Deep Green: modelling time-series of software energy consumption

Stephen Romansky, Neil C. Borle, Shaiful Chowdhury, Abram Hindle, Russ Greiner
Department of Computing Science
University of Alberta
Edmonton, Canada
Email: {romansky,nborle,shaiful,hindle1,rgreiner}@ualberta.ca

***Abstract*—Inefficient mobile software kills battery life. Yet, developers lack the tools necessary to detect and solve energy bugs in software. In addition, developers are usually tasked with the creation of software features and triaging existing bugs. This means that most developers do not have the time or resources to research, build, or employ energy debugging tools.**

We present a new method for predicting software energy consumption to help debug software energy issues. Our approach enables developers to align traces of software behavior with traces of software energy consumption. This allows developers to match run-time energy hot spots to the corresponding execution. We accomplish this by applying recent neural network models to predict time series of energy consumption given a software's behavior. We compare our time series models to prior state-of-the-art models that only predict total software energy consumption. We found that machine learning based time series based models, and LSTM based time series based models, can often be more accurate at predicting instantaneous power use and total energy consumption.

I. Introduction

Developers are relatively unaware of techniques and tools used to address software energy issues [1]. Energy bugs deplete mobile batteries and cost developers time to debug. Battery life is an important resource for end-users because it dictates how long their applications can be used. To meet the end-users' demand for energy efficient applications, developers are more energy aware than ever, and ask more questions about their software's energy efficiency [1]. While answers to their energy related questions help developers to some extent, in order to build energy efficient applications, developers need actual feedback on their applications' energy consumption. Researchers have built tools for developers to model the energy consumption of mobile software and hardware components [2], [3]. However, these tools are not as easy to use as widely available performance metrics like: CPU load, network interface usage, or disk usage. We are interested in building tools that help developers to understand when and where their applications consume the most energy. Such tools would enable the software development processes to better track and maintain energy goals by finding hotspots and energy bugs [4], [5].

Previous energy estimation tools [3], [6], [7] offer a summary of the total energy consumption for a given run of the application. Although this gives an idea about an app's energy consumption, it does not give much about the location of the energy bugs and hotspots [4], [5]—which segment of source code is responsible for the most energy drain. Developers provided with tools to align time series of software energy consumption to program behaviour can more easily debug energy bugs.

Researchers have worked towards online energy estimation tools [8], [9]. However, current instantaneous energy prediction tools assume that developers have access to battery API information or that developers have hardware access to power measurement tools. Whereas, our work makes no hardware assumptions and provides developers with hardware-free energy usage predictions as if the developers had access to energy meters.

In this paper we propose a series of simple to measure and construct models that use software performance measurements to predict instantaneous energy consumption. We can align performance measurements with our energy measurements to train neural networks and other machine learning models to predict the instantaneous energy consumption of a software application.

We have to assess the quality of our proposed energy consumption prediction method compared to existing solutions [10]. Our method uses instantaneous time measurements throughout a software test run whereas existing state-of-the-art models measure a summary of software features at the beginning and end of a software test run. We call the existing models *time-summarized*, or *1step* for short, as the software run is measured or aggregated in a single time step instead of many instantaneous time steps or windows.

To assess our models we raise the following questions: (RQ1) How well do time series models perform when predicting total energy consumption?; (RQ2) Do time series models with built-in state or memory perform better than time series models without state when predicting energy consumption?; (RQ3) Energy prediction models can be trained with different feature sets; how does performance change when using them individually versus together? What features affect the energy consumption models predictive power?; and (RQ4) Do shallow multi-layer-perceptrons perform similarly to existing stateless models such as those based on linear regression?

II. Prior work

Energy-aware software engineering poses many challenges in the development of energy-efficient software [11], [12], [7].

The research approaches to profiling energy on given mobile devices can be partitioned into three categories: software, hardware, or hybrid models [3]. Software based energy profilers use recordable features like API calls, system calls, and resource utilization to model the energy consumption of a mobile device [3], [13], [14], [6]. Whereas, hardware based energy profilers can measure the energy consumption of specific components [3], [15]. Hybrid approaches use both software and hardware based features [3]. Furthermore, a software based profiler might rely on an API to the battery to measure energy consumption on the device during model construction [3], [13]. The measurement of the energy consumption through an API is distinct from a hardware based profiler, which can sample the current used by the phone with an external measurement tool [3]. Here are two examples of energy profilers: *se-same* [13] is a smart-battery interface based application energy consumption predictor; therefore, it is a software based profiler [3]; and *Netw-trace* [15] which uses an external device to measure the energy used by the phone, and software to record the network traffic on the device's WiFi and radio devices [3]. *Netw-trace* is a hybrid profiler, because it uses software and hardware reading to predict energy consumption. Work by Banerjee et al. [4] and the *GreenMiner* [16] are examples of hardware and software based energy measurement tools used to study software execution and energy consumption.

Researchers have also investigated energy draining hardware and software components that software applications use. For instance, researchers have developed energy-efficiency targeted operating systems like *ErdOS* [17] and *CondOS* [18], [2]. *ErdOS* [17] predicts how a user interacts with hardware components to schedule more sleep behavior. Whereas, *CondOS* provides more energy efficient APIs for sensing resources like GPS so that developers do not need to develop their own energy efficient heuristics. More energy efficient networking protocols for battery limited mobile devices have also been studied since WiFi and radio components can consume large amounts of device energy [2], [19] Furthermore, researchers have also investigated the process of sharing mobile computation with the cloud [2], [17]. This enables a phone to place energy-expensive computation on another computer to prevent the battery from being drained by a costly operation.

Work has been accomplished on identifying software system calls as a method to profile the energy consumption of software [20]. Tools that predict the total energy consumption of software applications exist which are based on system calls and additional software and hardware features [10], [21]. However, these tools often do not provide details of the software behavior that influences energy consumption. *GreenOracle* is a regression tool which can predict the total energy consumption of a software application test [10]. It uses system call and resource utilization metrics to predict the energy consumption of given software applications under test. In this paper we employ *GreenOracle* features and models to produce time series predicting energy models.

Prior work has also worked towards creating online energy profiles of software applications similar to our goal [8], [9]. However, prior work has used hardware APIs like on the Android battery to perform online energy prediction [9]. Other prior work has also assumed there is access to an energy measurement platform when predicting how much energy is consumed by software [8]. However, our work does not assume that the developers using our models have access to a battery API or that developers have access to energy measurement hardware. Therefore, developers can use our prediction models to generate the energy consumption information of their applications under test as if they had access to an energy meter of their own.

III. Background

A. System Calls and The Process File System

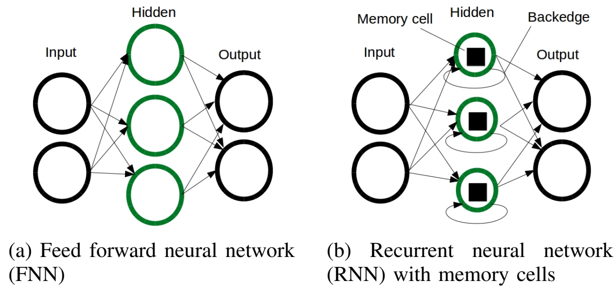
System calls are generated by Linux applications when requesting system resources [22]. For example, the *write* system call count explains how many times the processes have written to a file descriptor. Prior work has shown that energy models can be constructed from using system calls as input features [23], [10], [20], [13].

The process file system (*procfs*) provided by the Linux kernel provides process-specific run-time information [24]. For example, the process *utime* feature records how long a process has been scheduled in user land in clock ticks. The *procfs* provides additional resource-usage features associated with a running application that we can use to predict the energy consumption of a given application. It was also shown in prior work that using both system calls and process utilization features can create accurate models [13], [10].

B. Energy and Energy measurement

Energy (E) is the capacity to do work measured in *joules*. It is represented by power (P), which is the rate work is done, multiplied by time (T): $E = P \cdot T$. Power is measured in *watts*. In our case, energy is consumed by the smart phones to execute software applications.

We make use of the *GreenMiner* testbed: a set of 4 instrumented Android devices that replay software test runs while recording device energy usage [16]. We apply the *GreenMiner* to mine information from multiple Android applications versions to build a model-training corpus. To pick an application for energy profile mining on the *GreenMiner* a researcher needs to check if the application and each of its versions run on the devices. As well, the researcher needs to write software tests which simulate how an end-user would interact with the application. This means a software test needs to cover important functionality from the application under test. It is possible to see how the energy consumption of tested functions change over time because the software tests are repeated across multiple revisions of the same software application.



(a) Feed forward neural network (FNN)

(b) Recurrent neural network (RNN) with memory cells

Fig. 1. Feed forward and recurrent neural networks

C. Neural networks

Neural networks are currently quite successful models in machine learning capable of modelling linear and non-linear relationships. Successes have been demonstrated in fields such as speech recognition [25], [26], time series prediction [27], and even software engineering [28], [29], [30].

Deep neural networks are neural networks with multiple layers in contrast to shallow networks which have few layers. Deep learning is the process of training a model structure to represent a hierarchy of abstract relationships from a given data set [31]. It avoids the need for manual feature selection because the training algorithms for deep networks perform feature selection. Deep learning is also capable of exploiting non-linear relationships in the input data.

Common types of neural networks like feed forward neural networks (FNN) and recurrent neural networks (RNN) have nodes with specific edge configurations. FNNs are less complicated than RNNs because FNN nodes and edges do not contain loops. Both FNN and RNN networks have 1 or more layers, each with 1 or more nodes. In an FNN, the edge is restricted such that their node outputs can only go to lower layers. Whereas, RNNs allow edges from node outputs to be connected backwards or forwards. Figure 1a shows an example of a multi-layer perceptron (MLP) which is a type of FNN. The direction of the edges in the MLP is forward from left to right in contrast to the RNN in Figure 1b which has edges going in both directions.

RNNs are capable of learning more relationships from input data than FNNs due to their back-edges. Back-edges refer to the looping edges of the network as shown in Figure 1b because they go back from a layers output to somewhere higher in the network. Yet, RNNs have poor performance when used with time series of length 200 or more time steps [32]. This is due to the network nodes becoming saturated which leads to very large or very small parameters (edge weights and biases). However, this can be managed by adding memory cells to the RNN nodes. Popular networks that use memory cells are the long-short term memory layers (LSTM). LSTMs avoid the part of the saturation problem faced by RNNs. This makes LSTMs better at modelling the relationships that exist in sequential input data than RNNs and FNNs. Both RNN and LSTM networks have achieved strong results on tasks that require modelling sequences of inputs which motivates our exploratory usage of the models in this work [31], [33].

Neural networks use activation functions in each neuron to transform the output of the network and to update the weights at each neuron. In our work we apply linear, sigmoid, hard sigmoid, and tanh activation functions. We use the linear activation function to model Gaussian distributions, and we apply the tanh and hard sigmoid activation functions in our LSTMs to model the sequence-based relationships of our input measurements [34]. We use the cross-entropy cost function with our networks as it has been found to yield good results in the training process due to training becoming a convex optimization problem [32].

D. Glossary

Here we introduce several terms that we use throughout the paper: a *time series* refers to a sequence of measurements or predictions grouped into time steps by time of measurement/prediction. We refer to models that have been constructed to consume *time series* and output *time series* predictions as *time series* models. If a model is unable to handle the multiple measurements across time, then we remove time by aggregating via summation every measurement together. We refer to the *time-summarized* data as a single time step or *1step*. Because, the time series only contains information aggregated into a single time step which covers from the beginning of a software test run to the end of its duration. We refer to software tests, where we observe and collect our data, as *test runs*.

IV. Methodology

The steps of the methodology used to evaluate the energy models are:

1) Find applications with multiple versions for Android.
2) Write test cases to simulate user interaction with the applications.
3) Run application tests on the GreenMiner and collect data.
4) Create cross folds: 5 applications train; 1 application test.
5) Train models, or perform parameter selection, per fold.
6) Evaluate models on test folds.

We want to make energy models that can predict the energy consumption of software as it runs. Such models would enable developers to associate their software behavior with corresponding changes in energy consumption. To train energy prediction models, however, we need examples of running software and their resource and energy usage.

We used the *GreenMiner* testbed, described in Section IV-A, to collect time series information from Android applications concerning *system calls* from *strace* and *process resource utilizations* from *procfs*. To use the GreenMiner we identified Android applications with multiple versions; we write test cases to simulate user interaction with the applications; and we record information from the tests while they run on the GreenMiner.

TABLE I
MINED APPLICATIONS

Application	Description
Calculator	An Android calculator application
Blockinger	A Tetris like Android game
Dalvik Explorer	A phone properties viewer for Android
2048	A number puzzle game
Pinball	An Android pinball game
Memopad	A freehand drawing application

The collected *system calls* provide information about software resource requests over time. However, system calls do not provide any information on CPU usage and other relevant information like number of interrupts, number of context switches etc. This led us to periodically access and collect those measurements from the Linux *procfs* throughout a software test. We also used hardware instrumentation to collect the energy consumption of mobile devices as they run software tests. In Section IV-B1, we discuss the various models that we evaluate in our work and the reasons why they were chosen. In Section IV-B2 we discuss how the models are compared with one another to answer our research questions.

A. Data Collection

Section IV-A1 discusses how behavior and energy consumption profiles are extracted from software applications. Section IV-A2 explains how we collect behavior and energy measurements during the mining process of Section IV-A1. Section IV-A3 shows preprocessing steps taken prior to training.

1) Mining Applications: In order to train our models, we needed example applications so that we can run them with a test case, collect their resource usage (as model input) and energy consumption (as model output) periodically. We selected 6 Android applications from the *GreenOracle* dataset, as listed in Table I (step 1). Each application had multiple software revisions available and the applications are open source software. Each application had at least 30 unique software revisions, except `Dalvik Explorer` which only had 13 software revisions. 30 was chosen arbitrarily to provide enough statistical power for many of the distribution comparison tests and to keep measurement time reasonable. Having multiple versions of each application was good for models' accuracy. We needed to write only one test case for a particular application, but could enlarge our training data by running the same test case for the multiple versions (step 2). We only chose a subset of the *GreenOracle* dataset because it was time consuming to collect time series of the system call and *procfs* features (step 3). We tried to pick applications that used the phone differently than one another to provide a representation of different hardware usage per application. In addition, with a larger number of applications, the training time of our deep networks is significantly longer.

To measure energy consumption we wrote test runs which simulate common use cases of the applications (step 2). The common use case of an application was assessed by the authors to determine how the application is intended to be used. For example, test runs were written to do small calculations, like finding the GST of purchased goods for the Calculator application, because this is what we would expect an average user to do with a pocket calculator. Dalvik explorer displayed phone meta-data. Blockinger's test dropped and rotated tetris pieces randomly. 2048's test made random moves. Pinball's test randomly throws and paddles the pinball. Memopad's test draws a hexagon monster with legs. To reduce the generalization error of our measurements we repeat the test runs multiple times per revision to create multiple data sets for the same revision. In our study we repeated test runs 20 times each matching the methodology of other works [10].

2) Mining Features and Labels: The software features we are interested in include: system calls, and device resource utilization (step 3). Similar to the *GreenOracle* [10], we used the *strace* program to collect all the system calls and retrieved measurements from `/proc/stat`, `/proc/pid/stat`, and `/proc/pid/statm` file systems to collect process related information. However, in contrast to *GreenOracle*, all of our features were collected periodically throughout the duration of a software test run.

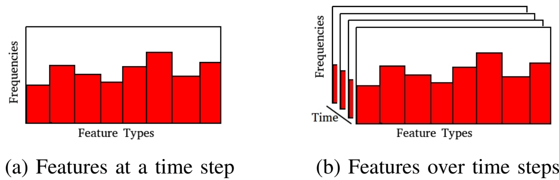
To measure resource usage for a given application over time, we define a time step to be a period of 0.02 seconds. The period of 0.02 is chosen because this is similar to the sampling rate of the *GreenMiner*. Figure 2a shows an example of what our model features look like in a given time step. In our energy models, after we have recorded an applications resource usage and system calls we partition the information into time steps. For each period of 0.02 seconds, we count the number of each system call that occurs in the time step, then we approximate additional applications resource utilization from *procfs* such as CPU load.

We also collect a power sample, which is a single watt measurement for the period, that is paired with the software-behavior measurements. We convert the watt measurement to energy usage using the the duration between two watt samples which will be used as the dependent variable in our models.

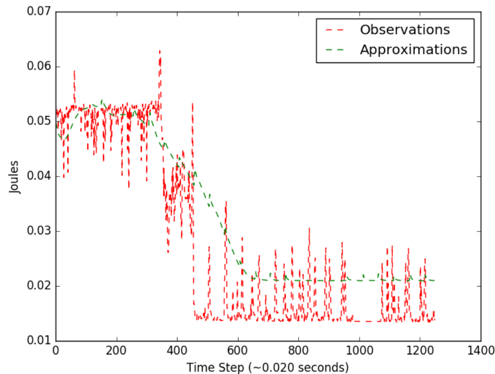
We collect the measurements and energy usage periodically throughout a software applications usage. We then synchronize the data measurements with energy usage to create partitions, or time steps, of the data for training as in Figure 2b. This provides one time step for every 0.02 seconds, and the corresponding energy measurements as shown in Figure 2c. We preprocess these timeseries for training our energy models.

3) Preprocessing: We time align and bin our measurements for training. The software and energy measurement tools in our experiment are not aligned with respect to time (step 4). Therefore, for the *procfs* and energy measurements that we collect we apply linear interpolation to approximate measurements at fixed times. Fixed times allow us to partition our time series measurements into time steps which can be used by the model as inputs. The system call measurements which we collect are counted for each of the fixed time periods to create time steps. We collect every feature reported by *strace* and the selected *procfs* files.

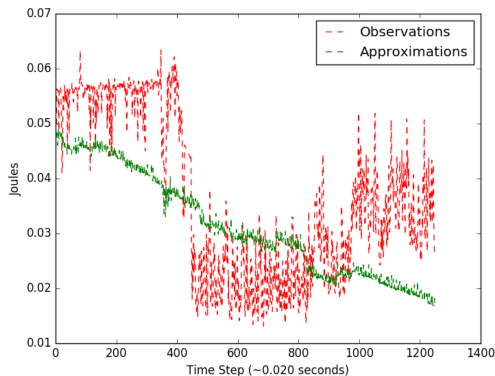
We also take the difference of consecutive samples of



(a) Features at a time step (b) Features over time steps



(c) Observed energy usage (red) over time steps with 2 layer LSTM with Sigmoid output layer predicting energy consumption (green) on Green Oracle sample



(d) Observed energy usage (red) over time steps with Linear kernel SVR model predicting energy consumption (green) from Green Oracle features

Fig. 2. Figure (a) and (b): feature set measurements. Figure (c), and (d) show measured energy over time compared with LSTM and SVR model predictions that are described later.

procfs counters. Because, a single sample will only explain the current values of given resources used by a process. Whereas, the difference explains the rate of change in the resource that was caused by the running process. The *procfs* features that do not change throughout the duration of a test run are discarded.

With system calls, it is possible to have 2 semantically similar system calls occur in separate applications. We provide a list of the system call groups that we apply to our data set in Table II. Duplicate names create complications when tuning the weights in machine learning models; because, the training algorithm tries to attribute weight to multiple features causing the same effect in the models. We use a subset of the system call groupings as in the *GreenOracle* for comparison of our time series models against *1step* models [10]. An example of semantically similar system calls with different names are: *fstat64* and *stat64* which both get file status information.

TABLE II
SYSTEM CALL FEATURE GROUPINGS

Group	System calls
lseek	lseek, _llseek
write	write, pwrite
read	read, pread
stat	fstat64, lstat64, stat64
fsync	fdatasync, fsync

To improve the training times of our models, and to try and avoid gradient problems in the neural networks models we train, input normalization is used in our preprocessing step. We chose to use min-max scaling for each approximated *procfs* measurement, system call count, and approximated energy usage values. For each software feature i in our data set, we calculate the scaled values $\boldsymbol{X}_{scaled_i}$ from the given software feature $\boldsymbol{X}_i$ using the following formula:

$$\boldsymbol{X}_{scaled_i} = \frac{\boldsymbol{X_i} - min(\boldsymbol{X}_i)}{max(\boldsymbol{X}_i) - min(\boldsymbol{X}_i)}$$

Feature normalization scales each features to the range of $[0, 1]$. Normalization is used to prevent the weights in models like neural networks from growing to infinity or shrinking to 0 known as the exploding or vanishing gradient problem. We also apply normalization to our energy measurements for model training. When training our models, input normalization tended to improve prediction performance.

B. Models

Section IV-B1 provides the list of models we consider in our experiments. It also discusses the difference between models that consider a time series and models that are *1step* (step 5). The statefulness of LSTM models and whether or not that improves performance is also discussed. Section IV-B2 explains how it is possible for us to compare the time series models with the *1step* models (step 5).

1) Model Selection: Tools for predicting the time series energy consumption of software are not widely available at the time of writing, although models do exist [23] or some models could be converted to do so [13], yet implementations are not available. Therefore, we do not have time series models that we can compare our work against. However, there are models that predict the total energy usage of software components or hardware devices such as linear regression models of energy consumption [10]. We use the same performance metric as the prior work to make our models comparable to the existing state-of-the-art.

Because it is unknown which machine learning models could perform well, predicting the time series energy consumption, we select several. We evaluate ridge and lasso regression, and multiple support vector machine regressors based on the results of the prior work [10]. However, none

of the models from the prior work are designed for taking advantage of the sequence in which data in a time series occur. Therefore, we also evaluate two different LSTM RNNs because we think that they will perform better on the time series energy prediction task. We also evaluate several MLPs because we think that they will perform similarly to the ridge and lasso regression models due to the number of weights in a single layer MLP being similar to the number of weights in a linear regression model.

We evaluate all of the models with 4 different feature sets to get an understanding of how the models perform. We use a full feature set which consists of all the features we collected from *system calls* and *procfs*, a *GreenOracle* feature set which consists of the best linear features as selected in the prior work [10], a *system call* feature set which only consists of system call features, and a *procfs* feature set which only consists of the process related features.

For the MLP and LSTM RNNs, we rely on stochastic gradient descent to perform model tuning. We select specific structures, in the case of the MLPs we look at: a sigmoid hidden layer with a linear output, two sigmoid hidden layers with a linear output, and three hidden sigmoid layers with a linear output. On the deep neural networks, we consider the use of one and two hidden LSTM layers with either a linear output layer or a sigmoidal output layer. Our neural network input and hidden layers contain one node per feature being evaluated by the model. For example, a 2 layer MLP evaluated on the GreenOracle feature set would have 14 input nodes, 14 nodes on the hidden layer, and 1 node on the output layer.

Furthermore, the selection of appropriate MLP and RNN LSTM structures is further complicated by neural network hyper parameters like number of nodes, number of layers, and activation functions. Capacity of a model is associated with how much information the model can learn and store from the training data. Capacity is associated with the number of edges, neurons, and whether or not the model neurons contain memory cells. It is not clear if more or less capacity would help with the learning task on the MLP and our deep learning LSTM models. It has been found that linear outputs can model Gaussian distributions and our physical measurements of the mobile devices often have Gaussian distributions [32]. So, we evaluate several types of neural network structures to get a better understanding of the problem which we are approaching. A list of the models is provided in Table III.

2) Model Comparison: We investigate how models perform when trained with each time step, and when we ignore time in the data set by aggregating our time series into a *single time step* (step 6). This means, we have a set of models which predict every instantaneous time step of a time series. The models use program resource usage to predict energy consumption. We train models to predict, from a whole time series, the total energy consumption of the application. Prior works evaluated performance using data sets similar to the *single time step* models [10]. We also use the mean relative error as the loss function. This is calculated by checking if the predicted total energy usage for an application is similar to the observed total energy usage for the application.

TABLE III
TIME SERIES AND TIME-SUMMED *1step* MODEL NAMES AND THEIR DESCRIPTIONS

Time Series Model	Description
MLP_1_layer	1 layer MLP
MLP_2_layer	2 layer MLP
MLP_3_layer	3 layer MLP
lstm3sigm	2 layer LSTM and sigmoid output
lstm2sigm	1 layer LSTM and sigmoid output
lstm3line	2 layer LSTM and linear output
lstm2line	1 layer LSTM and linear output
linear_regressor	lasso regression model
ridge_regressor	ridge regression model
svr_poly	polynomial SVR
svr_rbf	radial-basis-function SVR
svr_plain	linear SVR
1step Model	Description
1step_MLP_1_layer	1 layer MLP
1step_MLP_2_layer	2 layer MLP
1step_MLP_3_layer	3 layer MLP
1step_linear_regressor	linear regression model
1step_ridge_regressor	ridge regression model
1step_svr_poly	polynomial SVR
1step_svr_rbf	radial-basis-function SVR
1step_svr_plain	linear SVR

Since, we use the same loss function in the instantaneous time step and *single time step* formats of the data set, we are able to compare models trained on the separate contexts with one another because they are evaluating the total predicted joules. It would not be possible to compare the error of the per-time-window energy predictions of the time series models to the total energy predictions of the *1step* models. The loss function is given below, where $\boldsymbol{Y}$ denotes the energy observations per-time-window of the time series in joules and $\hat{\boldsymbol{Y}}$ denotes the energy predictions in joules for each time step in a series.

$$f(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \left| \frac{\Sigma \hat{y} - \Sigma y}{\Sigma y} \right|$$

The models which predict each time step of a given series will produce a normalized energy prediction, we denormalize the prediction and denote it $\hat{y}$, for each step of the series. Whereas, in the case of the *1step* models which take the whole time series and predict the total energy consumption, in normalized joules for an application test run, there is only one predicted $\hat{y}$ value which we denormalize for evaluation. Furthermore, we test several different feature sets for comparison of our model performance. We test a full feature set (80 features), which consists of all system calls and process related features collected from the applications; a replica of the *GreenOracle* feature set (14 features), which consists of a subset of the full feature set; a feature set based only on the system calls (61 features); and a feature set based only on the process related features (19 features). The *GreenOracle* model used linear methods to perform feature selection, whereas our stateful models perform their own version of feature selection between features. So, if the full-feature set performs remarkably better than the *GreenOracle* feature set, we would expect a non-

TABLE IV
SVR PARAMETERS

Parameter	Coefficients
Penalty	$2^{-5}, 2^{-3}, 2^{-1}, 2^{1}, 2^{3}, 2^{5}, 2^{7}, 2^{9}, 2^{11}, 2^{13}$
Gamma	$2^{-15}, 2^{-13}, 2^{-11}, 2^{-9}, 2^{-7}, 2^{-5}, 2^{-3}, 2^{-1}, 2^{1}, 2^{3}$
Degree	2

linear relationship to have been identified. We investigate the system call and process feature sets to determine if combining the two feature sets improves the models accuracy.

We use 6-fold leave-one-out analysis on our 6 applications to train multiple instances of our models (step 5). The data collected from each application is used to build a holdout fold. The models are evaluated on the respective holdouts (step 6). We calculate the effect-size of a representative sample from the training set and use it to train each SVM model. This evaluation format was chosen because it will explain if a particular model performs well for a given feature set.We compare model performance by combining all of the folds for comparison of each model.

For the SVR models [35] which we train, we perform parameter selection using 5-fold cross validation on each fold. Table IV shows the list of hyper parameters which we use for parameter selection. The *Gamma* coefficients are used for radial basis and linear kernels, and the *degree* coefficient is used for a polynomial kernel. SVRs are applied to the full and *GreenOracle* feature sets only, as these are combined feature sets of the system-call-only and processor-only feature sets.

V. EVALUATION

We provide our models with the time series of recorded software behavior for given software test runs. The time series for the *1step* models are summarized because the *1step* models ignore the time dimension of the series. Each of the multistep models will predict a series of energy predictions for each of the input time steps. The models predict the energy consumption of each step of the time series or cumulatively the energy consumption for the whole series.

Before it is possible to compare all of the models, we have to perform hyper parameter selection for our SVR models. The SVR training parameter selections are shown in Table V. The Full feature set was chosen because it might contain non-linear relationships which the RBF or polynomial kernel SVR could identify, and the *GreenOracle* feature set was chosen due to the previous success with this feature set [10].

We want to know how our models compare against one another on the task of predicting normalized joules from software behavior. We evaluate our models by denormalizing the joule predictions such that we can compare energy usage across the $1step$ and time series models. With joules, we can compare our prediction accuracy to the prior works [10].

We had numerous models and thus we wanted to see if models were significantly different in their performance. To check whether or not our models produced normal distributions for the data being predicted, we used the Anderson-Darling test.

TABLE V
SELECTED SVR PARAMETERS AND FEATURE SETS

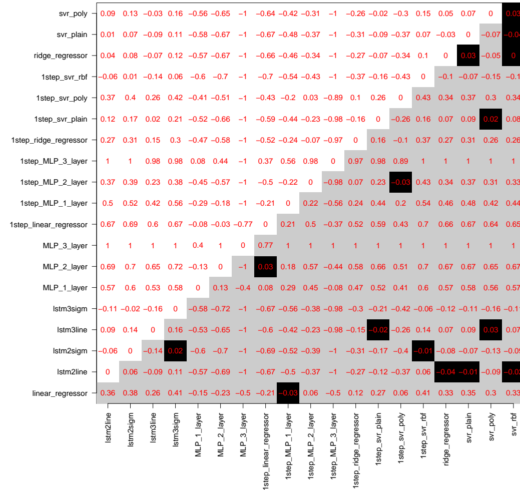
Model	Feature set	Penalty	Gamma	Degree
linear	Full	0.125	0.0125	-
rbf	Full	2	8	-
polynomial	Full	2048	-	2
linear	GreenOracle	0.03125	0.0714	-
rbf	GreenOracle	0.5	8	-
polynomial	GreenOracle	8	-	2
*1step*_linear	Full	0.125	0.0125	-
*1step*_rbf	Full	2	8	-
*1step*_polynomial	Full	2048	-	2
*1step*_linear	GreenOracle	0.125	0.0714	-
*1step*_rbf	GreenOracle	0.5	8	-
*1step*_polynomial	GreenOracle	32	-	2

However, we found p-values below $2e{-}16$ which confirmed that none of the distributions under consideration are normal. Thus, non-parametric tools are used to assess the relationships between the models. A Kruskal-Wallis test on each feature set shows a p-value $< 2e{-}16$ which implies that the model type has a significant influence on the output. Thus the models are mostly different from each other and we can show this with the pairwise Wilcoxon test with Holm p-value correction for multiple comparisons. The Wilcoxon and Cliff's Delta methods do not make assumptions about the data distributions being evaluated and the two tools can inform us whether or not two models perform significantly different from one another.

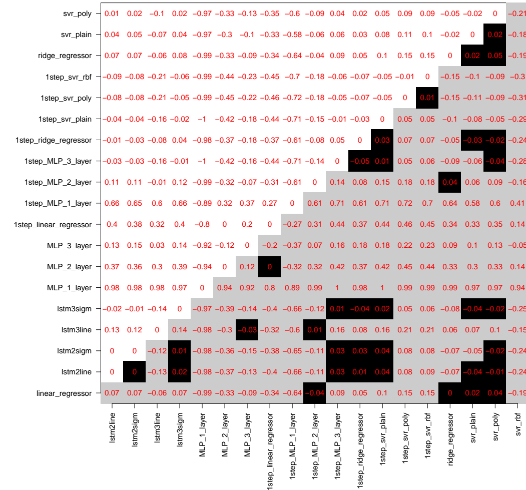
In the Figures 3a, 3b, 3c, 3d, 4a, 4b, 4c, and 4d, the text *1step* denotes a model which takes the total counts of different syscalls and other resource usage as the input and predicts the total energy consumption for the test run. Models that do not include the *1step* prefix predict energy at each time step for the given time series of a given test.

Figure 3a, Figure 3b, Figure 3c, and Figure 3d show the evaluation of pairwise Wilcoxon tests on each group of models as well as the evaluation of Cliff's Delta. The color of a square shows the Wilcoxon result where: *white* means the two models were not tested against one another for their Wilcoxon p-values, *black* (■) means a p-value above or equal to 0.05, and *grey* (■) means a p-value less than 0.05. A p-value less than 0.05 means that two given models are significantly different from one another as shown by the grey squares. Most models were different from each other, as there were few black squares. The number on the square identifies the Cliff's Delta estimate of how different the two models under comparison are. A number close to 0 means that the models are very similar, whereas -1 or 1 mean that the models predict different intervals of joules from one another. Models which predict different intervals from each other can be compared to see if one creates stronger results than the other. We compare the models performance against each other using their total energy prediction accuracy [10].

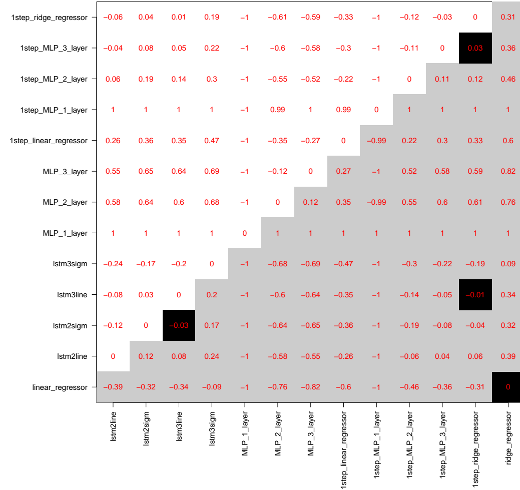
Now that we've demonstrated that the models can predict software energy use and are mostly different from each other we want to evaluate the performance of each model. We show box plots, ordered by the mean total joule prediction error



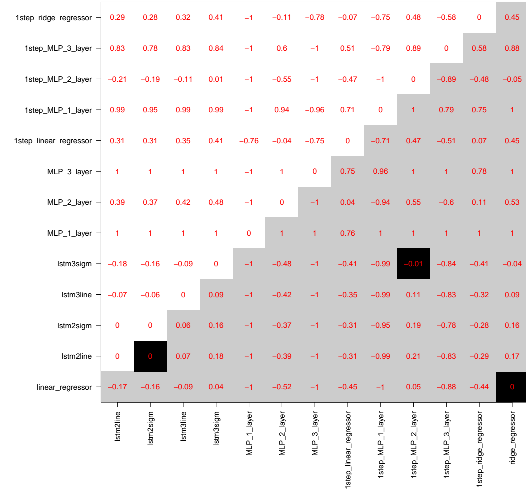
(a) Full feature set



(b) GreenOracle feature set



(c) System call feature set



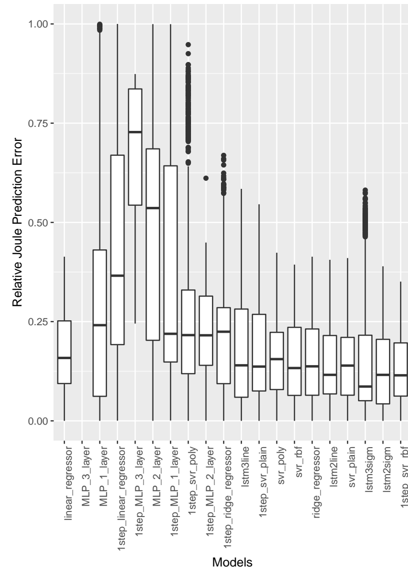
(d) Processor feature set

Fig. 3. Pairwise Wilcoxon and Cliff's Delta of energy models under 4 feature sets. A ▒ denotes a significant difference and ■ denotes no-significant difference using Wilcoxon t-test with Holm correction. Note these are the lower triangle of comparisons.

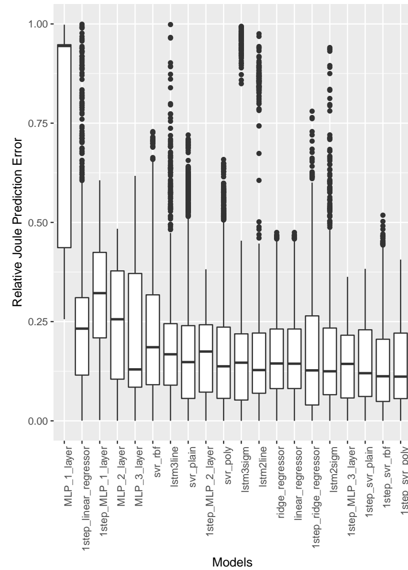
for each model, to illustrate the difference in performance between models. Figure 4a, Figure 4b, Figure 4c, and Figure 4d show box plots of model predictions using different feature sets. To generate the values for each box plot, the model is evaluated on how well it predicts the total energy consumption for each test run. We omit per-step boxplots of timeseries models as the aggregate statistics of per-step relative errors are identical to the cumulative mean average relative error. We show the combined results, in terms of relative error, of each model trained on a given fold predicting the respective holdout set. Note in some of figures that the models boxplots are missing, such as lasso regression or 3 layer MLP, because the models have huge cumulative error and would not fit within the domain of the plot.

VI. Research Questions and Discussion

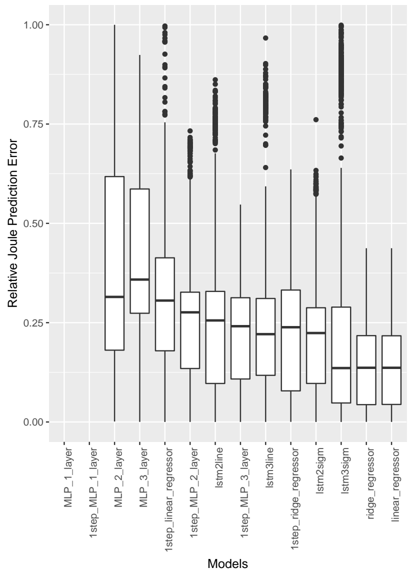
RQ1 *How well do time series models perform when predicting total energy consumption?* Time series models often perform just as well or better than time-summarized/1step models. We evaluated whether or not any of the models perform similarly by checking their relative prediction error of total energy consumption for given software test runs. Our Wilcoxon and Kruskal-Wallis results suggest that each of our models perform differently. We plotted the relative errors of evaluation for each model predicting energy consumption sorted by mean performance in Figure 4a, Figure 4b, Figure 4c, and Figure 4d. These figures show that the time series and *1step* models perform similarly to each other. For example, as shown in



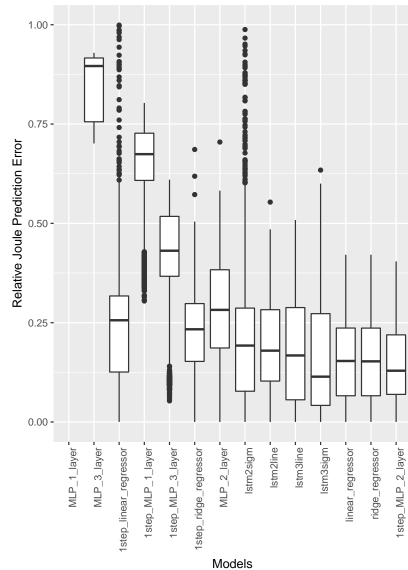
(a) Full feature set evaluated on *1step* and discrete contexts



(b) GreenOracle feature set evaluated on *1step* and discrete contexts



(c) System call feature set evaluated on *1step* and discrete contexts



(d) Processor feature set evaluated on *1step* and discrete contexts

Fig. 4. Model relative prediction error box plots under various feature sets

Figure 4a there is no well defined partition which divides the models into time series or *1step* models. Furthermore, Figure 4b shows that the time series and time-summarized models are competitive with one another. As, the *1step ridge regressor* model can outperform several time series models, but it is outperformed by the time series *lstm2sigm* model. The *1step ridge regressor* model is our baseline for comparison because it represents the state-of-the-art software energy prediction model *GreenOracle* [10]. It is surprising that 4 of the time-summarized models outperform both the *1step ridge regressor* and the *lstm2sigm* models as outlined in Figure 4b. The Wilcoxon rank sum test shows that there is a significant difference ($p < 2.2e - 16$), one shot models have slightly more mean relative error than timeseries models (95% CI $[0.0342, 0.0400]$), but Cliff's delta (0.119) suggests this 3% to 4% difference is negligible. Model performance depends on the feature set but timeseries models do slightly better.

RQ2 *Do time series models with built-in state perform better than time series models without state when predicting energy consumption?* The answer is Yes. Figure 4a and Figure 4b show that the median error of the LSTM models is very low for most of the models. In the Figure 4a, 4 of the top 5 models

are LSTMs. In Figure 4b, with *GreenOracle* feature set, 1 of the top 5 models is an LSTM. The best LSTM-based model in Figure 4b has a median performance of 12% error. Prior work showed that a median performance of roughly 14% was expected for 6 mobile applications [10]. Therefore, we conclude that the LSTM models are competitive with the state-of-the-art. The stateful models are outperformed by *1step* models, or by lasso and ridge regression when considering fewer features since the LSTMs are unable to do their own feature selection. On the full feature set, the Wilcoxon rank sum test shows that there is a significant difference ($p < 2.2e - 16$), stateless models have more mean relative error than stateful LSTM models (95% CI $[0.101, 0.108]$), and a medium effect size (Cliff's delta 0.376).

We also select two random examples to show how our models perform in predicting the actual energy time series. Do they maintain similar shape and scale as the ground truths? Shown in Figure 2d is a stateless time series SVR model predicting the denormalized energy consumption of a given test run. This model's predictions do not fit the observed data well. On the other hand Figure 2c shows a stateful-time series LSTM model predicting the energy consumption of a given test run. The LSTM is capable of achieving a better fit to the shape of the observed data with its denormalized energy predictions. The SVR model does well in the aggregate rather than shape. Whereas, the LSTM model is capable of predicting the fit in some runs, but can miss-predict when the energy consumption of an application is going to change significantly. Figure 2c demonstrates that a developer using the *lstm2sigm* model could predict their energy consumption for the given test run and attribute the energy consumption to a part of that test run.

RQ3 *Energy prediction models can be trained with different feature sets. How does performance change when using them individually versus together?* Models with multiple kinds of features tend to perform better. Two sources of software behaviour are collected from `strace` and process summary information from *procfs* throughout the software tests. Figure 4c shows the performance of models trained only using the collected system call features and Figure 4d shows the performance of models only trained on the *procfs* features. In these figures it is clear that most of the models have a median error near 20%. Whereas, in Figure 4b and Figure 4a most of the models have median errors below 20%. The Wilcoxon rank sum test shows that there is a significant difference ($p < 2.2e - 16$), models trained on smaller feature sets tend to have more mean relative error than the full feature set models (95% CI $[0.00712, 0.0103]$) with negligible effect size (Cliff's delta 0.0289). When we compare *procfs* features to a full feature set the model behave significantly differently (95% CI $[0.0438, 0.0489]$) with nearly a 4.5% difference, although the effect size is negligible (Cliff's Delta 0.131). This suggests there is a difference but across all models it is a 1% difference or less. The full set of features perform better than any subset. CPU/*procfs* features alone are not enough.

RQ4 *Do shallow multi-layer-perceptrons perform similarly to existing stateless models such as those based on linear regression?* In most cases the MLP models performed very poorly on the learning task. Lasso and ridge regression models performed well on the system-call only and process-only feature sets. The Wilcoxon rank sum test shows that there is a significant difference ($p < 2.2e - 16$), MLP models have more mean relative error than linear models (95% CI $[0.377, 0.397]$) with a medium effect size (Cliff's δ 0.387).

A. Threats to Validity

Construct validity is threatened by energy and software behaviour measurements. Internal validity is threatened by time-series alignment, synchronization, and interpolation. External validity is threatened by the small number of applications used to produce the models (5 apps per fold), this is balanced by using multiple versions of each application.

VII. Conclusion

In summary we explored software energy consumption prediction using time series regression models. Our work evaluated and compared several existing state of the art machine learning based energy prediction methods. Deep learning based models based on LSTMs did well but often simpler linear regression models performed just as well. LSTM based models had best median relative error on the full feature set.

Time series based models were accurate per step and cumulatively across the whole test run. We found that machine learning based time series models and *1step* models perform similarly when predicting the total energy consumption of a software test run. We found evidence that stateful time series models like LSTMs who maintained state/memory about the past, predicted energy consumption better than stateless models, like SVR. We also found that shallow MLPs performed poorly in comparison to the lasso and ridge regression models when predicting energy consumption for a given test run.

Thus if a trained model is distributed to developers[1], tools need only to record output *strace* and *procfs* output in order to estimate the energy consumption of their application as it runs without the need for external hardware measurement.

Future work will focus on shape matching time-series, as well as closer correlating source code to energy consumption time-series.

Acknowledgments

We gratefully acknowledge the support of the NSERC discovery grant and we gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

[1]Download our models here https://archive.org/details/deep_green.tar

References

[1] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 22–31.

[2] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 179–198, 2013.

[3] R. W. Ahmad, A. Gani, S. H. A. Hamid, F. Xia, and M. Shiraz, "A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues," *Journal of Network and Computer Applications*, vol. 58, pp. 42–59, 2015.

[4] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 588–598.

[5] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An empirical study of the energy consumption of android applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sept 2014, pp. 121–130.

[6] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Mobile Application Energy Consumption Using Program Analysis," in *ICSE '13*, 2013, pp. 92–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486801

[7] D. Li, A. H. Tran, and W. G. J. Halfond, "Making Web Applications More Energy Efficient for OLED Smartphones," in *ICSE 2014*, Hyderabad, India, June 2014, pp. 527–538. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568321

[8] D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 2013, pp. 78–89.

[9] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?" in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 103–114.

[10] S. A. Chowdhury and A. Hindle, "Greenoracle: Estimating software energy consumption with energy measurement corpora," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901763

[11] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597085

[12] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Optimizing energy consumption of guis in android apps: A multi-objective approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786847

[13] M. Dong and L. Zhong, "Sesame: Self-constructive system energy modeling for battery-powered mobile systems," *arXiv preprint arXiv:1012.2831*, 2010.

[14] ——, "Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems," in *Proceedings of the MobiSys '11*, June 2011, pp. 335–348. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000027

[15] A. C. Rice and S. Hay, "Decomposing power measurements for mobile devices." in *PerCom*, vol. 10, 2010, pp. 70–78.

[16] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 12–21. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597097

[17] N. Vallina-Rodriguez and J. Crowcroft, "Erdos: achieving energy savings in mobile os," in *Proceedings of the sixth international workshop on MobiArch*. ACM, 2011, pp. 37–42.

[18] D. Chu, A. Kansal, J. Liu, and F. Zhao, "Mobile apps: It's time to move up to condos." in *HotOS*, 2011.

[19] S. A. Chowdhury, V. Sapra, and A. Hindle, "Client-side energy efficiency of http/2 for web and mobile app developers," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 529–540.

[20] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia, "The power of system call traces: Predicting the software energy consumption impact of changes," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2014, pp. 219–233.

[21] S. A. Chowdhury, S. Gil, S. Romansky, and A. Hindle, "Greenscaler: Automatically training software energy model with big data," PeerJ Preprints, Tech. Rep., 2016.

[22] *syscalls(2) Linux Programmer's Manual*, 4th ed., January 2015.

[23] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 29–42.

[24] *proc(2) Linux Programmer's Manual*, 4th ed., January 2015.

[25] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," 2014, preprint. [Online]. Available: http://arxiv.org/abs/1412.5567

[26] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *arXiv e-prints*, vol. abs/1512.02595, 2015, preprint. [Online]. Available: http://arxiv.org/abs/1512.02595

[27] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[28] M. White, C. Vendome, M. Linares-Vasquez, and D. Poshyvanyk, "Toward Deep Learning Software Repositories," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 334–345.

[29] V. Raychev, M. Vechev, and E. Yahav, "Code Completion with Statistical Language Models," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI'14. ACM, 2014, pp. 419–428.

[30] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 404–415.

[31] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[32] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[33] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[34] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.