Energy-Efficient Machine Learning on the Edges

Mohit Kumar, Xingzhou Zhang, Liangkai Liu, Yifan Wang, and Weisong Shi
Department of Computer Science, Wayne State University, Detroit, MI 48202, USA

***Abstract*—Machine learning-based software is vital for future Internet of Things (IoT) applications and Connected and Autonomous Vehicles (CAVs) as it provides the core value of these services by leveraging the enormous amount of data collected on the edge. These services utilize various machine learning models which make it computationally intensive on the edges. There has been a lot of work to make the hardware efficient. No matter how efficient is the hardware, an inefficient machine learning model can account for high energy consumption and overheating problem. However, there are very few tools available that can help software developers or researchers to make the machine learning models energy efficient.**

Our main contributions of this paper are two-fold: First, we summarize the state-of-the-art techniques about energy-efficient machine learning on the edges. Second, targeting specific Java programming language, we present an Eclipse plugin named Java Energy Profiler & Optimizer (JEPO) which can help in profiling and optimizing machine learning source code written in Java. JEPO can automatically measure the energy consumption of source code at method granularity. It provides energy efficiency suggestions for data types, operators, control statements, String, exception, objects, and Arrays in Java. JEPO evaluation has shown up to 14.46% improvement in energy consumption when used to optimize the machine learning software WEKA with only 0.48% drop in accuracy.

***Index Terms*—CAVs, IoT, Software, Java, Energy-Efficiency, Eclipse Plugin**

I. Introduction

Technologies like Internet of Things (IoT) and Connected and Autonomous Vehicles (CAVs) have resulted in rapid growth of data that needs to be processed in real-time for building machine learning models. IoT devices will hit a count of 30 billion by 2020 which will produce more than 5 quintillions of data every day [1]. In the future, CAVs will generate 4TB of data just for one hour driving a day [2]. Due to latency, bandwidth, availability and privacy constraints, it is estimated that more than 90% of this data will be processed locally [3], [4]. Edge computing promises handling these constraints by processing data locally [5].

In the scenario of IoT applications, lots of edge devices with unique identifiers are connected to each other for continuously collect, process, and transmit data. Machine learning models have a plethora of applications in smart homes, retail, travel, finance, healthcare, industry, social media, and research. Example of one such application is EdgeBox [6]. EdgeBox leverages edge computing for automatic video analysis to detect safety threaten events in real-time. EdgeBox requires continuous transmission of data to edge node which causes higher energy consumption. In such applications where hardware is always fully utilized, energy efficient software plays a significant role in avoiding the overheating of hardware.

Similarly, various machine learning algorithms are deployed for CAVs scenario for applications like perception, object recognition and decision making [7]. First, data is collected using multiple sensors like GPS/IMU, LiDAR, cameras, radar, and sonar. Then these data will be fed to Deep Neural Networks for object detection, object tracking, path planning, and obstacle avoidance. One challenge is how to process huge amount of data with limited computing resources on the edge. The real-time processing requirements and battery constraints make the energy efficiency more challenging. The computing systems of the CAVs are in badly need of a customized software and hardware for energy efficiency.

The energy efficiency of machine learning algorithms is important as IoT or edge devices have limited computation and energy resources. For the hardware, machine learning accelerators are proposed which can improve performance and energy consumption. Intel Nervana Neural Network Processors and Nvidia DGX-2 are some of the examples of chips released specifically for machine learning. These chips can run machine learning models very efficiently. However, the software is one of the most critical bottlenecks for such chips as it can easily negate their performance and energy efficiency. No matter how efficient the hardware is, if it is not managed properly by the software, it can not help in making IoT or edge devices energy efficient. Therefore, software is as important as hardware to optimize IoT and edge devices in terms of performance and energy. Researches are focused on how to build the energy-efficient system by optimizing the widely-used techniques, from hardware to operating system to algorithms. Software energy consumption can be optimized in several ways like choosing the energy-efficient option in a programming language, using an energy-efficient programming language or choosing an energy-efficient compiling option.

In this work, we concentrate on command-line level methods to optimize software energy efficiency. Today's programming languages provide software developers with several options to perform the same task. Take Java for example, an Array can be copied to other Array either manually or using Java methods. However, not all available options are energy-efficient and the software developers lack the knowledge to choose the best energy-efficient option. We perform various analyses to choose the best option for different components of the Java programming language [8], [9]. These components include data types, operators, control statements, String, exceptions, objects, and Arrays. We also evaluate different Java command-line options in [10]. In addition, we found that there are little software development tools to address the challenges. So in this work, we present our JEPO tool

to help software developers to write energy-efficient machine learning code. This tool is an Eclipse IDE plugin which provides energy-efficient suggestions for Java programming language. It can provide suggestions dynamically while writing code or statically to refactor already written code. To provide suggestions, it analyzes each line of Java file and matches it to the pool of suggestions that we gather from the findings in our earlier work [8], [9]. JEPO can also help the software developers to measure energy consumption at method granularity to determine the energy-hungry Java methods in software. To measure the energy, it injects energy and time measurement code at the start and end of each method in the project. This injected code leverages Intel Running Average Power Limit (RAPL) technology to access machine specific registers (MSR) and measures energy consumption for the package. We perform the initial evaluation of JEPO using an open-source machine learning software Waikato Environment for Knowledge Analysis (WEKA). We are able to achieve up to 14.46% improvement in package energy consumption, up to 14.19% improvement in core energy consumption, and up to 12.93% improvement in execution time. The changes result in only 0.48% drop in accuracy of the classifiers.

The rest of the paper is organized as follows. Section 2 outlines the background work. From Section 3 to Section 6, state-of-the-art techniques of energy-efficient machine learning are discussed, they are hardware & architectures, software & packages, algorithms, and software development. This paper finds that the related work in software development energy-efficiency is not sufficient, so the design and details of JEPO are provided in Section 7. Section 8 describes WEKA and uses it to evaluate JEPO. Section 9 concludes the paper and discusses future work.

II. Background

Pushed by the edge computing techniques and pulled by the AI applications, edge intelligence has been pushed to the horizon. Edge intelligence was defined as the capability to enable edges to execute artificial intelligence algorithms in [11]. The edge device which owns the edge intelligence capability will be able to process image, video, natural language, and time-series data generated by cameras, microphones, and other sensors without uploading data to the cloud and waiting for the response. However, the edge is usually resource-constrained compared to the cloud data center, which is not a good fit for executing DNN represented machine learning algorithms since DNN requires a large energy consumption that comes from the computing and storage requirements. Therefore, the development of edge intelligence calls for the energy-efficiency techniques.

There are two main reasons to achieve the energy-efficiency on the edge intelligence scenarios. Firstly, improving energy-efficiency will reduce the temperature of the edge and therefore solve the cooling problem. In general, the complexity of machine learning algorithms is high and the computing requirement of edge devices is relatively large, which generates a large amount of heat. The excessive heat may affect the normal operation of the edge and cooling the edge will consume more energy. Even worse, in the scenarios like the base station and edge server, it will lead to fire and endanger the safety of residents if the heat is not dissipated in time.

Secondly, achieving energy efficiency will increase the usage time of the battery. Most of the edge devices, such as the mobile phone and electric vehicles, rely on the battery and the length of battery life will greatly affect the user experience. For example, an electric car will drive for 500 kilometers continuously when it is fully charged in general. If energy efficiency is increased by 20%, the continuous driving distance will increase to 600 kilometers. Therefore, the improvement in energy efficiency is an effective way to increase the availability of edges and expand the radius of life. Therefore, improving energy efficiency on the edge is indispensable to realize the edge intelligence.

III. Hardware & Architecture

To meet the growing demand of the pervasive machine learning application in edge, a lot of research works have focused on designing specific accelerator architecture to reduce the process time and improve the energy-efficiency of the machine learning algorithms running on edge. New hardware architecture forces the community to optimize the system software and algorithms to fully exploit hardware performance. So the research work on hardware and architecture can be divided into two categories:

- specific hardware accelerator architecture for machine learning on edge;
- hardware based full-stack optimization method for machine learning on edge.

A. Accelerator Architecture

GPU provides multiple micro computing units to exploit the parallelism of the machine learning algorithms, especially for convolutional neural network (CNN), and it has been widely used to accelerate the machine learning algorithms both in cloud and edge[12]–[14]. NVIDIA has released a series of embedded energy-efficiency GPU products to support the edge scenarios, such as the NVIDIA® DRIVE™ PX2 ($250W$) for connected and autonomous vehicles[15], and NVIDIA® Jetson™ platforms ($5 - 30W$) for robots and drones[16].

DianNao family[17]–[19] is a set of energy-efficient hardware accelerators for machine learning algorithms. DianNao accelerator provided a Neural Functional Unit (NFU) and three SRAM buffers. DianNao can execute neural networks at different scales by splitting a large neural networks into small workloads to reuse the NFU and SRAM buffers. According to the evaluation results, DianNao reduces the total energy by $21.08x$ compared with SIMD processors. ShiDianNao[20] is one of the accelerators for DianNao family designed for visual algorithms in edge. It achieved $60x$ energy efficiency than the previous state-of-the-art AI hardware in 2015, which deploy the AI processor next to the camera sensors, reducing the power consumption of sensor data loading and storing.

Google Tensor Processing Unit (TPU)[21] is the neural networks accelerator in data center. And Edge TPU[22] is the embedded version of TPU for edge computing. The basic novelty of TPU is similar to DianNao, which is splitting neural networks to fit and reuse the basic computing units. TPU takes the traditional systolic matrix multipliers as the basic processing units, systolic execution of matrix multipliers saves energy by reducing reads and writes of the Unified Buffer. And TPU achieves $30x - 80x$ TOPS/Watt compared with CPU and GPU. The systolic array architecture has also been deployed in FPGA to achieve higher energy-efficiency for processing CNN[23].

IBM TrueNorth[24] and Intel Loihi[25] both are neuromorphic processors whose architecture is well suited to many complex neural networks algorithms. The neuromorphic architecture is an efficient and flexible non–von Neumann architecture, which uses silicon technology to implement the programmable neurons and synapses to mimic human brain to execute the computing tasks. The novel architecture only comsumes milliwatts-scale energy when processing neural networks applications in real-time.

B. Hardware Based Full-Stack Optimization

Several studies have focused on algorithm-hardware co-design method to achieve energy-efficiency machine learning on edge. ESE[26] used FPGAs to accelerate the LSTM model on mobile devices, which adopted the load-balance-aware pruning method to ensure high hardware utilization and scheduled the compressed model to multiple PEs for parallelism. The hardware architecture they implemented have achieved $40\times$ and $11.5\times$ higher energy efficiency compared with the CPU and GPU.

EIE[27] is an energy efficient inference engine for compressed deep neural network. It leverages multiple methods to improve energy efficiency for deep learning algorithms, such as saving the model in on-chip SRAM instead of external DRAM, skipping zero activations from ReLU, exploiting sparsity and sharing weights. According to the benchmarking results, EIE have achieved $24,000\times$, $3,400\times$ and $19\times$ more energy efficient than a CPU, GPU and DaDianNao respectively.

ADMM-NN[28] is an algorithm-hardware co-design framework of deep nerual networks using Alternating Direction Method of Multipliers (ADMM). It proposed a joint framework of weight pruning and quantization using ADMM. The regularization target dynamically updated in each ADMM iteration resulted in higher performance in model compression, and the the computation reduction leaded the energy efficiency improvement. And many research works start to use similar method to deploy deep learning algorithms on edge to achieve energy efficiency[29]–[31].

IV. Software & Packages

Many projects focus on developing and optimizing software platforms and machine learning packages to meet the low-power requirement of the edge.

A. Software platform

To run the intelligence applications on the edge with the limited energy resources, many lightweight operating systems and energy-efficient oriented computing frameworks are designed.

TinyOS[32] is an embedded, component-based operating system for low-power edge devices. The application-specific nature of TinyOS ensures that no unnecessary functions consume energy. What' more, three aspects are used to decrease the power consumption: application-transparent CPU power management, power management interfaces, and efficiency gains arising from hardware/software transparency. The experimental result based on an AI application, object tracking, shows that TinyOS reduces energy consumption by 30% with minimal degradation of tracking accuracy.

Phi-Stack[33] is a co-designed hardware-software stack to natively support web and intelligence applications with minimized cost, footprint, and energy consumption. Phi-OS and Phi-DK are designed to meet the low-power requirement of the smart IoT devices. Each command of Phi-Stack takes no more than 100 ms latency and consumes no more than 200mW of power when using the intelligence application (such as turning on the light by voice) as the benchmark.

OpenEI [34] was proposed in 2019 as a lightweight software platform to equip the edge with intelligent processing capability. OpenEI is designed to reduce the latency, energy, and memory footprint while guaranteeing the latency. To decrease the energy consumption, OpenEI leverages a lightweight package manager to run the AI algorithms on the edge. Besides, a model selector is used to pick up the best marching hardware and software combination to save energy.

For the connected and autonomous vehicles, OpenVDAP [4] was proposed as an open vehicular data analysis platform. EdgeOS_v is an operating system inside OpenVDAP, which is designed to provide a security running environment and energy-efficient and latency-aware resource management for upper applications. Besides, E2M [35] is an energy efficient middleware for autonomous mobile robots which tackles the inefficiencies during the sensor data accessing, machine learning model execution, and the management of multipurpose applications to save energy of the computing system.

B. Machine learning packages

To execute AI algorithms on the edge efficiently, several deep learning packages, are specifically designed, such as Caffe2, MXNet, and PyTorch. Compared with cloud versions, these packages require significantly fewer resources to achieve a lower power consumption, lower memory footprint while behaving almost the same in terms of inference.

MXNet [36] is a flexible and efficient library for deep learning to support CNN and long short-term memory networks (LSTM). Caffe2 [37] is a lightweight, modular, and scalable deep learning framework, which is built based on the Caffe. PyTorch [38] is a python package that provides tensor computation with strong GPU acceleration and deep neural networks built on a tape-based auto-grad system.

In addition to the above packages which handle the training and inference tasks, many packages are designed to support the inference only, such as TensorFlow Lite, CoreML, QNNPACK, and Paddle Lite. TensorFlow Lite [39] is TensorFlow's lightweight solution which is designed for mobile and edge devices by optimized and quantized kernels to reduce the latency and energy consumption. Apple published CoreML [40], a deep learning package optimized for on-device performance to minimizes memory footprint and power consumption. Facebook developed QNNPACK (Quantized Neural Networks PACKage) [41], which is a mobile-optimized library and provides an implementation of common neural network operators on quantized 8-bit tensors to achieve low energy and high-performance. Paddle Lite [42] is designed to make it easy to perform inference on edge and IoT devices and it is compatible with PaddlePaddle and other pre-trained models. The execution module and analysis module are decoupled to guarantee the lightweight and low power consumption.

Zhang *et al.* made a comprehensive performance comparison of these packages on the edges and evaluated the energy consumption[43]. They found that no packages could achieve the best performance in all dimensions, which indicated that there was a large space to improve the performance of the packages on the edge.

V. Algorithms

In order to improve the energy-efficiency of the machine learning models, various efforts have been made in the co-design of machine learning algorithms and hardware. Generally, the co-design approaches can be divided into two categories [44]:

- reduce the computational requirements;
- reduce the accuracy of operations and operands.

A. Lightweight machine learning models

To achieve energy efficiency, reducing the computational requirements of machine learning algorithms is a critical approach.

In 2015, Han *et al.* [45] proposed that pruning redundant connections and retraining the deep learning models to fine tune the weights is an effective way to reduce the computing complexity. It helped in improving the energy efficiency of neural networks. After their projects, research on model compression which includes pruning and quantification became popular. Subsequently, Iandola *et al.* [46] developed Squeezenet, a small CNN architecture. It achieves AlexNet-level [47] accuracy with 50 times fewer parameters on ImageNet.

Google Inc. [48] presented MobileNets, the efficient CNN for mobile vision applications, in 2017. It not only focuses on optimizing latency but also builds small networks to reduce the memory footprint and energy consumption. ESPNetv2 [49] was designed in 2019 as a light-weight and power efficient neural network, which used group point-wise and depth-wise dilated separable convolutions to learn representations with fewer FLOPs and parameters. Experimental results based on NVIDIA GTX 1080 Ti and NVIDIA Jetson TX2 show that ESPNetv2 is much more power efficient than MobileNet. Eyeriss [50] is a project which seeks for the hardware and software co-design for energy efficient execution of deep learning algorithms. Based on the profiling results that data movement consumes the most energy, a architecture which exploits data re-usability is proposed and evaluated on a real test chip.

In 2018, Bonsai [51] refers to a tree-based algorithm used for efficient prediction on IoT devices. More specifically, it is designed not for the deep learning models, but for other machine learning tasks such as regression, ranking, and multi-class classification, etc.

B. Reduce Accuracy

In addition to achieve energy efficiency by lightweight models, there are also a lot of works on improving energy-efficiency during runtime as well as designing energy-efficient DNN models by reducing the accuracy of operations and operands. Reducing accuracy is usually achieved by reducing the number of bits/levels to represent the data, which decreases the computation requirements as well as the storage cost [44].

Quantization is proposed to reduce accuracy by mapping the data to quantization levels. There are three types of quantization: uniform quantization, log quantization, or learned quantization. The difference between them is that whether the distribution of distances between each quantization level is uniform, log-based or learned-based.

For uniform quantization, the key is to convert values and operations from floating points to fixed points. Dynamic fixed point is one method of conversion which is based on the limited range of weights and activation [52], [53]. Through fixed dynamic fixed point, the bitwidth can be decreased from 32 bit to 8 bit for weights and 10 bits for activation. Since the memory access and data movement make up a large portion of the energy consumption, the model becomes more energy efficient. Some other works explore using even lower bits for the value weights and activation while maintaining accuracy loss less than 1 percent [54], [55]. Besides, BinaryConnect [56] introduces to use a single bit to represent weight values but is has an accuracy loss of 19 percent. In order to decrease this accuracy loss, lots of works like QNNs [57] and HWGQ-Net [58] focused on slightly increasing the bit size of the activation.

For nonuniform quantization, the distribution of weights and activation are based on either log-based or learned-based. Compared with uniform quantization, log-based quantization achieves much less accuracy loss [59]. In addition, the log 2 based quantization makes the multiplication operation easier. Learned-based is proposed to support weight sharing in a filter or layer. One example of weight sharing is deep compression [60], in which the number of weights per layer is reduced to 256 for convolutional layer and 16 for fully connected layer. Compared with other quantization approaches, the learned-based approach does not reduce the accuracy and only reduces the storage requirements.

VI. Software Development

Software energy measurement challenges like lack of energy measurement tools, lack of instrumentation to estimate energy consumption for various OS and processors are presented in [61]. SEEDS and Chameleon frameworks for automating code-level changes and optimizing Java applications were introduced in [62] and [63]. These frameworks can select the most efficient collection for improving the energy efficiency of an application. SEEDS resulted in 17% energy consumption improvement. Software change-impact analysis tool, GreenAdvisor, help software developers to estimate the change in energy profiles due to change in an application system calls [64]. Eco, a programming model, is introduced in [65] to provide support for energy-aware applications. A similar tool, EnSights, provides energy change information by analyzing the change in the structure of a code [66]. It can estimate the change in energy consumption with F-scores of up to 86%. jStanley - an Eclipse plugin - provides a suggestion for energy consumption usage of collections in Java [67]. The software developers can use the suggestion to replace a collection with a better collection. It shows to improve energy consumption between 2% and 17%. In this chapter, we present an Eclipse plugin which gives suggestions about data types, operators, control statements, String, exceptions, objects, and Arrays. It can also help software developers in measuring the energy consumption of software applications at method granularity.

VII. JEPO

Machine learning models can be made energy-efficient either by implementing a new energy-efficient model from scratch or by refactoring already implemented models. However, software developers don't have the tools which can help them to write energy-efficient code or to refactor already existing code. Therefore, in this section, we present JEPO which can help both in the implementation of new energy-efficient models and refactoring already existing models.

JEPO[1] is an Eclipse plugin developed to provide suggestions for software developers to write energy-efficient machine learning code in real-time or to use suggestions to refactor already written code. It is a mix of a static and dynamic tool as it can not only give real time suggestions while writing the code but also can be used to get suggestions for already written code. The suggestions are a result of our earlier work in which we analyze various components of Java programming language [8], [9]. These suggestions are hardcoded in the tool and displayed whenever the tool detect specific Java components like data types, operators, control statements, String, exceptions, objects, and Arrays.

JEPO analyzes each line of the code and checks for a specific pattern of code to generate various suggestions. These patterns relate to various components of Java programming language and are shown with suggestions in Table I. For primitive data types - `byte`, `short`, `int`, `long`, `float`, `double` and `char` - `int` primitive data type is recommended for the best energy efficiency. Decimal numbers when typed as scientific notation consumes lesser energy. Wrapper classes are object representation of primitive data types. Integer wrapper class object consumes lesser energy than any other wrapper class. Static keyword result in up to 17,700% increase in energy consumption of variables. Modulus is the most energy-expensive arithmetic operator. Ternary operator consumes higher energy than if-then-else option. Putting the most common cases in a short-circuit operator helps in saving energy. For string concatenation, `StringBuilder append` is the best way to concatenate string. String comparison method `compareTo` results in higher energy consumption than `equals` method. `System.arraycopy()` is the best way to copy array. Array column traversal is energy expensive than row traversal. All these suggestions work better when the above java components are used repeatedly in a program. More general suggestions can be found in our earlier work [8], [9].

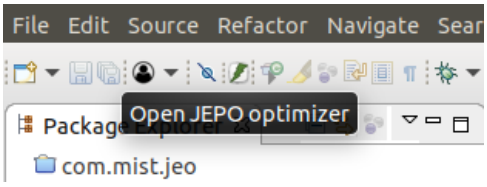

Fig. 1. JEPO toolbar button

JEPO can also help software developers to determine the energy-hungry method in a Java project in Eclipse. This is achieved by injecting code in bytecode to read the machine specific registers (MSR) at the start and end of each method in the Java project using Javassist library [68]. We first search for all classes that have main method in the project. If there is only one main class, then we choose it as our main class. If there is more than one, then we take user input to determine the correct main class for the project. After we finalize the main class, we create a new Java file named JEPOInsert in com.mist.jepo package. The purpose of this class is to inject the energy measurement code for each method in the project and then run the earlier selected main class. The injected code measure and store the MSR, as well as the execution start or stop time, whenever a method is executed. When the execution end, the energy consumption and execution time for all the executed methods are stored in a result.txt file in Java project directory and also shown in JEPO view. If one method is executed more than once, then the measurements are stored for each execution.

JEPO include one toolbar button and two pop-up menu buttons. The toolbar button (shown in Fig.1) opens JEPO view if it's not already open and then shows the suggestions for the already open Java file. If the Java file is not already open then JEPO will show an empty view. The toolbar button view provides dynamic suggestion as shown in Fig. 2. The pop-up menu button can be accessed by doing right-click on any Java project. The pop-up menu will then show a button named JEPO with two sub-menu button - JEPO profiler

[1] https://github.com/mohitkumar14/JEPO

TABLE I
JAVA COMPONENTS & SUGGESTIONS

Java Components	Suggestions
Primitive data types	int is the most energy-efficient primitive data type. Replace if possible.
Scientific notation	Scientific notation results in lower energy consumption of decimal numbers.
Wrapper classes	Integer Wrapper class object is the most energy-efficient. Replace if possible.
Static keyword	static keyword consumes up to 17,700% more energy. Avoid if possible.
Arithmetic operators	Modulus arithmetic operator consumes up to 1,620% more energy than other arithmetic operators.
Ternary operator	Ternary operator consumes up to 37% more energy than if-then-else statement.
Short circuit operator	Put most common case first for lower energy consumption.
String concatenation operator	StringBuilder append method consumes much lower energy than String concatenation operator.
String comparison	String compareTo method consumes up to 33% more energy than the String equals method.
Arrays copy	System.arraycopy() is the most energy-efficient way to copy Arrays.
Array traversal	Two-dimensional Array column traversal result in up to 793% more energy.

TABLE II
WEKA CLASSIFIERS METRICS CALCULATED USING ECLIPSE METRICS PLUG-IN [69] AND CLASS DEPENDENCY ANALYZER (CDA) [70]

Classifiers	Dependencies	Attributes	Methods	Packages	LOC
J48	684	3263	7746	41	101172
Random Tree	668	3235	7611	41	99938
Random Forest	673	3270	7736	42	101812
REP Tree	668	3235	7619	41	100074
Naive Bayes	668	3229	7582	40	99221
Logistic	666	3216	7553	40	98812
SMO	677	3305	7796	43	102250
SGD	669	3222	7585	40	99304
KStar	671	3282	7576	41	99421
IBk	671	3268	7703	41	100339

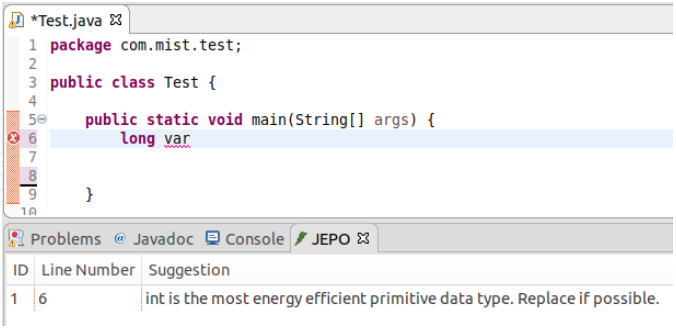

Fig. 2. JEPO dynamic suggestion

and JEPO optimizer (Fig. 3). The JEPO profiler creates the JEPOInsert.java file to measure the energy consumption at method granularity. It shows the energy consumption for each method executed while running a Java project in JEPO view as shown in Fig. 4. The first column shows the method name with package and class name, the second column shows the execution time, and the third column shows the energy consumed. The JEPO optimizer provides suggestions for all the classes in a Java project. The JEPO optimizer view is shown in Fig. 5. The first column shows the class name with package, the second column shows the line number of the class with possible suggestions, and the third column shows the suggestions.

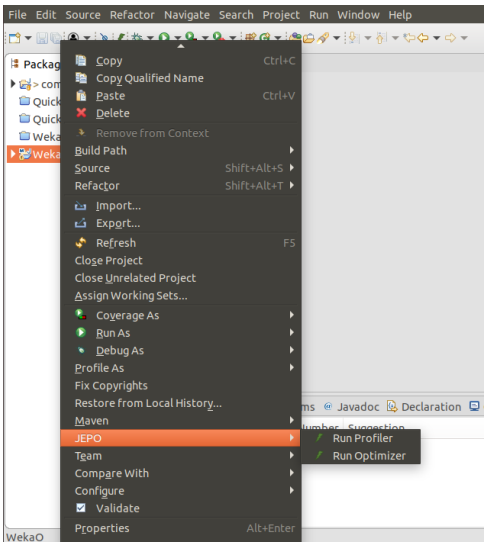

Fig. 3. JEPO pop-up menu buttons

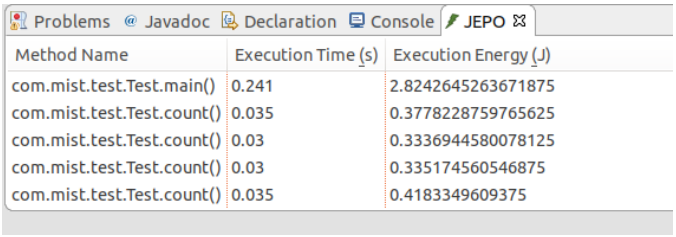

Fig. 4. JEPO profiler view

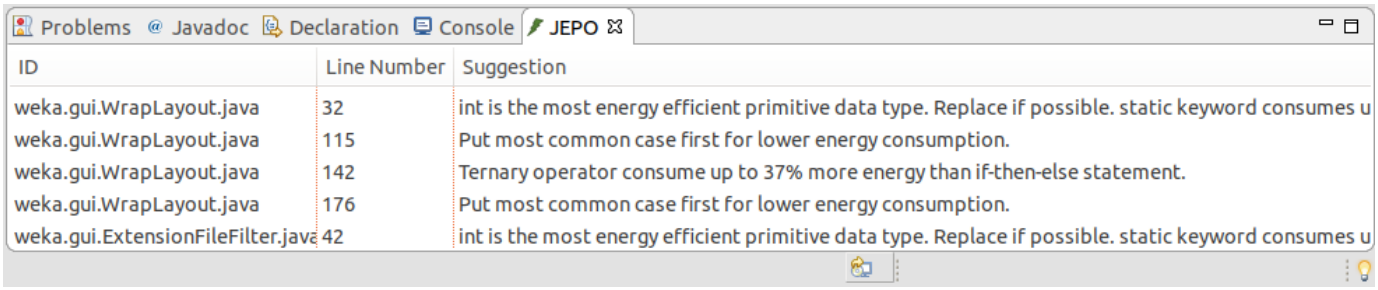

Fig. 5. JEPO optimizer view

TABLE III
MOA AIRLINES DATA

Attributes	Type
Airline	Nominal
Flight	Numeric
Airport From	Nominal
Airport To	Nominal
Day Of Week	Nominal
Time	Numeric
Length	Numeric
Delay	Binary

VIII. Validation

For evaluating JEPO, we leverage WEKA an open-source machine-learning software. We first make changes to WEKA as per JEPO suggestions and then evaluated the different classifiers on a laptop with Ubuntu 16.04.4 LTS, Intel(R) Core(TM) i5-3317U v5, 4 GB of RAM, and JDK version 1.8.0_151.

WEKA software has 3373 classes and different classifiers specifications are shown in Table II. Dependencies, attributes, methods, packages, and line of code (LOC) have almost the same count for all classifiers. J48 implements a modified version of C4.5 which uses decision tree for classification. For building trees, RandomTree takes into account a given number of random features at each node without performing any pruning. RandomForest uses bagging on ensemble of random trees. REPTree uses information gain and variance reduction for constructing decision or regression tree. For pruning, reduced-error prunning method is used. Naive Bayes is a probabilistic classifier which is based on Bayes theorem. Logistic builds a multinomial logistic regression that uses a ridge estimator to guard against overfitting by penalizing large coefficients based on [71]. SMO uses polynomial or Gaussian kernels to implement the sequential minimal optimization algorithm for training a support vector qualifier [72], [73]. SGD is a stochastic gradient descent learning model with various loss functions. KStar and IBK are lazy classifiers that work only during the classification time. KStar implements a nearest-neighbor classifier with generalized distance function based on transformations whereas IBk implements a k-nearest-neighbour classifier.

The data used for classification is Massive Online Analysis (MOA) data [74], which is used to predict whether a flight will be delayed or not. The data has 8 attributes and 539,383 instances. We reduce the number of instances to 10,000 due to limited heap memory. The attributes are shown in Table III. The attributes refer to the airline name, flight name, airport from where the flight departs, airport to which flight arrive, day of the week, time of the flight, distance of the flight and whether the flight gets delayed or not. There are 4 nominal, 3 numeric and one binary attribute. For airline and airports nominal values, the distinct values are 18 and 293, respectively.

Next, we make changes to the dependent classes as per JEPO suggestions and evaluated various classifiers using stratified 10-fold cross-validation. We first run each classifier 10 times to measure Package energy, CPU energy, and execution time using `perf` Linux tool. After that, we detect outliers using Tukey's method [75] from each metric, replace the outliers measurements with new measurements and again check for outliers. We repeat this process until no outlier is left. When no outlier is left, we calculated the mean of values. The final values are shown in Table IV. As expected, the changes made are almost same due to the same number of dependencies. However, other metrics do not agree with the number of changes. For Package energy consumption, CPU energy consumption and execution time, Random Forest shows the highest improvement of 14.46%, 14.19%, and 12.93% respectively. Random Tree shows the most amount of accuracy drop of 0.48% which is very low and acceptable. We have to calculate accuracy drop as there was precision loss when we changed `double` to `float` or `long` to `int`.

These results show an increase in metrics improvement when we increase the number of instances of MOA data to 20,000. For autonomous vehicles, data centers, and supercomputers, where huge amount of data is analyzed in short time, JEPO can help to significantly reduce the energy consumption of software.

IX. Conclusion

IoT and CAVs services run various machine learning models that can have a huge impact on their performance and energy. Machine learning software optimization can help in such scenarios to improve performance and reduce energy consumption. Software energy-efficiency research has been around for some years. Researchers have performed analysis on various languages. However, there is a lack of tool

TABLE IV
WEKA EVALUATION

Classifiers	Changes	Package Improvement (%)	CPU Improvement (%)	Execution Time Improvement (%)	Accuracy Drop (%)
J48	877	4.44	4.68	3.96	0.00
Random Tree	709	0.02	0.01	0.01	0.48
Random Forest	719	14.46	14.19	12.93	0.00
REP Tree	723	3.70	3.49	2.01	0.00
Naive Bayes	711	3.58	3.82	0.00	0.00
Logistic	711	0.10	0.10	0.00	0.00
SMO	713	0.05	0.08	0.04	0.17
SGD	713	7.48	5.76	5.56	0.05
KStar	711	6.82	5.31	0.00	0.00
IBk	711	5.50	5.34	6.01	0.00

which can disseminate these software energy savings findings to software developers. Therefore, in this work, we present an Eclipse plugin JEPO to help software developers write energy-efficient code, dynamically and statically. JEPO can also provide measurements for energy consumption at method granularity. Using JEPO we were able to achieve up to 14.46% improvement in package energy consumption, up to 14.19% improvement in CPU energy consumption, up to 12.93% in execution time and with only 0.48% drop in accuracy. In the future, we hope to improve JEPO by including more suggestions for software developers.

ACKNOWLEDGMENT

This work is supported in part by National Science Foundation (NSF) grant CNS-1561216.

REFERENCES

[1] T. Stack, "Internet of Things (IoT) data continues to explode exponentially. Who is using that data and how?" *URl: https://blogs. cisco. com*, 2018.

[2] P. Nelson, "Just one autonomous car will use 4,000 GB of data/day," *Network World. Available online: www. networkworld. com/article/3147892/internet/one-autonomous-car-will-use-4000gb-of-dataday. html (accessed on 24 December 2018)*, 2016.

[3] R. Kelly, "Internet of things data to top 1.6 zettabytes by 2022," *Campus Technology*, vol. 9, pp. 1536–1233, 2016.

[4] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong, "OpenVDAP: An open vehicular data analytics platform for CAVs," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1310–1320.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] B. Luo, S. Tan, Z. Yu, and W. Shi, "EdgeBox: Live edge video analytics for near real-time event detection," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 347–348.

[7] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating autonomous vehicle systems," *Synthesis Lectures on Computer Science*, vol. 6, no. 1, pp. i–186, 2017.

[8] M. Kumar, Y. Li, and W. Shi, "Energy consumption in java: An early experience," in *Green and Sustainable Computing Conference (IGSC), 2017 Eighth International*. IEEE, 2017, pp. 1–8.

[9] M. Kumar, *Energy Efficiency of Java Programming Language*. Wayne State University, 2018.

[10] M. Kumar and W. Shi, "Energy consumption analysis of java command-line options," in *Green and Sustainable Computing Conference (IGSC), 2019 Tenth International*. IEEE, 2019.

[11] X. Zhang, Y. Wang, S. Lu, L. Liu, L. Xu, and W. Shi, "OpenEI: An open framework for edge intelligence," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1840–1851.

[12] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with COTS HPC systems," in *International conference on machine learning*, 2013, pp. 1337–1345.

[13] S. Lu, Y. Yao, and W. Shi, "Collaborative learning on the edges: A case study on connected vehicles," in *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[14] Y. Wang, L. Liu, X. Zhang, and W. Shi, "HydraOne: An indoor experimental research and education platform for CAVs," in *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[15] NVIDIA Corporation. (2019) NVIDIA DRIVE PX2: Scalable AI platform for Autonomous Driving. [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-platform

[16] ——. (2019) NVIDIA Jetson Platform. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[17] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2014, pp. 269–284.

[18] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 393–405.

[19] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "DianNao family: energy-efficient hardware accelerators for machine learning," *Communications of the ACM*, pp. 105–112, 2016.

[20] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 92–104.

[21] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. Piscataway, NJ: IEEE, 2017, pp. 1–12.

[22] Google LLC. (2019) Google Edge TPU. [Online]. Available: https://cloud.google.com/edge-tpu/

[23] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 29.

[24] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[25] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[26] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[27] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 243–254.

[28] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 925–938.

[29] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] H. Li, N. Liu, X. Ma, S. Lin, S. Ye, T. Zhang, X. Lin, W. Xu, and Y. Wang, "ADMM-based weight pruning for real-time deep learning acceleration on mobile devices," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. ACM, 2019, pp. 501–506.

[31] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang *et al.*, "Progressive DNN compression: A key to achieve ultra-high weight pruning and quantization rates using admm," *arXiv preprint arXiv:1903.09769*, 2019.

[32] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *et al.*, "TinyOS: An operating system for sensor networks," in *Ambient intelligence*. Springer, 2005, pp. 115–148.

[33] Z. Xu, X. Peng, L. Zhang, D. Li, and N. Sun, "The ϕ-stack for smart web of things," in *Proceedings of the Workshop on Smart Internet of Things*. ACM, 2017, p. 10.

[34] X. Zhang, Y. Wang, S. Lu, L. Liu, L. xu, and W. Shi, "OpenEI: An open framework for edge intelligence," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019, pp. 1840–1851.

[35] L. Liu, J. Chen, M. Brocanelli, and W. Shi, "E2M: an energy-efficient middleware for computer vision applications on autonomous mobile robots," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 59–73.

[36] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *LearningSys at NIPS*. ACM, 2015.

[37] (2018) Caffe2: A new lightweight, modular, and scalable deep learning framework. https://caffe2.ai/. [Online]. Available: https://caffe2.ai/

[38] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS workshop*, no. EPFL-CONF-192376, 2011.

[39] (2018) Introduction to TensorFlow Lite. https://www.tensorflow.org/mobile/tflite/. [Online]. Available: https://www.tensorflow.org/mobile/tflite/

[40] (2019) Core ML: Integrate machine learning models into your app. https://developer.apple.com/documentation/coreml. [Online]. Available: https://developer.apple.com/documentation/coreml

[41] D. Marat, W. Yiming, and L. Hao. (2018) QNNPACK: Open source library for optimized mobile deep learning. [Online]. Available: https://code.fb.com/ml-applications/qnnpack/

[42] (2019) Paddle Lite: Multi-platform high performance deep learning inference engine. [Online]. Available: https://paddlepaddle.github.io/Paddle-Lite/

[43] X. Zhang, Y. Wang, and W. Shi, "pCAMP: Performance Comparison of Machine Learning Packages on the Edges," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/zhang

[44] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[45] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[46] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[48] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[49] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9190–9200.

[50] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 367–379.

[51] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2KB RAM for the internet of things," in *International Conference on Machine Learning*, 2017, pp. 1935–1944.

[52] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of convolutional neural networks onto FPGA," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–8.

[53] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[54] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[55] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale convnets," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*. IEEE, 2016, pp. 1–2.

[56] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[57] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[58] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5918–5926.

[59] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "LogNet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5900–5904.

[60] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[61] A. Hindle, "Green software engineering: the curse of methodology," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 5. IEEE, 2016, pp. 46–55.

[62] I. Manotas, L. Pollock, and J. Clause, "SEEDS: a software engineer's energy-optimization decision support framework," in *Proc. 36th Int. Conf. on Software Engineering*. ACM, 2014, pp. 503–514.

[63] O. Shacham, M. Vechev, and E. Yahav, "Chameleon: adaptive selection of collections," in *ACM Sigplan Notices*, vol. 44, no. 6. ACM, 2009, pp. 408–418.

[64] K. Aggarwal, A. Hindle, and E. Stroulia, "Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 311–320.

[65] H. S. Zhu, C. Lin, and Y. D. Liu, "A programming model for sustainable software," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1. IEEE, 2015, pp. 767–777.

[66] H. M. Alvi, H. Sahar, A. A. Bangash, and M. O. Beg, "Ensights: A tool for energy aware software development," in *Emerging Technologies (ICET), 2017 13th International Conference on*. IEEE, 2017, pp. 1–6.

[67] R. Pereira, P. Simão, J. Cunha, and J. Saraiva, "jstanley: placing a green thumb on java collections," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 856–859.

[68] S. Chiba, "Javassist—a reflection-based programming wizard for Java," in *Proceedings of OOPSLA'98 Workshop on Reflective Programming in C++ and Java*, vol. 174, 1998, p. 21.
[69] (2019) Eclipse metrics plug-in 1.3.6. Accessed: 2019-17-09. [Online]. Available: http://metrics.sourceforge.net/
[70] (2019) Class dependency analyzer. Accessed: 2019-17-09. [Online]. Available: http://www.dependency-analyzer.org/
[71] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 41, no. 1, pp. 191–201, 1992.
[72] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.
[73] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to platt's SMO algorithm for SVM classifier design," *Neural computation*, vol. 13, no. 3, pp. 637–649, 2001.
[74] (2019) MOA. Accessed: 2019-17-09. [Online]. Available: https://sourceforge.net/projects/moa-datastream/files/Datasets/Classification/
[75] J. W. Tukey, *Exploratory data analysis*. Reading, Mass., 1977, vol. 2.