Regression Model Trees: Compact Energy Models for Complex IoT Devices

Birte Friesel
Universität Osnabrück
Osnabrück, Germany
birte.friesel@uos.de

Olaf Spinczyk
Universität Osnabrück
Osnabrück, Germany
olaf@uos.de

***Abstract*—The energy and timing behaviour of embedded components such as radio chips or sensors plays an important role when developing energy-efficient cyber-physical systems and IoT devices. However, datasheet values generally have low accuracy and may be incomplete, and performing new energy measurements after each code or hardware configuration change is time-consuming. While energy models – automatically generated from benchmarks exercising all relevant device configurations – offer a solution, they should have both low prediction error and low complexity in order to be useful to humans as well as energy simulations. With today's increasingly complex devices and drivers, generating compact and accurate energy models is becoming harder due to non-linear effects and interdependencies between configuration parameters. To address this issue, we present Regression Model Trees. By combining software product line engineering and energy modeling methodologies, these are capable of automatically learning complex energy models from benchmark data. Using energy and timing benchmarks on two embedded radio chips and an air quality sensor, we show that Regression Model Trees are both more accurate than conventional energy models and less complex than state-of-the-art approaches from the product line engineering community. Thus, they are easier to understand and use for humans and algorithms alike. We observe two- to 100-fold complexity reduction, and a maximum energy model error of 6 % with cross-validation.**

***Index Terms*—IoT, energy models, regression trees**

I. Introduction

Many cyber-physical system components have a variety of configuration options. Both software (e.g. operating systems or mesh network protocols) and hardware (e.g. radio chips or sensors) can be configured towards specific use cases and trade-offs, with profound effects on energy requirements and latency of individual operations.

While benchmarks of cyber-physical systems and IoT components can capture these effects for a specific configuration and use case, and allow for reasoning about which network protocol or radio chip is most efficient in that specific application, energy models are much more useful. By benchmarking the energy behaviour in the entire configuration space and learning an energy model from these observations, the resulting model can be used to estimate the energy properties of any hardware configuration in any use case, thus eliminating the need for repetitive benchmark runs. With appropriate models, multi-objective optimization is also possible, e.g. considering the influence of bit rate and transmit power on transmission energy vs. expected packet loss [1].

Many energy modeling approaches rely on internal state machines [2]. For example, a radio chip is either in a sleep, idle, transmit, or receive state. They assume that the energy behaviour of each state is either constant (i.e., the arithmetic mean of observations) or can be expressed by a linear function (which is fitted on observations by linear regression).

In practice, however, configuration options often have non-linear and conditional effects. If a radio module is configured for variable packet length, its transmission duration will be a function of packet length (linear) and bit rate (inverse) – otherwise, it is only a function of bit rate. Conventional energy models cannot deal with these situations. We propose to address this issue by combining two different approaches.

Fitted Function Sets are capable of automatically determining and fitting both linear and non-linear functions to describe the energy behaviour of individual device states. We have successfully used these for energy model generation in the past [3]. However, they do not support interdependent configuration variables with conditional effects.

Regression Trees are tailored towards high-dimensional configuration spaces and naturally support interdependent configuration options [4]. They are frequently used in the software product line engineering community to express the influence of configuration options on *non-functional properties* such as latency, memory usage, or energy. Extensions with linear functions exist, but are not optimized for non-linear effects [5].

In this paper, we present *Regression Model Trees*: a combination of regression trees and fitted function sets that is able to automatically generate compact and accurate energy models for complex IoT devices. We contribute a formal definition of regression model trees, and an evaluation of model accuracy and complexity on two radio modules, an environmental sensor, and an embedded AI application. Utilizing these, we show that regression model trees fall within a sweet spot of high accuracy (up to $6\,\%$ model error for energy, and $18\,\%$ for non-energy attributes) and low complexity (up to 100 times fewer tree nodes than regression trees), making them an ideal choice for machine- and human-readable energy models.

Our open-source implementation of regression model trees, benchmark data, and evaluation code are available at https://ess.cs.uos.de/git/software/cpsiotbench-2022-artifacts.

In the next section, we introduce the various modeling approaches related to this paper. We then present our con-

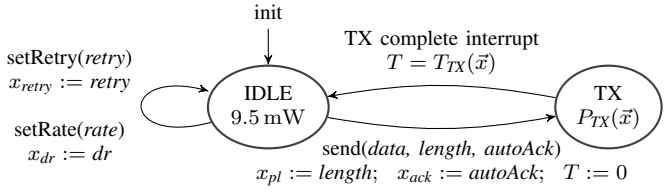

Fig. 1. PPTA excerpt for an nRF24 radio transceiver with configurable data rate, packet length, auto-ack, and auto-retry. Transmit power P_{TX} and duration T_{TX} depend on hardware configuration $\vec{x}$. The transceiver automatically switches from TX to IDLE once the time specified by T_{TX} has passed.

tribution, regression model trees, in Section III, and provide evaluation results in Section IV. After examining related work in Section V, we conclude in Section VI.

II. Modeling Methods

We cover state machines with configurable parameters, fitted function sets, regression trees, and regression forests.

A. Parameterized Priced Timed Automata

In general, embedded devices are driver- and environment-controlled state machines with distinct energy requirements in each state. For instance, a radio chip typically has idle, receive and transmit states as well as at least one low-power sleep state. State transitions can be caused by driver function calls, external events (e.g. packet received), or timeouts (e.g. transmission complete) [6].

This makes *Parameterized Priced Timed Automata* (PPTA) a natural choice for energy models. PPTA extend deterministic finite automata with *timers* that can be reset by transitions and cause other transitions when a timeout is reached, *prices*, and *parameters*. In our case, prices are the power consumption of each individual state. If desired, the duration and power consumption of state transitions (i.e., driver function calls) can be modeled as well.

Parameters capture the active hardware configuration. Each driver function or interrupt may change configuration parameters, and each power or timeout attribute is a function of these parameters. Users need only specify the PPTA structure and how function calls affect parameters; based on that, benchmark generation and execution as well as energy model generation can be performed automatically [6]. Fig. 1 shows a PPTA excerpt for an nRF24L01 radio transceiver.

Now, the challenge is to automatically find suitable functions for each power, duration, and timeout attribute. Given configurations $\vec{x}_1, \vec{x}_2, \ldots \in \mathbb{R}^n$ and a set S of corresponding benchmark results $y_1, y_2, \ldots \in \mathbb{R}$, each function should be able to accurately predict the energy consumption of previously unseen configurations, and not require manual intervention during model generation. In the remainder of this section, we examine regression methods that are suitable for this task. We note that these can also be used to learn and predict attributes that are not part of a PPTA, such as latency, size, or memory requirements of embedded applications.

B. Fitted Function Sets

Regression analysis is a decades-old method of optimizing (*fitting*) a function so that it learns to predict observations y from input vectors $\vec{x}$ [7]. Given a function $f(\vec{x}, \vec{\beta})$, regression algorithms adjust regression variables $\vec{\beta}$ so that the loss of the error term $\varepsilon_i = y_i - f(\vec{x}_i, \vec{\beta})$ is minimal. A typical loss function is the sum of squared residuals $\sum_i \varepsilon_i^2$.

Regression analysis is often used in energy models for embedded devices [8]. For instance, consider the duration of a radio transmission with packet length x_{pl} and bitrate x_{dr}. After a constant set-up time, the radio module sends a fixed-length preamble followed by variable-length data. So, a suitable function is $f(\vec{x}, \vec{\beta}) = \beta_0 + \beta_1 \frac{1}{x_{dr}} + \beta_2 \frac{x_{pl}}{x_{dr}}$.

One way of automatically determining such a function is symbolic regression via genetic programming [9]. However, this is prone to overfitting, and compensating for that is still an active research area [10].

Fitted Function Sets (FFS) is a different approach that relies on detection of relevant parameters and multi-step regression. We have shown that this works well for energy model generation, while reducing the risk of overfitting [3]. Here, we briefly outline the concept, and refer to our prior work for details.

1) For each parameter x_i, partition S into subsets $\hat{S}_i$ in which all x_j with $j \neq i$ are constant, and subsets $\tilde{S}$ in which all variables are constant.
2) If the mean standard deviation of $\hat{S}_i$ is at least twice the mean standard deviation of $\tilde{S}$, x_i is relevant. If no variable is relevant, return the arithmetic mean $\mu(S)$.
3) For each relevant variable x_i, fit each function candidate on each partition $\hat{S}_i$, and select the one with the lowest loss, using the sum of squared residuals.
4) Build a model function $f(\vec{x}, \vec{\beta})$ using the power set of the selected functions, and fit it on S.

Going back to the radio example, the individual functions are $(x_{pl}, \vec{\beta}') \mapsto \beta_0' + \beta_1' x_{pl}$ and $(x_{dr}, \vec{\beta}'') \mapsto \beta_0'' + \beta_1'' \frac{1}{x_{dr}}$. The power set of $x_{pl} \mapsto x_{pl}$ and $x_{dr} \mapsto \frac{1}{x_{dr}}$, combined with fresh regression variables, is the model function:

$$f(\vec{x}, \vec{\beta}) = \beta_0 + \beta_1 x_{pl} + \beta_2 \frac{1}{x_{dr}} + \beta_3 \frac{x_{pl}}{x_{dr}}$$

C. Regression Trees

Regression Trees (also known as Classification and Regression Trees, or CART) have been introduced in 1984 and continue to be relevant to this day [11]. Formally, they express a piecewise constant function $f : \mathbb{R}^n \to \mathbb{R}$ by means of a binary tree. Each non-leaf node holds a binary decision such as $x_1 \geq 3$, and each leaf holds a value defining the function output for the configuration variable values or intervals described by the path from the root to the leaf.

Fig. 2 shows an excerpt from a simplified regression tree model for an air sensor's power consumption. If the sensor only performs temperature and humidity measurements ($x_{iaq} = 0$), the power is approximately constant. If it performs air quality measurements ($x_{iaq} = 1$), the power draw depends on measurement time (x_t) and internal heater temperature

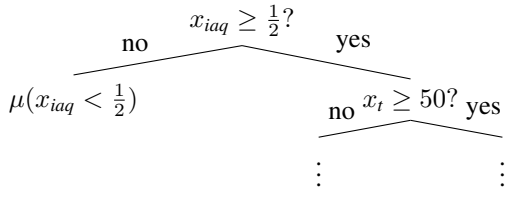

Fig. 2. Regression tree excerpt. Each leaf holds a constant model μ for the corresponding partial system configuration.

(x_{heat}, not shown). We note that CART work both with scalar and boolean variables (here, $x_{iaq} \in \{0, 1\}$).

Leaf values $\mu(\cdots)$ are the arithmetic mean of the corresponding observations. For example, $\mu(x_{iaq} < \frac{1}{2})$ is the mean of all measurements with $x_{iaq} < \frac{1}{2}$.

The key idea of CART generation algorithms is to try different binary splits of the observations and greedily choose the one yielding the highest model accuracy improvement. This is repeated recursively until a stop criterion is satisfied. We briefly outline the CART generation algorithm used in this paper, and refer to Breiman et al. for details [11].

1) Let $T_i = \{t_{i,1}, t_{i,2}, \dots\}$ be the set of unique values of variable x_i, with $t_{i,j} < t_{i,j+1} \forall j$.
2) If a stop criterion is satisfied: return a leaf node using the mean of observed data $\mu(S)$ as model value.
3) For each pair $(x_i, t_{i,j})$, split S into partitions $S_{i,j,\text{left}}$ (containing only samples with $x_i \leq t_{i,j}$) and $S_{i,j,\text{right}}$ ($x_i > t_{i,j}$).
4) Select the pair $(x_i, t_{i,j})$ with the lowest loss and transform it into a regression tree node "$x_i \geq \frac{t_{i,j}+t_{i,j+1}}{2}$?".
5) Repeat recursively with $S := S_{i,j,\text{left}}$ (left child node) and $S := S_{i,j,\text{right}}$ (right child node).

Common stop criteria are number of samples ($|S| < T$) or standard deviation ($\sigma(S) < T$) with a user-provided threshold T, or tree size and depth limits. Properly chosen, these minimize the risk of overfitting. Again, a typical loss function is the sum of squared residuals:

$$\sum_{y \in S_{i,j,\text{left}}} (y - \mu(S_{i,j,\text{left}}))^2 + \sum_{y \in S_{i,j,\text{right}}} (y - \mu(S_{i,j,\text{right}}))^2$$

We note that functions expressed by CART are well-defined. For any configuration $\vec{x}$, there is a path from the root node to an appropriate leaf node.

D. Linear Model Trees

Linear Model Trees (LMT) are an extension of CART. They also rely on regression trees, but use both static values and linear functions in leaves [5]. This allows them to express piecewise linear functions $\mathbb{R}^n \to \mathbb{R}$ and better capture configuration parameters with linear influence on the model value.

The LMT fragment in Fig. 3 describes memory requirements of an embedded AI application. If an AI architecture with batch processing support is used ($x_b = 1$), memory usage depends on batch size (x_{bs}). Otherwise, it is constant.

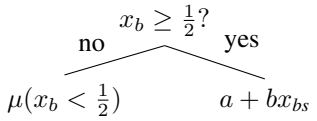

Fig. 3. A linear model tree. Each leaf holds a constant model μ or a linear model $a + bx_i + cx_j + \cdots$.

Model tree generation algorithms work by first building a CART, and then using a bottom-up *pruning algorithm* combined with linear regression analysis to transform subtrees into linear functions. Details vary; here we use an implementation of the M5' algorithm [12].

E. Tree Boosting

The last related approach, *Extreme Gradient Boosting* (an implementation of *tree boosting*), is an ensemble learning method. The model consists of a group (*forest* or *ensemble*) $\mathcal{F} = \{f_1, f_2, \dots f_K\}$ of regression trees instead of just a single one. Each tree may have a different structure. The model output is the sum of the individual tree models: $\mathcal{F}(\vec{x}) = \sum_{i=1}^{K} f_i(\vec{x})$.

Learning accurate ensembles from training data is considerably more challenging than for the previously presented single-function models. Essentially, Extreme Gradient Boosting (XGBoost) iteratively builds regression trees and adds them to the forest. However, its CART algorithm uses a custom loss function that greedily selects splits to reduce the forest's overall loss $\mathcal{L}$. Thus, tree generation takes the accuracy of the entire forest and not just the current tree into account.

$\mathcal{L}$ is a regularized loss function meant to penalize complex forests and thus reduce the risk of overfitting. Given a non-regularized function l, such as the sum of squared residuals, an ensemble $\mathcal{F} = \{f_1, \dots, f_K\}$, and observations $y_1, \dots, y_n$ of system configuration $\vec{x}_1, \dots, \vec{x}_n$, it is defined as follows.

$$\mathcal{L}(\mathcal{F}) = \sum_{i=1}^{n} l(\mathcal{F}(\vec{x}_i), y_i) + \sum_{k=1}^{K} \Omega(f_k)$$

The regularization term $\Omega(f)$ calculates the complexity of each tree f based on the number of leaves and leaf values. We refer to Chen et al. for details [13].

III. Regression Model Trees

While regression trees can learn piecewise constant or linear approximations of arbitrary functions, this leads to large models and has a considerable risk of overfitting, especially when faced with noisy data. Fitted function sets, on the other hand, cannot handle interdependent configuration parameters.

To address these issues, and obtain models that are both compact and accurate, we propose *Regression Model Trees* (RMT). These combine CART with fitted function sets to express a piecewise continuous function $f : \mathbb{R}^n \to \mathbb{R}$. Like CART and LMT, they utilize regression trees, but support both static values $\mu(\cdots)$ and fitted functions $f(\vec{x}, \vec{\beta})$ in leaves.

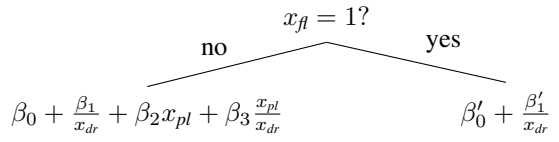

Fig. 4. Regression model tree excerpt. Non-leaf nodes express boolean decisions. Each leaf holds a fitted function set.

In order to obtain compact, easy-to-understand tree structures, we do not use LMT-style bottom-up pruning. Instead, we first generate a CART that only splits on relevant boolean variables (i.e., variables which have an effect on the standard deviation of the observation set S). Then, we use FFS to find and fit a function for each leaf. Given observations $S = \{y_1, \dots\}$ of parameter vectors $\vec{x}_1, \dots$, the algorithm is:

1) For each boolean variable x_i: determine whether x_i is relevant for the current set S (see Section II-B).
2) If a stop criterion is satisfied, or no boolean variable x_i is relevant: return a leaf node containing a fitted function set for S.
3) For each relevant boolean variable x_i, split S into partitions $S_{i,\text{left}}$ (containing only samples with $x_i = 0$) and $S_{i,\text{right}}$ ($x_i = 1$).
4) Select the variable x_i with the lowest loss and transform it into a regression tree node "$x_i = 1$?".
5) Repeat recursively with $S := S_{i,\text{left}}$ (left child node) and $S := S_{i,\text{right}}$ (right child node).

We use the same stop criteria as ordinary CART generation (see Section II-C).

Fig. 4 shows an example RMT for the transmission time of a radio module with configurable packet length and data rate, and a "fixed packet length" switch. If fixed length is enabled ($x_{fl} = 1$), data rate (x_{dr}) is the only variable affecting the transmission time. Otherwise, both data rate and packet length (x_{pl}) determine how long a transmission takes.

IV. Evaluation

We examine model accuracy (prediction error) and model complexity (number of tree nodes) of regression model tree (RMT), linear model tree (LMT), regression tree (CART), and tree boosting (XGB) models. We consider three questions.

Q1: How accurate is model prediction for parameter combinations that were not part of the training set? **Q2:** Can the models explain hardware behaviour at a level of complexity suitable for human interpretation? **Q3:** Do RMT reach a sweet spot of high accuracy and low complexity?

To answer these, we generate energy models for CC1200 and nRF24 radio transceivers and a BME680 air quality sensor. We use PPTA-based benchmark generation to perform measurements over the entire configuration space to avoid accuracy issues caused by selective data acquisition [6].

Additionally, we repeat each measurement five to ten times. This allows the model generation algorithms to distinguish intermediate model loss caused by parameter-dependent hardware behaviour from model loss due to measurement inaccuracies and fluctuations in hardware behaviour. To assess whether RMT are suitable for modeling properties other than energy, we also generate a non-functional property model for an embedded AI application.

For CC1200 and nRF24, we examine the duration (T) and power consumption (P) of the transmit state (TX), and the power consumption of the receive state (RX). We do not present results for sleep and idle states, as we found their power consumption to be independent of device configuration. RX and TX, on the other hand, are both the most energy-intensive and the most configurable device states. Configurable parameters are packet length, transmit power, data rate, and (for nRF24) fixed length packets, auto-retry, and auto-ack.

The BME680 sensor supports configurable oversampling of temperature, humidity, and pressure readings, as well as optional air quality (IAQ) measurements by means of a heated metal oxide layer with configurable measurement time and temperature. We examine the power consumption of its measurement state.

For the AI application, we examine inference throughput and memory usage as a function of AI architecture identifier, AI inference platform, batch size, quantization settings, and hardware platform.

A. Model Accuracy

We determine the generalization error (i.e., model error when predicting unseen configurations) by performing 10-fold parameter-aware cross-validation. To this end, we partition observations into training and validation sets not based on their position in the set of observations S, but on their configuration vector $\vec{x}$. This ensures that $\vec{x}_t \neq \vec{x}_v$ for any pair of training set entry (with configuration $\vec{x}_t$) and validation set entry ($\vec{x}_v$). So, the validation set only contains parameter combinations that were not part of the training set.

We use two additional models to put accuracy figures into perspective. The *static* model serves as upper bound for prediction error: it does not care about parameter vectors and simply uses $\mu(S)$ for prediction. The *LUT* (look-up table) model serves as lower bound. It partitions observations by parameter vector into sets $S_1, S_2, \dots$ so that $\forall k : \forall y_i, y_j \in S_k : \vec{x}_i = \vec{x}_j$. When the model value for a parameter vector $\vec{x}$ is requested, it selects the set S_k with matching parameter vectors, and returns $\mu(S_k)$. We do not use cross validation for the LUT model, as it is incapable of predicting unseen configurations.

We use the symmetric mean absolute percentage error (SMAPE) metric to determine model error. Given predictions $P = \{p_1, \dots, p_n\}$ and ground truth $Y = \{y_1, \dots, y_n\}$, it is defined as follows.

$$\text{SMAPE}(P, Y) = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|p_i - y_i|}{\frac{|p_i| + |y_i|}{2}}$$

Table I shows the mean model error for ten different pairs of training and validation sets. We see that RMT, LMT, CART, and XGB accuracy is nearly the same (and close to the lower error bound) in many cases. The only notable exception for

TABLE I
SYMMETRIC MEAN ABSOLUTE PERCENTAGE ERROR (SMAPE) OF STATIC, LUT, AND REGRESSION MODELS WITH 10-FOLD CROSS VALIDATION.

Attribute	Static	LUT	**RMT**	LMT	CART	XGB
BME680 P	82.5	2.4	5.5	4.7	6.6	5.5
CC1200 TX T	88.4	0.1	0.1	0.1	0.1	0.2
CC1200 TX P	12.5	0.1	0.8	0.8	1.3	1.1
CC1200 RX P	0.3	0.0	0.1	0.0	0.0	0.1
nRF24 TX T	103.2	0.1	0.4	20.2	0.2	0.2
nRF24 TX P	39.3	0.1	6.2	8.2	2.3	1.4
nRF24 RX P	2.2	0.0	0.0	0.0	0.0	0.1
AI Memory	79.9	1.1	18.1	41.7	21.0	16.9
AI Throughput	87.9	3.6	12.5	94.2	9.4	10.8

energy modeling is nRF24 TX power, where CART and XGB are better, while LMT perform worse. Still, the generalization error of RMT is no more than $6.2\,\%$.

In the (non-energy) AI use case, we see that both RMT and XGB achieve a low (but far from ideal) prediction error for memory usage, whereas CART and XGB are slightly better for throughput. In both cases, RMT achieve a generalization error of no more than $18\,\%$.

We do not show fitted function set results in the table due to lack of space. For all CC1200 attributes, and nRF24 RX P, they have the same model error as RMT. For all other attributes, they are incapable of generating a function model, and have the same prediction error as the static model. We found no case where conventional linear regression performed better than the fitted function set approach.

Regarding **Q1**, we conclude that regression model trees achieve a much lower energy model error than fitted function sets, and that regression model trees as well as CART and XGB achieve close-to-optimal model accuracy in almost all of our benchmarks. We find non-tree models to be unsuitable for energy models of highly configurable embedded devices.

Although RMT do not appear to be an ideal choice for complex non-energy models, they still achieve reasonable performance in that case. Next, we examine model complexity.

B. Model Size

Table II shows the model size after training on all measurements (without cross-validation). For energy models, we see that regression model trees are clearly the most compact, generating a single-node tree for CC1200 attributes as well as nRF24 RX power, and less than ten nodes in all other energy modeling use cases. We note that a single-node RMT is no different from a fitted function set model.

Linear model trees are slightly larger, with 10 to 20 nodes. As LMT approximate non-linear functions by means of linear functions and decision trees, this is not surprising. Both RMT and LMT are small enough to be understandable by humans.

CART, which must approximate both non-linear and linear functions by mean of piecewise constant tree structures, are five to 100 times larger than LMT. Forests generated by XGB are larger still. In this size range, with hundreds to thousands of tree nodes, attempting to understand hardware behaviour by looking at the tree structure is a futile endeavour.

TABLE II
SIZE (NUMBER OF NODES) OF REGRESSION TREE-BASED MODELS.

Attribute	**RMT**	LMT	CART	XGB
BME680 P	7	13	959	5,398
CC1200 TX T	1	15	379	1,646
CC1200 TX P	1	11	419	1,716
CC1200 RX P	1	13	69	52
nRF24 TX T	7	15	1,359	2,754
nRF24 TX P	5	13	1,823	5,254
nRF24 RX P	1	17	141	80
AI Memory	1,706	13	10,381	79,192
AI Throughput	1,395	13	10,381	42,028

Overall, for **Q2**, we find that RMT and LMT can explain energy behaviour at a level of complexity suitable for humans, while CART and XGB cannot. For non-energy attributes, although RMT are ten times smaller than CART and XGB, only LMT are small enough.

C. RMT Performance

When modeling energy behaviour, RMT exhibit a generalization error of less than $7\,\%$, using trees with less than ten nodes and just one to four different variables in leaves. While there is one case where increased model complexity leads to higher accuracy (nRF24 TX power), this is not a general finding. For instance, no matter whether using seven or more than 5,000 nodes, all four modeling methods have difficulties when faced with BME680 power.

Thus, for energy models, we answer **Q3** with a definitive Yes: RMT have the lowest complexity of all evaluated modeling methods, while still providing high (and, in most cases, close-to-optimal) accuracy.

For non-energy models, LMT are least complex, but also by far the least accurate method. Although RMT achieve reasonable accuracy, at more than 1,000 tree nodes, they are not well-suited for human analysis. So, we may not have found the sweet spot for non-energy models yet.

V. RELATED WORK

We have already examined fitted function sets [3], regression trees [11], linear model trees [14], and extreme gradient boosting [13]. In our opinion, next to regression model trees, these are the most useful energy modeling methods for embedded devices.

CART and LMT are often used for modeling non-functional properties of software product lines [15]. For example, a boolean-only CART variant is capable of modeling various attributes of products such as the apache web server or the x264 video encoder with an error of less than $10\,\%$, while requiring only a small amount of benchmark data [4]. However, its data-efficient sampling method was designed for boolean-only configuration spaces, and may therefore not be suitable for energy models.

Similarly, linear model trees have been used to predict software faults based on software quality attributes with 5 to $50\,\%$ model error [16].

Fourier Learning is an entirely different approach [17]. In contrast to the previous methods, it provides guaranteed accuracy bounds, but at a cost: it requires a large amount of samples, and the resulting models are unsuitable for interpretation by humans. The exact number of required samples is a function of the desired accuracy bound.

L2S, on the other hand, automatically detects correlations between configurable features and hardware (or workload) changes, and generates benchmark configurations accordingly. Jamshidi et al. were able to achieve a model error of 7 to 20 % when modeling the latency of an AI application [18]. Their approach uses CART for latency prediction, but has only been tested with boolean configuration parameters.

Finally, Cherifi et al. propose using *Changepoint Detection* to automatically generate state machines for embedded device components, and handle parameter changes by means of state transitions (i.e., one state for each parameter configuration) [19]. This concept is similar to CART, and appears to have the same drawbacks as well: generated models are accurate, but quickly become so complex that they no longer offer an intuitive grasp of hardware behaviour.

VI. Conclusion

We have presented *Regression Model Trees* (RMT), a combination of regression trees and automatically-generated regression functions. In an evaluation with three highly configurable embedded devices, we have shown that energy models using regression model trees are both accurate (with a maximum generalization error of 6 %) and compact (less than ten tree nodes and simple regression functions). This makes them an ideal choice for energy models that offer an intuitive understanding of how configuration options affect hardware behaviour. While RMT accuracy is also reasonable for non-energy attributes, they have significant complexity in that case.

Additionally, we have found that regression trees (CART) and gradient-boosted forests (XGB) can be more accurate than RMT in some cases, but are also considerably more complex. While unsuitable for human-readable energy models, they may be useful when highest accuracy is needed.

Acknowledgments

This work was partially supported by funds of the Federal Ministry of Food and Agriculture (BMEL) based on a decision of the Parliament of the Federal Republic of Germany via the Federal Office for Agriculture and Food (BLE) under the innovation support programme.

References

[1] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside, "Modelling and multi-objective optimization of quality attributes in variability-rich software," in *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, ser. NFPinDSML '12. New York, NY, USA: Association for Computing Machinery, 2012. [Online]. Available: https://doi.org/10.1145/2420942.2420944

[2] B. Snajder, V. Jelicic, Z. Kalafatic, and V. Bilas, "Wireless sensor node modelling for energy efficiency analysis in data-intensive periodic monitoring," *Ad Hoc Networks*, vol. 49, pp. 29 – 41, 2016.

[3] B. Friesel, M. Buschhoff, and O. Spinczyk, "Parameter-aware energy models for embedded-system peripherals," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, June 2018, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8442096

[4] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, and H. Yu, "Data-efficient performance learning for configurable systems," *Empirical Softw. Engg.*, vol. 23, no. 3, p. 1826–1867, Jun. 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9573-6

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

[6] M. Buschhoff, B. Friesel, and O. Spinczyk, "Energy models in the loop," *Procedia Computer Science*, vol. 130, pp. 1063 – 1068, 2018, the 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918305167

[7] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, 1998.

[8] G. Berthou, K. Marquet, T. Risset, and G. Salagnac, "Accurate power consumption evaluation for peripherals in ultra low-power embedded systems," in *2020 Global Internet of Things Summit (GIoTS)*, 2020, pp. 1–6.

[9] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[10] C. Raymond, Q. Chen, B. Xue, and M. Zhang, "Genetic programming with rademacher complexity for symbolic regression," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 2657–2664.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 1984.

[12] Y. Wang and I. H. Witten, "Induction of model trees for predicting continuous classes," 1996.

[13] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[14] D. Malerba, F. Esposito, M. Ceci, and A. Appice, "Top-down induction of model trees with regression and splitting nodes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 5, pp. 612–625, 2004.

[15] J. A. Pereira, M. Acher, H. Martin, J.-M. Jézéquel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," *Journal of Systems and Software*, vol. 182, p. 111044, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221001412

[16] S. S. Rathore and S. Kumar, "A decision tree regression based approach for the number of software faults prediction," *ACM SIGSOFT Software Engineering Notes*, vol. 41, no. 1, pp. 1–6, 2016.

[17] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki, "Performance prediction of configurable software systems by fourier learning (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 365–373.

[18] P. Jamshidi, M. Velez, C. Kästner, and N. Siegmund, "Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 71–82. [Online]. Available: https://doi.org/10.1145/3236024.3236074

[19] N. Cherifi, T. Vantroys, A. Boe, C. Herault, and G. Grimaud, "Automatic inference of energy models for peripheral components in embedded systems," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2017, pp. 120–127.