshold could either lead to low accuracy (if the cut-off is too short) or to a huge runtime and low efficiency in learning the model (if the cut-off is too long).

**Solution highlights.** The framework we propose to solve the period inference problem suggests a four-stage pipeline where Stage 0 extracts features and Stages 1 to 3 are for accuracy improvement of period estimation. Fig. 2 shows the pipeline and the stages. In Stage 0, we extract a fixed set of features from the top $k$ highest-rank candidates of the periodogram and autocorrelation methods (see Sec. III-A). In Stage 1, we use supervised-learning methods, and in particular, the regression-based machine learning (RBML) methods, to determine the relationship between our feature vectors and the target output, i.e., task period (see Sec. III-B). RBML methods are commonly used when the goal is to predict a continuous output that takes order into consideration (in our case, the period). We call our RBML solution *regression-based period*

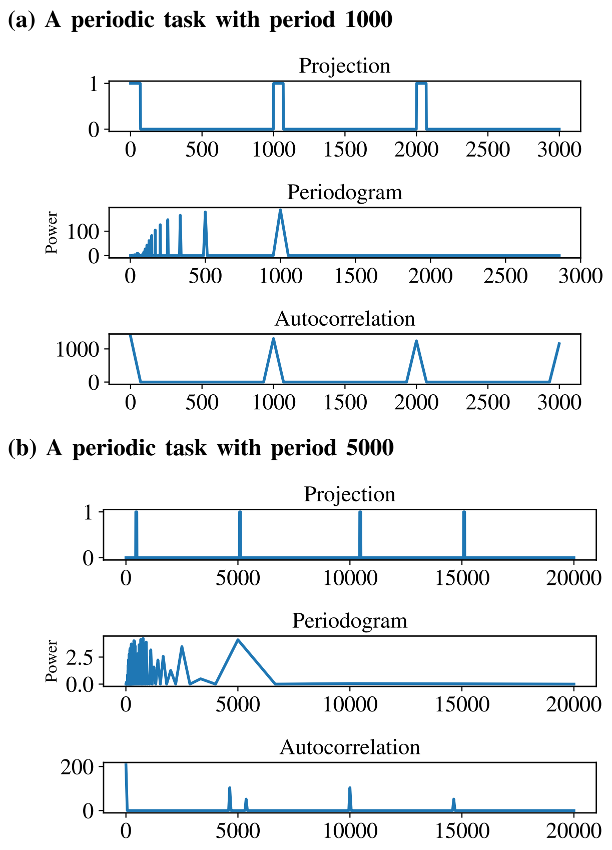

Fig. 3: Periodogram and circular autocorrelation methods applied on task projections for a task with **(a)** period 1000, and **(b)** period 5000 from a system containing 4 tasks, with a total utilization of 30% scheduled by rate monotonic. The other two tasks have a period of 2000 and 10000, respectively.

*miner* (RPM). In Stage 2, we further adjust the predictions of RPM according to a set of high-ranked candidates from periodogram and autocorrelation. This aims to use RPM as a referee whose purpose is to highlight the most accurate peak from the two signal-processing methods (see Sec. III-C) Finally, Stage 3 introduces some pruning rules using the extra information provided in ternary projections to further restrict the number of candidates (Sec. III-D).

## *A. Feature Extraction*

Next, we explain our feature extraction using and briefly introduce the periodogram and autocorrelation methods.

**Periodogram [10].** Consider the binary projection as a sequence $P_i^B$, where $p_n$ is the nth item of the projection and its discrete Fourier transform $X(f)$. The periodogram $\mathcal{P}$ gives an estimation of the spectral density of the discrete signal $P_i^B$ and is obtained from the squared magnitude of the Fourier coefficients $X(f)$, as presented in [22]:

$$\mathcal{P}(f) = \frac{1}{N} \|X(f)\|^2, \tag{3}$$

where $N = |P_i^B|$ is the sequence length and $\mathcal{P}(f)$ is the power of frequency $f$. The Fourier coefficients $X(f)$ can be obtained from the sequence $P_i^B$ as follows

$$X(f) = \sum_m p_m \cdot e^{-j \cdot 2 \cdot \pi \cdot f \cdot m}. \tag{4}$$

The norm of a Fourier coefficient is the magnitude of that coefficient, namely, $\|X(f)\| = \sqrt{Re\{X(f)\}^2 + Im\{X(f)\}^2}$, where $Re\{X(f)\}$ and $Im\{X(f)\}$ are the real and imaginary coefficients for each frequency $f$, respectively.

Fig. 3-(a) and (b) show two periodograms obtained for two periodic tasks with period 1000 and 5000 from a task set with four tasks scheduled by rate monotonic scheduling policy. The horizontal axis shows the frequency values $f$ and the vertical axis shows the power of each frequency, i.e., $\mathcal{P}(f)$. As can be seen in Fig. 3(a), the highest peak in this example (here, peak refers to a jump in the diagram) of the periodogram indicates the true period of the task, i.e., 1000. However, for the task with period 5000, this observation does not hold; the true period of this task is not the highest-peak but the 5th highest peak. The lower the priority of a task, the higher is the amount of interference it will have in its schedule. These interferences make the projections less regular and hence result in a more irregular periodogram that has many peaks.

**Circular Autocorrelation [18]**. It is a metric that describes how similar is a sequence to its past values for different circular phase shifts. We use Vlachos et al. [12] method to compute the circular autocorrelation:

$$\mathcal{ACF}(k) = \frac{1}{N} \sum_{n=0}^{N-1} p_k \cdot p_{n+k}, \tag{5}$$

where $N$ is the sequence length and $k$ is the phase shift. In the case of period inference problem, we would expect that the highest value of the autocorrelation function would be at a lag $k$ equal to the true period.

A practical way to compute the $\mathcal{ACF}$ is to translate the operations into the frequency domain. Since (5) is a convolution, one can compute it with the dot product between the Fourier coefficients of the sequence and their complex conjugates [12]:

$$\mathcal{ACF} = \mathcal{DFT}^{-1} X \cdot X^*, \tag{6}$$

In this paper, we apply the discrete Fourier transform on the projection and extract the Fourier coefficients using (4). Furthermore, we perform the dot product between the coefficients and their complex conjugates and apply the inverse Fourier transform on the result to obtain the autocorrelation.

Fig. 3 illustrates the usage of autocorrelation when the input is a projected trace. Firstly, we notice that the highest value that this technique exhibits is for a lag (period) of 0. This behavior is normal, since the highest similarity between a signal and itself is present when the two signals perfectly overlap with each other, e.g., at time 0. Hence, the peak at 0 is excluded from the examination. The other observation is that, similar to the periodogram, the autocorrelation method is able to discover the true period only in the case from Fig. 3(a), while for the second period, its top peak indicates an erroneous value. Moreover, we observe that the autocorrelation is sensitive to low utilization values. In Fig. 3(b), although the task has not been preempted,

| Algorithm | Nickname | Category |
|---|---|---|
| Cubist Regression [24]–[26] | *cubist* | Rule-based |
| Generalized Boosting Regression [27] | *gbm* | Boosting |
| Averaged Neural Network [28] | *avNNet* | Neural Networks |
| Extremely Randomized Regression Trees [29] | *extraTrees* | Random Forests |
| Bayesian Additive Regression Tree [30] | *bartMachine* | Bayesian Models |
| Support Vector Regression [31] | *svr* | Support Vector Machines |

TABLE I: Overview of best performing families of regression algorithms and for each family the best model [32].

the slight variation at the start of its execution causes the projection to not have any overlap with itself when is shifted by the true period of 5000. As a result, the autocorrelation method could not detect the actual periodic behavior. However, it could observe two smaller peaks slightly shorter and slightly larger than 5000 at 4635 and 5365, respectively.

It is worth noting that, both periodogram and autocorrelation methods have an $O(m \log m)$ time complexity, where $m$ is the length of the projection [23].

**Extracting fixed-size features.** Our fixed-size candidate list is constructed from the top $k = 3$ peaks of the outputs of the two methods, namely, we gather $k$-highest peaks from periodogram and $k$-highest peaks from the autocorrelation methods[2]. This allows us to work on a much smaller dimension for the input-data and have fixed input size to use with our regression-based solution. For the cases when there are fewer than $k$ peaks for a method, the number of features is completed by appending the highest peak of that method until we reach the desired $k$. The choice on the number of features, i.e., $k = 3$, was made after evaluating the impact of $k$ on various scenarios and finding out the suitable value that results in a high accuracy without increasing the dimensions of the feature space (Fig. 5(b) in Sec. IV compares different choices.).

## B. Regression Methods

Regression analysis is a method originating from statistics, whose purpose is to estimate the relationship between a dependent variable (or ”outcome“) denoted by $Y$ and one or more independent variables (or ”features“) denoted by $X$. In machine learning, regression is employed when the aim is to predict a continuous output, which takes order into consideration. A regression model is formally described by

$$Y_i = f(X_i, \beta) + e_i, \tag{7}$$

where $Y_i$ is the outcome variable, $X_i$ is a feature vector, $\beta$ represents unknown parameters, and $e_i$ is an additive error term (residual) associated with the prediction.

Since we try to estimate the period from a projection, in our regression scenario, the dependent variable $Y_i$ is the task's period $T_i$. The independent variables $X_i$ contain the features we extracted at the previous step, while the function $f$ comes from the choice of a regression algorithm, whose parameters $\beta$ we need to estimate through training. Thus, our goal is to choose the form of function $f$ and to compute the estimates of the parameters $\hat{\beta}$ such that the function has the best fit on the data. In order to assess how well the model fits the data, the predicted outcome, i.e., $\hat{Y}_i = f(X_i, \hat{\beta})$, is compared against the true dependent variable. The comparison is present in the shape of a *loss function* $L(Y, f(X, \hat{\beta}))$, where $Y$ is a vector containing the outcome variables and $X$ includes all vectors of independent variables. For instance, the most commonly used loss function is the *mean square error* (MSE) (also used in our paper):

$$MSE = \frac{\sum_{j=1}^{N} (Y_j - \hat{Y}_j)^2}{N}, \tag{8}$$

where $N$ is the total number of observations.

[2]Note that the width of a peak is correlated with the position of the peak in periodogram (the further from the origin the larger the width). Thus, it does not provide enough information to be considered as a feature for regression.

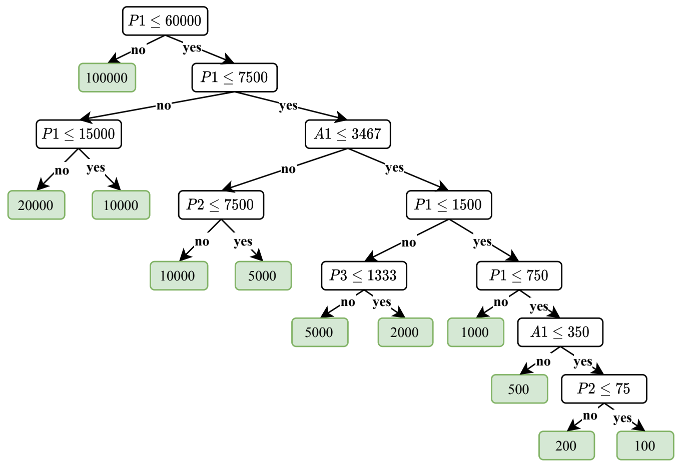

Fig. 4: A simple regression tree fitted on a data set with 4 tasks and a total utilization of 30% (see details in IV-A for automotive tasks)

**The choice of regression methods.** Table I lists the overall best performing families of regression algorithms and for each family the best model, as suggested by Delgado et al. [32] in their extensive recent survey on the performance and effectiveness of regression methods. These methods present distinctive characteristics in their implementation, namely, they do not theoretically dominate each other. Hence, in order to answer the question “which regression method performs best for the period-inference problem”, we implemented and investigated all of these methods to gather insights about their performance on our particular problem. We, however, anticipate to see that the tree-based solutions (*cubist*, *gbm*, *extraTrees*, *bartMachine*) have a better performance than *svm* and *avNNet* because we expect the transition from a set of candidate periods (the features) to the true period to be better approximated by a set of rules and/or comparisons rather than a linear or non-linear combination of these features as in *svr* and *avNNet*, respectively.

**Regression trees.** A majority of the RBML methods in Table I are variations of *regression trees*. A regression tree [33] recursively partitions the feature space of the data into smaller regions until the final sub-divisions are similar enough to be summarized by a simple model in a leaf. This model can be simply the average of the outcomes from that sub-division.

Fig. 4 shows the rules generated by a regression tree that was trained on the automotive task sets with four periodic tasks and 30% utilization (see details of the task set generation in IV-A). The features used for training are the three highest peaks from the periodogram (denoted by $P1$, $P2$, and $P3$) and autocorrelation (denoted by $A1, A2$, and $A3$) methods. The non-terminal nodes represent the *rules* that will be used to guide the inference process by narrowing down the period estimate of a new task.

To make it more tangible, we explain how to use the regression tree in Fig. 4 to estimate the period of the two tasks in Fig. 3(a) and (b). In the first step, we derive the three highest peaks of the periodogram and autocorrelation methods to build the feature vectors $X_1$ and $X_2$ for the first and second tasks, respectively. Here, $X_1 = \langle P1$=1000, $P2$=500, $P3$=333, $A1$=1000, $A2$=2000, $A3$=3000$\rangle$ and $X_2 = \langle P1$=769, $P2$=666, $P3$=5000, $A1$=4635, $A2$=10000, $A3$=5365$\rangle$. Next, we traverse the tree by evaluating the rules starting from the root node. For example, for the first task, $P1 = 1000$ and hence the condition in the root node (i.e., $P1 \leq 60000$) is satisfied. Thus, we go to the right branch and repeat the process until we reach to a leaf. The value in the leaf is the period estimate. In this example, the trained model can accurately estimate both tasks' period.

An interesting observation in Fig. 4 is the exclusion of $A2$ and $A3$ in the tree's rules which basically means that these two features had no impact on the final period estimate. With a further investigation, we observed that typically in task sets with low utilization, the trained regression trees tend to be smaller and rules contain fewer features because there are less preemptions (noise) in the input. However, with an increase in utilization, the tree is forced to consider more features and even become deeper to keep the estimation error low.

Training a regression tree can be done in $O(m \cdot N \cdot \log N)$, where $m$ is the number of features (in our case 6) and $N$ is the number of samples (projections) used for training. Later in Sec. IV, we provide an evaluation on the runtime and memory consumption of various RBML methods.

**Cubist Regression (cubist) [24]–[26,34].** It is a regression tree whose leaves embed linear regression models instead of simple 'estimates of the output'. The tree can be further reduced by combining or pruning the rules via collapsing the nodes of the trees into rules.

By training a *cubist* regression model on the same data-set as in Fig. 4, we obtain the following rules:

1) **If** $(A1 \leq 2000)$ **then** return $P1$,
2) **If** $(P1 \leq 1250 \ \wedge \ A1 > 2000)$ **then** return 5000,
3) **If** $(P1 > 1250)$ **then** return $P1$.

In this example, we observe that while the rules and outputs rely on the top candidates of the periodogram, they are not limited to them. For example, rule 2 outputs the period 5000 which is not among the three top features of periodogram. The *cubist* regression uses these rules to compensate for projections where the periodogram is wrong.

*Cubist* regressions consume notably less memory than the regression trees (see Sec. IV) and hence they might be a better choice when the solution must have low memory consumption and runtime. However, we also noticed a growth in the number of rules when they are trained on task sets with high utilization because then the underlying regression tree from which the cubist regression rules are obtained gets larger and deeper when the number of preemptions increases.

**Generalized Boosting Regression (gbm) [27,35].** This algorithm is a regression tree-based solution which uses a committee of regression trees of fixed size. The initial prediction of the algorithm starts from a leaf, which contains the average value of the outcome variables (i.e., the periods). The next step is to compute the residuals of this initial prediction against the true output (true period). Next, a regression tree is fitted on the data, but having the previously computed residuals as the outcome variables. In order to preserve the generalization capabilities of the model, the results from the tree are multiplied by a constant value. Afterwards, the output from the tree is added to the initial leaf to obtain a new set of predictions, which are again used to compute residuals. The process is repeated until a maximum number of trees is reached.

**Extremely Randomized Regression Trees (extraTrees) [29,36].** The algorithm relies on a committee of regression trees for its predictions. When building the trees, this method randomly picks a rule for each feature (instead of searching for a rule that minimizes the error) and then chooses the one that provides the lowest error. Hence, a randomized regression tree is much faster to build than a regular regression tree.

**Bayesian Additive Regression Tree (bartMachine) [30, 37].** Similar to *gbm*, this method also relies on a group of trees, where each tree is fit on the residuals of the predictions from a previous tree. The major difference is that *bartMachine* is based on a probability model containing a set of priors for the tree structure and a likelihood for the leaves' values. *extraTrees*, *gbm*, and *bartMachine* stop building the model when a given (maximum) number of trees is achieved.

**Averaged Neural Network (avNNet) [28,38].** The technique involves a committee of five multilayer perceptrons having the same size, but trained using different random seeds. The network is set to have linear output neurons, which makes it suitable for regression. Finally, the predictions from the five networks are averaged to provide the final estimate.

**Support Vector Regression (svm) [31,39].** The goal of *svr* is to find a line or a hyperplane that is able to fit the most data points within a certain margin from it. Moreover, it can accommodate non-linear trends by fitting the line in a transformed feature space using a kernel function.

Sec. IV and V provide further insights about the performance of these RBML methods.

## C. Candidate Selection

As the example in Fig. 3 shows, the true period is among one of the peaks of the periodogram and autocorrelation, although not always by the highest. After further investigations, we observed that on the one hand, in a majority of projections, the true period is indeed among the peaks of periodogram and autocorrelation. However, it is hard to know which of those peaks just by looking at their power or rank. On the other hand, the RBML methods typically predict only an approximation of the real period which is not always equal to the true one (resulting in non-zero errors in most cases). Thus, we introduce a further pruning phase on the output of our RPM method and create a method called *RPM with period adjustment* (RPMPA).

RPMPA treats the RPM method as a referee which chooses the right period from a set of candidates. Namely, it first calculates the period estimate using the RPM method and then finds the closest period to this estimate from a fixed set of values gathered from the 20 highest peaks of each of the periodogram and autocorrelation (hence, 40 candidates in total). The number of candidates (i.e., 40) is a hand-tuned value and comes from experimenting on many task sets (see Sec. IV-B).

### D. Improving the Accuracy for Ternary Projections

Choosing from a pool of candidates can help in considerably increasing the performance. However, if the regression algorithm significantly deviate from the true period, the chosen candidate may be even further from the target. Thus, by ensuring that only the most relevant candidates are kept in the set, we can expect an even better accuracy. The goal of our *space-pruning method* (SPM) is to derive a lower and an upper bound on the possible set of period values. These bounds allow us to remove the impossible period values from the candidate set generated from the highest 20 peaks of each of the periodogram and autocorrelation methods.

In order to prune impossible candidates and find an upper bound on the period, we use the extra information available in ternary projections, i.e., the idle times. Since the underlying scheduling policy is work-conserving, it will not leave the recourse idle if there is a ready task in the system. Thus, if a task has accessed the resource anywhere between two idle times, it must have released a job somewhere between those idle times. For example, according to Fig. 1(d), at least one job of $\tau_3$ must have been released in the interval [10, 19] since there is at least a '1' in the task's projection in this interval.

Our key idea to derive an upper bound on the task's period is to traverse the ternary projection to find pairs of consecutive intervals separated by idle times in which the task has occupied the resource. We call them *effective intervals*. From each pair of effective intervals, we then obtain one upper bound on the task's period. After traversing the whole projection, the smallest upper bound found is the value we use to prune the candidates obtained from the peaks of periodogram and autocorrelation.

**Extracting effective intervals.** Let $x$ be a time instant at which $p_x = 1$ and $\exists z < x$ in the ternary projection such that $p_z = idle$, then the beginning of the effective interval that contains the time instant $x$, denoted by $I^s(x)$, is the latest idle time prior to the execution of $\tau_i$, namely,

$$I^s(x) = \max\{k \mid z \leq k < x \ \wedge \ p_k = idle \ \wedge \\ \forall p_y,\ k < y < x,\ p_y \neq idle\}. \tag{9}$$

For example, in Fig. 1(c), $I^s(11) = 9$, $I^s(13) = 9$, or $I^s(14) = 9$ while $I^s(20)$ or $I^s(24)$ are 19. Using Eq. (9) for any $p_x = 1$, one can obtain all starting points of effective intervals in a projection. In the example in Fig. 1(c), the starting point of the effective intervals are at times 9 and 19 (assuming that the resource was idle prior to time 0), otherwise the first effective interval starts at 19.

**Calculating an upper bound for $T_i$.** Let $I^s_j$ and $I^s_{j+1}$ be the starting points of two consecutive effective intervals for task $\tau_i$. The latest time instant at which a job of $\tau_i$ has occupied the resource during interval $I_j = [I^s_j, I^s_{j+1})$ is obtained as follows

$$\mathit{fin}(I^s_j, I^s_{j+1}) = \max\{k \mid k \in [I^s_j, I^s_{j+1}) \ \wedge \ p_k = 1 \ \wedge \\ \forall p_y, k < y < I^s_{j+1}, p_y \neq 1\}, \tag{10}$$

where $p_x$ is the value of the ternary projection at time $x$. For example, in Fig. 1(c), $\mathit{fin}(9, 19) = 15$.

**Theorem 1.** *Given the start time of two consecutive effective intervals $I^s_j$ and $I^s_{j+1}$ for task $\tau_i$, $T_i$ must follow*

$$T_i \leq \mathit{fin}(I^s_j, I^s_{j+1}) - I^s_{j-1}. \tag{11}$$

*Proof.* By definition, there is at least one time instant in $I_{j-1} = [I^s_{j-1}, I^s_j)$ in which task $\tau_i$ has occupied the resource. Hence, the earliest release time of $\tau_i$ in $I_{j-1}$ is $I^s_{j-1}$. Moreover, since the last moment at which the task has occupied the resource in the interval $I_j = [I^s_j, I^s_{j+1})$ was at $\mathit{fin}(I^s_j, I^s_{j+1})$, the latest possible release time of a job of the task cannot be later than $\mathit{fin}(I^s_j, I^s_{j+1})$. Hence, (11) provides an upper bound on the inter-arrival time between two jobs of $\tau_i$. $\square$

For the example in Fig. 1(c), $T_i \leq \mathit{fin}(9, 19) - 0 = 15$.

**Calculating a lower bound for $T_i$.** If we know that the projection does not contain any deadline misses (e.g., because the system uses separate monitoring tools to indicate a deadline miss and a dropped job), we can obtain a lower bound on the period of task $\tau_i$ (otherwise the lower bound is 0). To do so, we extract the largest interval $L$ in which the task $\tau_i$ does not occupy the resource. If there is no deadline miss in the projection, then the length of the largest interval in which no job of task $\tau_i$ has occupied the resource is upper bounded by $|L| \leq 2 \cdot T_i$. Namely, the period of the task cannot be smaller than half of that interval. The reason is that, in the worst case, the largest interval that can be observed is between a job which executed right after its release and the next job that executed right before its deadline, resulting in a value slightly lower than two times the period. For example, in Fig. 3(b), the interval [5200, 11300] is the largest interval in which no job of the task has occupied the resource. Hence, period of $\tau_i$ must be at least 3050. This allows us to remove a large number of peaks of periodogram from the candidate set since it typically generates many peaks at small frequencies.

Since both the lower bound and the upper bound can be calculated at the same time (by passing through the projection only once), they have a linear time complexity w.r.t. the projection length.

## IV. Empirical Results

We performed a set of experiments to answer the following questions: **(i)** Does our framework improve the accuracy w.r.t. the state of the art? **(ii)** How robust is our solution against uncertainties and non-deterministic events? **(iii)** How do various families of RBML methods compare against each other? **(iv)**

What are the tradeoffs between the accuracy, runtime, and the memory requirements of various RBML methods? and **(v)** How good our solution generalizes to systems that are widely different from those on which it trained?

We divided our task systems into three groups: periodic task systems where every task is periodic but tasks might have release jitter or execution time variation (Sec. IV-C and IV-F), non-periodic task systems, where the task under analysis is periodic but the rest of the system might not be periodic (Sec. IV-D), and case studies from actual systems (Sec. IV-E). The source code and our evaluation framework for these experiments are both available on *github* [40].

## A. Experimental Setup

For the experiments in Sec. IV-C, IV-D, and IV-F, we considered two types of task sets: automotive benchmark application and synthetic task sets. For the automotive benchmark applications, we adopted the model proposed by Kramer et al. [41] for task sets used in automotive industry, where task periods are chosen randomly from $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ms with a non-uniform distribution provided in [41]. For simplicity, we refer to the traces of these task sets as *automotive traces*. Our synthetic task sets are comprised of non-harmonic periods. In order to ensure that the chosen periods cover evenly all magnitudes, we used a log-uniform distribution as suggested and described by Emberson et al. [42]. The periods are thereby generated for the range [100, 10000] with a base period of 100ms. For simplicity, we refer to the traces of these task sets as *log-uniform traces*. We use Strafford's Randfixedsum algorithm [42] to generate random utilization values for the tasks and then use the utilization and the period to calculate the WCET of each task. To generate the traces, we use *Simso* [43], an open source and flexible simulation tool that generates schedules under various scheduling policies and setups.

**Evaluation strategy.** The data set used for training the regression models is composed of the projections from 2000 traces (we saw no benefit in increasing the data set size in our preliminary experiment). The length of a trace is set to be either six hyperperiods (traces without random variations) and ten hyperperiods (execution time variation and release jitter) for the experiments in Sec. IV-C to IV-E. The same trace lengths are used for the testing to capture enough random behavior. In Sec. IV-F, we specifically investigate the impact of trace length on the accuracy of testing.

**Metric.** The metric we use to evaluate the accuracy is the average error, which is the mean of the individual errors a method makes for every period in a test set unless it is explicitly stated that the error has been obtained for only one task in the task set. Furthermore, we calculate the error of one experiment (that includes 2000 task sets) by using *5-fold cross-validation*. Namely, we divide the data set into five randomly chosen subsets of equal size. Out of the five subsets, four are used for training and one is used for testing. We measure the error of the testing and repeat the process until all five subsets have been used once for testing.

**Baselines.** We considered three baselines: **(i)** PeTaMi, a mining algorithm for periodic tasks [6], **(ii)** periodogram [10], and **(iii)** autocorrelation [18]. PeTaMi represents the state of the art on period inference in the real-time systems community, while the other two represent widely used solutions from the signal-processing literature. These two were chosen to evaluate the improvements made by our RPM and RPMPA over solutions that are (only) based on signal-processing techniques.

We compare the RBML methods mentioned in Table I, denoted by *cubist* [26], *gbm* [27], *avNNet* [28], *extraTrees* [29], *bartMachine* [30], and *svr* [31]. Each of these methods is defined by a set of hyperparameters that require tuning for improving the model's fit on the data. Hence, we performed an additional tuning phase using random search on the parameter's space. This step was integrated in the cross-validation process such that every training set comprised of the four subsets, is further split into a training and validation set. The parameters are varied while being trained on the training set and the model's performance is estimated on the validation set. The purpose of doing one more split is to avoid bias by not involving the test set into the parameter choice.

To be able to focus on the accuracy of the RBML methods, we only show the results of RPM method in Figs. 5, 6(a) to (o), and 8. We compare the accuracy of RPM with RPMPA in Figs. 6(p, q, r), 7, and Table II. We performed our evaluation on a Dutch supercomputer based in the cloud. We used thin nodes with $2 \times 16$-core 2.6GHz Intel Xeon E5-2697A v4 (Broadwell) and 64GB of memory.

## B. Parameter Tuning

Before evaluating our solutions, we need to determine their parameters, i.e., the number of features for RPM and the number of candidates for RPMPA, since they impact the solution's accuracy. The evaluation from Fig. 5(a) was performed on an aggregated data set, containing automotive traces with four levels of utilization (30%, 50%, 70% and 90%). Similarly, for the second experiment from Fig. 5(b) we used data sets incorporating the four utilization values and we also kept 20% execution time variation for the tasks in both data sets. We picked *extraTrees*, since it is a representative member of tree-based algorithms and is less affected by the increase in the number of its features in terms of runtime. The experiments were conducted by generating 20 random splits of the data set into training and testing sets (for every parameter value). The model would then be fit on the training data and the average error measured on the test data.

Fig. 5(a) shows how the error for *extraTrees* decreases when we include more features. The gain in accuracy becomes insignificant after adding more than three features from periodogram and autocorrelation. Thus, we kept three features from each of the periodogram and autocorrelation (i.e., six in total). We further analyzed the impact of the number of candidates for RPMPA method on accuracy. As shown in Fig. 5(b), a relatively small number of candidates is required in order to achieve a low error until it reaches saturation.

## C. Assessing Accuracy in Periodic Systems

**Impact of system utilization.** Figs. 6(a, b) show the average error as a function of the total utilization for task sets with

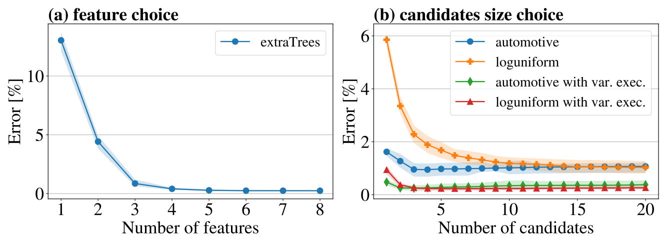

Fig. 5: The impact of the number of features (for RPM) and the number of candidates (for RPMPA) on the solution's accuracy. Note that the shade around the curves represents the confidence intervals for 0.95 confidence level.

8 tasks. The error of the regression models increase with the increase in the utilization (which in-turn increases the number of preemptions). Furthermore, we observe a dramatic shift in PeTaMi's accuracy when it is applied on log-uniform traces. This shift is a result of having non-harmonic periods in log-uniform traces. In contrast, we see that the accuracy of our regression-based solutions has not been negatively affected when applied on non-harmonic periods.

**Impact of the number of tasks.** Figs. 6(c, d) shows that the error reduces when there are more tasks in log-uniform traces for some of the tree-based solutions such as *gbm*. It is due to the decrease in the individual task utilization. Thus, although the system is as congested, the individual projection of a task contains larger idle intervals and shorter execution times that are likely not preempted much. This enables the periodogram to extract more meaningful features. However, in automotive task sets, the algorithms are rather unaffected by the number of tasks in the trace since they already have a good performance even for lower number of tasks.

**Impact of execution time variations.** From Figs. 6(e, f), we observe that the regression-based methods are more robust to runtime execution-time variations than the baselines, showing a similar trend for both types of traces to the case with constant execution time. Also, with an increase in the execution time variation in Figs. 6(g) and 6(h), we notice that most of the RBML methods are robust (w.r.t. to this variation) for automotive traces, while for log-uniform traces the error decreases with the increase in variation. This behavior is due to the reduction in the average execution time for individual tasks. Since the execution time for a job is drawn from a uniform distribution in the range $[(1-\alpha)\times WCET,\ WCET]$, the wider the interval becomes the lower is the average execution time. Having lower execution time is associated with lower utilization for the system and we previously observed that the methods perform better for lower utilization values.

**Impact of release jitter.** Figs. 6(i, j) show that the release jitter has a much bigger impact on the error than the execution-time variation. One possible explanation is that the periodogram, which provides most of the information to the algorithms, is negatively impacted by jitter, thus, it produces less useful features for training. However, we observe that some regression algorithms such as *extraTrees* and *cubist* are still able to keep a low error even for this challenging scenario.

**Impact of candidate adjustment method (RPMPA).** Figs. 6(p, q) show that RPMPA has about 50% less error than RPM for most RBML methods when used for cases with execution time variation and, implicitly, on ideal traces too. However, when the signal processing techniques (periodogram and autocorrelation) are disturbed, as is the case for release jitter, the adjustment step has a negative impact on the accuracy.

**Impact of space-pruning method (SPM).** By analyzing Fig. 6(r) we observe that the inclusion of an upper and a lower bound for SPM contributes to reducing the error even further, proving that the regression is still prone to mistakes even when choosing candidates. However, this solution is expected to show little benefits for systems with large utilization, when fewer intervals of idle-time will be present. The reason is that in that case, SPM will provide upper bounds that are so large that they will not contribute much to filtering infeasible candidates. This indicates some room for improvement in the SPM method.

While conducting this experiment we noticed that under specific setups there can be cases where no candidates are left after the pruning phase. This situation occurs most frequently when the release time jitter disturbs so much the signal processing techniques that they only generate infeasible candidates. As a consequence, when analyzing the effect of release time jitter, we defined two criteria to provide a period estimate when no candidates are available. Namely, we either select the output of regression (SPM-R) or we select the upper bound (SPM-UB). Fig. 7 shows the results for SPM on traces with jitter. In all cases, both versions of SPM succeed in reducing the error of RPMPA by 45% points. Also, SPM-UB is able to achieve an average error below RPM for *cubist*, *gbm*, and *bartMachine*, while for *extraTrees*, although it has a larger error, it presents a much narrower confidence interval. Thus, we can expect that the estimate of SPM-UB based on *extraTrees* to be more reliable than the corresponding RPM.

## *D. Assessing Robustness*

**Robustness w.r.t. the presence of higher-priority aperiodic tasks.** Another step in our evaluation framework was to test the robustness of our methods for a setup that mimics possible real-world challenges. Thus, we created a configuration with 12 automotive tasks (6 periodic and 6 sporadic) scheduled by Rate Monotonic and under the interference of high-priority aperiodic tasks, arriving according to a Poisson process with a rate $\lambda$=0.0005 events/ns or 5 arrivals at every 10us. Furthermore, we focused on analysing one periodic task in scenarios of having high, medium, and low priority, respectively. For each of the three priority scenarios, the task's priority has been chosen randomly in the ranges [1, 3] for high, [4, 7] for medium, and [8, 12] for low-priority tasks.

Figs. 6(m, n, o) show the error as a function of utilization for the tree-based algorithms, periodogram and PeTaMi. The periodogram is affected significantly when priority changes from high to low (comparing Figs. 6(m) and (o)) due to the increase in the number of preemptions in low-priority tasks which in-turn causes more noise in periodogram. In contrast,

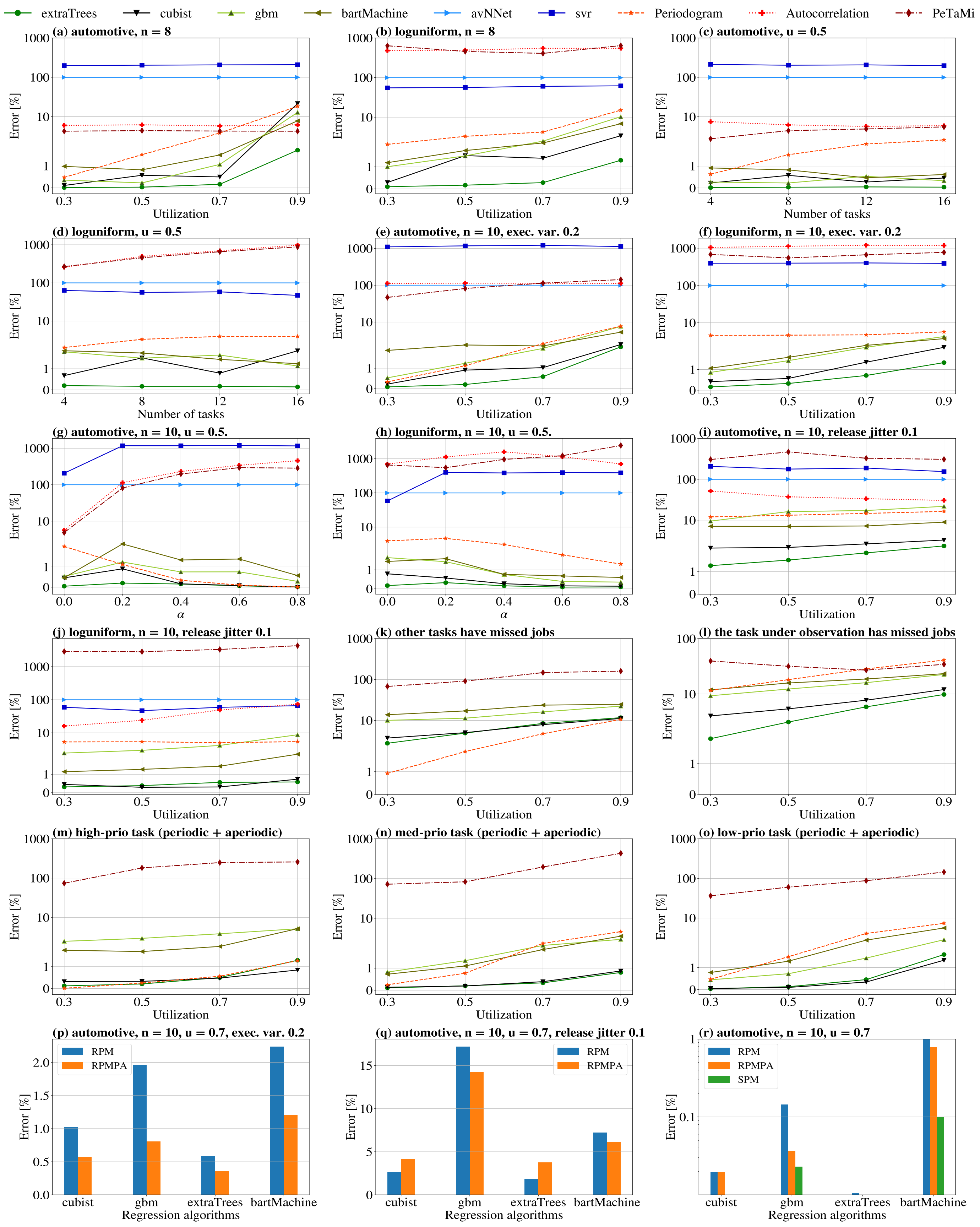

Fig. 6: Experimental results

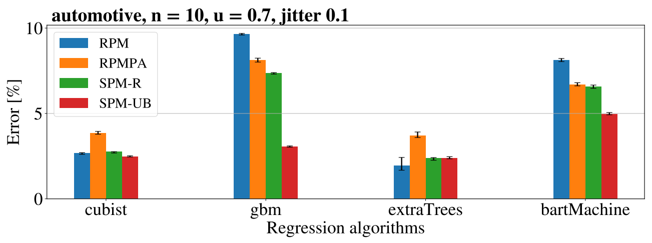

Fig. 7: Space-pruning for traces with jitter.

| Data set | Algorithm | RPM [%] | RPMPA [%] |
|---|---|---|---|
| CAN 1 [19,44] | *extraTrees* | 28.2568 | 1.6179 |
| | *cubist* | **3.2461** | **0.9703** |
| | *gbm* | 15.8224 | 1.3919 |
| | *bartMachine* | 20.8588 | 14.0103 |
| | Periodogram | 5.3368 | - |
| CAN 2 [20,44] | *extraTrees* | 13.0256 | **1.7039** |
| | *cubist* | **5.5005** | 2.8963 |
| | *gbm* | 19.8341 | 9.8881 |
| | *bartMachine* | 14.956 | 5.0264 |
| | Periodogram | 9.5448 | - |

TABLE II: Results on the two CAN data sets

the error of RPM algorithms increases only slightly in large utilization values. We also see that the error of *gbm* and *bartMachine* at low utilization values is smaller when the task under analysis has a low priority. It is due to the fact that these two algorithms may not be able to generalize well when the periodogram has low error. Having a low error for periodogram means having less significant (shorter) peaks, which in turn do not provide enough information for these algorithms to excel.

**Robustness w.r.t. dropping jobs.** Next, we explore the impact of having missed jobs in the input projections on the accuracy of our solutions. The setup included 10 automotive tasks. We considered two scenarios: **(i)** the tasks under analysis has dropped jobs (with a 15% probability), and **(ii)** all the other tasks were dropping jobs. Figs. 6(k, l) show that all algorithms exhibit a relatively higher error when there are dropped jobs and the utilization is higher in comparison with experiments with no dropped job (e.g., comparing Fig. 6(a) and (k) or (l)). This increase is due to the fact that projections are imperfect and even can be misleading when some jobs are dropped. Moreover, periodogram is affected notably when the task under analysis drops jobs. However, while the RBML methods (RPM) show little variations from one case to the other, they are still able to retain meaningful information from their features even when the task under analysis has a low utilization.

## *E. Case Study*

In this section, we validate our period-inference methods on two case studies. We use two data-sets consisting of traces coming from Controller Area Networks (CANs), denoted by CAN 1 [19,44] and CAN 2 [20,44] in Table II. The first data set consists of 988,987 messages with 27 tasks and the second one of 2,369,868 messages with 45 tasks.

In order to generate our test data, we split the projections from the messages into smaller projections of 100 jobs. As for our training data, we synthetically generated traces that would provide a good proxy for real data, namely, we created a data set of 6000 automotive traces, with 20 tasks scheduled by non-preemptive rate monotonic, with 50% utilization and 5% jitter as the training set. The results from Table II show that our methods successfully estimated the periods of the messages on the actual use case, having errors below 2% for both datasets.

## *F. Assessing Other Aspects*

**Number of rules for *cubist*.** Fig. 8(a) shows an increase in the number of rules generated by *cubist* as the utilization grows for log-uniform traces with 12 tasks. This behavior is expected since the projections become more complex as the number of preemptions increases in higher utilization values. For example, the average number of rules stored at 30% utilization is 16, but it raises to 65 at 90% utilization.

**Generalization assessment: training and testing on different number of tasks.** Figs. 8 (b)-(c) illustrate how the mismatch between the number of tasks within the traces used for training and the traces used for testing affects the accuracy of the tree-based algorithms. The experiment is conducted on log-uniform traces with 70% utilization and without uncertainties (since we want to capture a large range of periods and also isolate the effect of the discrepancy between the number of tasks). Fig. 8(b) shows that training on traces with fewer tasks than the target system leads to 2.9% higher error in average than the opposite scenario (Fig. 8(c)). Moreover, in both cases, the smaller the gap between the number of tasks in training and testing traces, the lower the error. We also see that *cubist* has the best generalization capability; it has less than 2.5% error as long as it is trained with task sets that have at least 8 tasks.

**Generalization assessment: training and testing on different task set types.** For this experiment, we train the models on an *aggregated set* containing 2000 log-uniform traces for every utilization value in {30%, 50%, 70%, 90%} and 12 tasks. Afterwards, the evaluation is completed on 2000 automotive traces with 12 tasks for each of the aforementioned utilization values *individually*. The results, summarized in Fig. 8(f), show that *cubist* has the lowest accuracy (i.e., below 8% for utilization values lower than 90%). Comparing Fig. 8(f) and Fig. 6(a) (where the training and testing were done on the same task set types) we see only a slight difference in the error of the RBML methods which shows that they rather generalize well.

**Generalization assessment: training and testing on different projection lengths.** The goal of this experiment is to see how the error of the RBML methods is affected by the length of their input trace during the inference phase (testing). This experiment is performed on log-uniform traces with 70% utilization and 10 tasks, with the particularity that, when testing, we limit the length of the projections to *a certain multiple of the task's period* (shown on the horizontal axis of Fig. 8(e)). As it can be seen in Fig. 8(e), both *cubist* and *extraTrees* are able to estimate the true period with less than 4% error even when only two jobs of the target task appear in the trace. As expected,

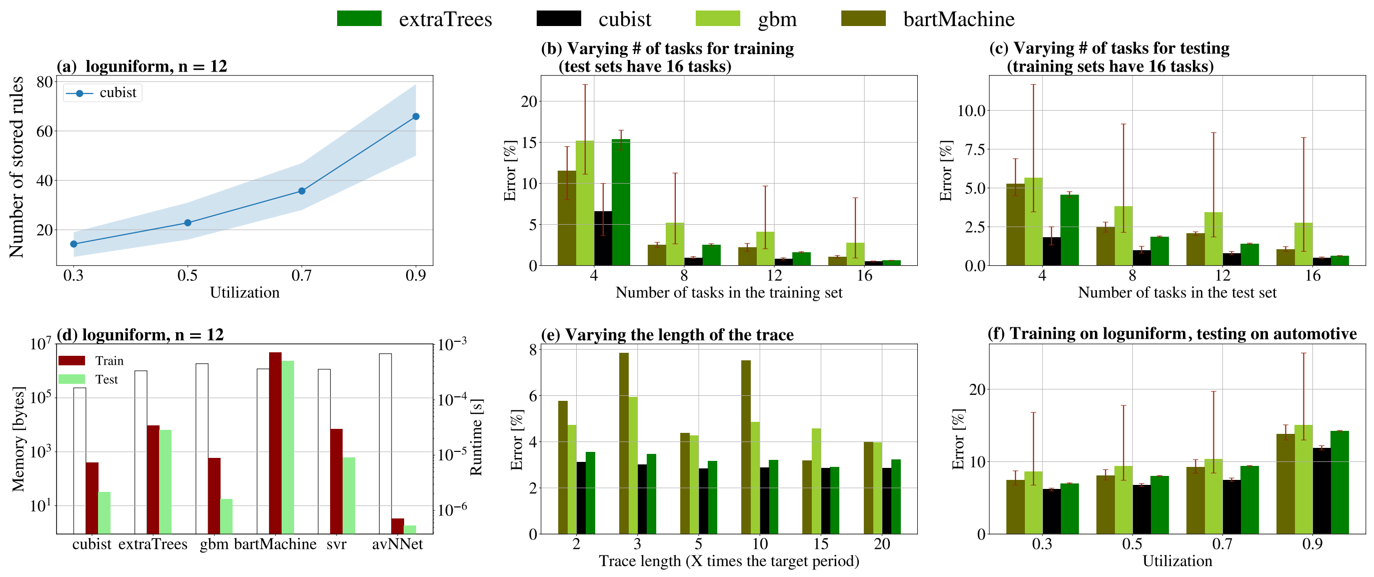

Fig. 8: Other evaluation criteria: **(a)** the number of rules generated by *cubist* algorithm as a function of utilization, **(b , c)** the impact of training and testing on data sets with different number of tasks, **(d)** the memory consumption and runtime of the RBML methods, **(e)** the impact of the trace length on the accuracy, and **(f)** the impact of training and testing on different task set types on the accuracy.

the error reduces gradually with the increase in the length of the trace. In contrast, *bartMachine* has the largest fluctuations in error, making it less reliable when the projection's length is lower than 20 times the period of the target task.

## V. Discussions

**ExtraTrees and cubist.** Among our experiments, *extraTrees* algorithm has the lowest error in almost all setups. However, when it trains on a different task set type than the ones used for testing (shown in Sec. IV-E or IV-F), its does not generalize well. This makes *extraTrees* a good choice when we have access to training data from the same system we want to test on (e.g., when it is used for monitoring a known system). On the other hand, when the target system is unknown, or we do not have access to the traces, *cubist* regression is a better choice; with a similar accuracy as *extraTrees* and better generalizability.

**Non-tree-based solutions.** The experiments have confirmed our statement from Sec. III-B, where we expected the tree-based solutions to outperform the other types of RBML methods, when applied to our problem. We noticed that *avNNet* has almost a constant yet very large error of 100% for all experiments. This expresses the inability of the algorithm to learn a non-linear mapping from the input to the output, deciding instead to approximate the output as a constant value, namely the average of the periods from the training set. Since the test set is generated the same way as the training set, it will have a similar average value for its periods, thus keeps the error constant. Furthermore, *svr* is also not a good choice for the period-inference problem since our feature space is rather sparse, namely the features from periodogram and autocorrelation do not have values that place the data points close to each other in this space. Hence, *svr* is not able to find a suitable hyperplane to fit the data. Moreover, we see that the candidate adjustment step has a significant impact on reducing the errors. However, in the presence of large release jitters, PRMPA must be used cautiously as it may increase the error.

**Discussions on memory consumption and runtime.** Fig. 8(d) addresses both the memory requirements and the runtime of the six considered RBML methods. We notice that *cubist* has a considerably low memory consumption compared to the rest. It is also almost the fastest solution during both the training and testing phases among the well-performing algorithms. On the other end of the spectrum, *bartMachine* has the slowest training and testing phases, and *avNNet* has the largest memory consumption among the considered algorithms.

## VI. Conclusions

In this paper, we introduced the first regression-based machine learning (RBML) solution for the problem of inferring a task's period from its binary projections. We investigated six most-successful families of RBML methods for this problem and provided comprehensive evaluations and discussions about their accuracy and robustness under various scenarios. We proposed further steps for improving the accuracy by creating period-adjustment and space-pruning methods that use the properties of a work-considering scheduler to prune the space of valid periods of a task. Our solutions proved to be robust and highly accurate. The average observed error of our (best) solution was under 1% in most scenarios including those with a mixture of periodic, aperiodic, and sporadic tasks, execution time variation, and release jitter while the existing work has two to three orders-of-magnitude higher errors. On the case studies from actual systems, the error of our best solution was 1.7%. In the future, we would like to explore RBML methods to infer the timing properties of parallel applications running on multiprocessor platforms.

## REFERENCES

[1] C. Young, H. Olufowobi, G. Bloom, and J. Zambreno, "Automotive intrusion detection based on constant can message frequencies across vehicle driving modes," in *ACM Workshop on Automotive Cybersecurity*, 2019, p. 9–14.

[2] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes, "On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 103–116.

[3] M. Salem, M. Crowley, and S. Fischmeister, "Anomaly detection using inter-arrival curves for real-time systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 97–106.

[4] O. Iegorov and S. Fischmeister, "Mining task precedence graphs from real-time embedded system traces," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 251–260.

[5] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "An empirical survey-based study into industry practice in real-time systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[6] O. Iegorov, R. Torres, and S. Fischmeister, "Periodic task mining in embedded system traces," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2017, pp. 331–340.

[7] C. Young, H. Olufowobi, G. Bloom, and J. Zambreno, "Automotive intrusion detection based on constant can message frequencies across vehicle driving modes," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, 2019, pp. 9–14.

[8] K. Lamichhane, C. Moreno, and S. Fischmeister, "Non-intrusive program tracing of non-preemptive multitasking systems using power consumption," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1147–1150.

[9] I. Sucholutsky, A. Narayan, M. Schonlau, and S. Fischmeister, "Deep learning for system trace restoration," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[10] A. Schuster, "On the investigation of hidden periodicities with application to a supposed 26 day period of meteorological phenomena," *Terrestrial Magnetism*, vol. 3, no. 1, pp. 13–41, 1898.

[11] C. Berberidis, W. G. Aref, M. Atallah, I. Vlahavas, and A. K. Elmagarmid, "Multiple and Partial Periodicity Mining in Time Series Databases," in *European Conference on Artificial Intelligence (ECAI)*, 2002, pp. 370–374.

[12] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *SIAM international conference on data mining*. SIAM, 2005, pp. 449–460.

[13] T.-H. Li, "Detection and Estimation of Hidden Periodicity in Asymmetric Noise by Using Quantile Periodogram," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 3969–3972.

[14] R. G. McKilliam, I. V. L. Clarkson, and B. G. Quinn, "Fast Sparse Period Estimation," *IEEE Signal Processing Letters*, vol. 22, no. 1, pp. 62–66, 2014.

[15] Y. Malode, D. Khadse, and D. Jamthe, "Efficient Periodicity Mining Using Circular Autocorrelation in Time Series Data," *International Research Journal of Engineering and Technology (IRJET)*, vol. 2, no. 3, pp. 430–436, 2015.

[16] P. Unnikrishnan and V. Jothiprakash, "Daily Rainfall Forecasting for One Year in a Single Run Using Singular Spectrum Analysis," *Journal of Hydrology*, vol. 561, no. 1, pp. 609–621, 2018.

[17] T. Puech, M. Boussard, A. D'Amato, and G. Millerand, "A Fully Automated Periodicity Detection in Time Series," in *International Workshop on Advanced Analysis and Learning on Temporal Data (AALTD)*, 2019, pp. 43–54.

[18] J. A. Gubner, *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.

[19] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," in *Annual Conference on Privacy, Security and Trust (PST)*, vol. 00, 2017, pp. 57–5709.

[20] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based intrusion detection system for in-vehicle network," in *Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–6.

[21] R. Bellman, "Curse of dimensionality," *Adaptive control processes: a guided tour. Princeton, NJ*, vol. 3, p. 2, 1961.

[22] C. T. Leondes, *Computer Techniques and Algorithms in Digital Signal Processing: Advances in Theory and Applications*. Elsevier, 1996.

[23] M. B. Priestley, *Spectral Analysis and Time Series Probability and Mathematical Statistics*, 1981, no. 04; QA280, P7.

[24] J. Quinlan, "Learning with continuous classes," in *5th Australian joint conference on artificial intelligence*, vol. 92. World Scientific, 1992, pp. 343–348.

[25] ——, "Combining instance-based and model-based learning," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 236–243.

[26] ——, *C4. 5: programs for machine learning*. Elsevier, 2014.

[27] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[28] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 2007.

[29] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[30] H. A. Chipman, E. I. George, R. E. McCulloch *et al.*, "Bart: Bayesian additive regression trees," *The Annals of Applied Statistics*, vol. 4, no. 1, pp. 266–298, 2010.

[31] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[32] M. F. Delgado, M. S. Sirsat, E. Cernadas, S. Alawadi, S. Barro, and M. Febrero-Bande, "An extensive experimental survey of regression methods." *Neural Networks*, vol. 111, pp. 11–34, 2019.

[33] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[34] M. Kuhn and R. Quinlan, *Cubist: Rule- And Instance-Based Regression Modeling*, 2020, r package version 0.2.3. [Online]. Available: https://CRAN.R-project.org/package=Cubist

[35] B. Greenwell, B. Boehmke, J. Cunningham, and G. Developers, *gbm: Generalized Boosted Regression Models*, 2019, r package version 2.1.5. [Online]. Available: https://CRAN.R-project.org/package=gbm

[36] J. Simm, I. M. de Abril, and M. Sugiyama, *Tree-Based Ensemble Multi-Task Learning Method for Classification and Regression*, 2014. [Online]. Available: http://CRAN.R-project.org/package=extraTrees

[37] A. Kapelner and J. Bleich, "bartMachine: Machine learning with Bayesian additive regression trees," *Journal of Statistical Software*, vol. 70, no. 4, pp. 1–40, 2016. [Online]. Available: http://www.rob-mcculloch.org/

[38] M. Kuhn, *caret: Classification and Regression Training*, 2020, r package version 6.0-86. [Online]. Available: https://CRAN.R-project.org/package=caret

[39] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch, *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2019, r package version 1.7-3. [Online]. Available: https://CRAN.R-project.org/package=e1071

[40] "Regression-Based Period Miner," 2020, https://github.com/SerbanVadineanu/period_inference.

[41] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[42] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, 2010, pp. 6–11.

[43] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," 2014.

[44] Hacking and Countermeasure Research Lab, http://ocslab.hksecurity.net/Datasets, 2010, [Online; accessed 01-July-2020].