Knowledge-Based Systems 270 (2023) 110558

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

# Detecting feature influences to quality attributes in large and partially measured spaces using smart sampling and dynamic learning

Daniel-Jesus Munoz *, Mónica Pinto, Lidia Fuentes

*ITIS Software, CAOSD Group, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Andalucía Tech, Málaga, 29071, Spain*

ARTICLE INFO

*Article history:*
Received 9 May 2022
Received in revised form 3 March 2023
Accepted 9 April 2023
Available online 14 April 2023

Dataset link: https://doi.org/10.5281/zenodo.6251045, https://hadas.caosd.lcc.uma.es/savrus.php

*Keywords:*
Configurable system
Sampling
Learning
Quality attributes
Influence
Interacting features

ABSTRACT

Emergent application domains (e.g., Edge Computing/Cloud/B5G systems) are complex to be built manually. They are characterised by high variability and are modelled by large *Variability Models* (VMs), leading to large configuration spaces. Due to the high number of variants present in such systems, it is challenging to find the best-ranked product regarding particular *Quality Attributes* (QAs) in a short time. Moreover, measuring QAs sometimes is not trivial, requiring a lot of time and resources, as is the case of the energy footprint of software systems — the focus of this paper. Hence, we need a mechanism to analyse how features and their interactions influence energy footprint, but without measuring all configurations. While practical, sampling and predictive techniques base their accuracy on uniform spaces or some initial domain knowledge, which are not always possible to achieve. Indeed, analysing the energy footprint of products in large configuration spaces raises specific requirements that we explore in this work. This paper presents **SAVRUS** (**S**mart **A**nalyser of **V**ariability **R**equirements in **U**nknown **S**paces), an approach for sampling and dynamic statistical learning without relying on initial domain knowledge of large and partially QA-measured spaces. SAVRUS reports the degree to which features and pairwise interactions influence a particular QA, like energy efficiency. We validate and evaluate SAVRUS with a selection of likewise systems, which define large searching spaces containing scattered measurements.



## 1. Introduction

A feature in variability modelling is any relevant characteristic or functionality of a system or a software that runs in it.[1] In practice, features can be treated as Boolean or numerical variables [1]. *Variability Models* (VMs) describe running systems (i.e., valid configurations) by specifying incremental relationships among features and their compatibility constraints [2]. Developers and domain experts are often interested in knowing the ranking or distribution of these system configurations regarding a certain non-functional *Quality Attribute* [3] (QA) (e.g., performance and energy consumption). This is especially important in emergent application domains, such as Industry 4.0, Internet of Things (IoT), Edge Computing and B5G systems, imposing specific challenges to represent variability and quality of configurations [4] properly. One of the recurrent QAs addressed by these domains is sustainability or, more concretely, the energy footprint of software systems [5].

The **first challenge** is that these domains bring about highly configurable systems, with *thousands of options and complex constraints, leading to large configuration spaces*. Such systems are composed of hundreds of heterogeneous devices (e.g., IoT devices, cloud servers) with many operating systems and communication options (e.g., WiFi, 5G), among other features. This variability is represented using large VMs.

The **second challenge** is that for large spaces (e.g., BusyBox has $2 * 10^{213}$ [6]) and some QAs (e.g., energy consumption), *it is impossible to build and measure every configuration manually*. These configurations contain hundreds of features representing software distributed in heterogeneous devices. Often, it is not possible to use the same measuring tool (e.g., specific for devices or operating systems), nor these tools or devices are always locally available [7]. In addition, energy estimation requires particular expertise and a lot of time and resources, which is a complex task. Another issue is modelling or representing QAs like energy footprint in VMs. Attributed VMs enrich features with attributes that assess their influence in a particular QA [8]; for example, a *cost* attribute can indicate the monetary cost of incorporating a specific device into a configuration. However, is it

* Corresponding author.
*E-mail addresses:* danimg@lcc.uma.es (D.-J. Munoz), pinto@lcc.uma.es (M. Pinto), lff@lcc.uma.es (L. Fuentes).

[1] This notion of feature in variability modelling differs from the one in machine learning, defined there as an individual measurable property or characteristic of a phenomenon. Notice that this last definition matches the notion of non-functional quality attribute in variability modelling.

https://doi.org/10.1016/j.knosys.2023.110558

realistic to specify that a *WiFi* feature always consumes 15 Watts using a single-valued attribute? We cannot use this approach to represent QAs like energy consumption, because the energy consumed changes dynamically based on other features. For the WiFi feature, the energy consumption depends mainly on the amount of data transferred and the WiFi version. Then, it is wrong to calculate the energy footprint of a system configuration by simply adding the consumption of isolated features [9], and instead, energy values have to be assigned to complete configurations [10]. Also, attributed VM techniques are typically unscalable when applied to the configurations space as the number of configurations is exponentially more extensive than the number of features. For instance, imagine an attributed VM composed of 1000 features and one attribute. The size of its attributed feature space is a maximum of a thousand; on the contrary, the size of its configurations space can reach a million. While attributed VMs is a linear technique, complex QAs need exponential techniques [11], as measuring or predicting up to a thousand instances are minor issues compared to measuring or predicting a million instances.

However, what is true is that certain features influence a particular QA more than others. Moreover, one feature can affect another feature's impact on the quality value of a particular configuration — *interacting features*. There are many examples of interacting features in IoT/Edge/Cloud systems: microprocessor and tasks parameters, operating system and virtualisation techniques, among others. The effect is that any change in a concrete configuration could concur in a cascade of positive and negative influences, which are then reflected in the QA values of the configurations where any of those features are present. Features can interact individually and in higher order in the form of sets (e.g., selecting features A and B degrades the behaviour of C). However, detecting higher-order interactions is a polynomial growth problem — e.g., quadratically for pairs, cubically for triples. Hence, detecting them in large VMs is impossible in a reasonable time [12].

While it is not affordable to assign immutable accurate values to each feature, an alternative is identifying the main culprits affecting a QA value (i.e., particular features and interactions among features or set of features) based on incomplete configuration measurements. The **third challenge is then** *to develop an approach to identify and analyse meaningful interactions when many configuration measurements are unavailable and they cannot be requested on the fly*. Current approaches deal with QAs mostly making use of sampling [11] with machine learning [13], basically as follows: sampling selects and measures a subset of the valid configurations generating a partial solution space, and machine learning predicts the rest of the space based on the (training) samples. Focusing on the domain of IoT/Edge/Cloud, we have also reviewed some approaches that estimate the energy footprint of code. GreenScaler [14], Palladio [15] and HADAS [16] are good representations of these tools. But again, they all require specific initial domain knowledge to start the energy analyses (e.g., they need equitable samples).

Henceforth, this paper proposes the Smart Analyser of Variability Requirements of Unknown Spaces (*SAVRUS*) approach. SAVRUS uses sampling, machine learning, and statistical methods for large and unknown spaces of complex QAs [17]. SAVRUS searches for the specific features and interactions statistically affecting a QA, like energy consumption, and provides a clear report classifying them by their learned influence sufficiently fast. Simply put, SAVRUS squeezes any available data to detect features-QAs relationships smartly. Our objective is to guide developers in optimising their products by pointing to the specific features with a high probability of improving a QA if their alternatives replace them. The main novelty of this tool is that it smartly combines methods for analysing configuration spaces, considering large VM with scattered quality measurements. While we share some existing techniques with other works, a complex combination adapted to work at the configuration level without relying on specific domain knowledge distinguishes SAVRUS from current alternatives. The novel contributions of our paper are:

- Identification of the challenges imposed by the variability modelling of emergent systems with huge configuration spaces and multi-features dependent QAs, such as energy consumption. These challenges prevent using most adopted sampling and machine learning techniques (Section 3);
- Reasoned selection of the most suitable sampling, statistical and learning methods and their configurable parameters for complex systems and QAs (Section 4);
- The modular SAVRUS approach and the Algorithm 1 of Section 5.1 integrate selected methods to detect pairwise quality insights in large, initial domain unknown spaces defined by heterogeneous systems;
- We performed an empirical evaluation of SAVRUS in Section 6 for a large case study using empirical data from our domain (i.e., Watts) — a generic energy-efficiency edge computing system [4] comprising 554 features and a total of $10^8$ legal configurations; and
- We present an empirical validation of SAVRUS in Section 7 with a set of completely measured real systems while discussing its accuracy and scalability.

SAVRUS prototype is available at:
https://hadas.caosd.lcc.uma.es/savrus

## 2. Problem statement

In this section, we are going to discuss relevant issues for the SAVRUS proposal regarding (1) QAs modelling in highly configurable systems, (2) interacting features and their influence on specific QAs, and (3) sampling and machine learning techniques.

### 2.1. Variability and quality modelling

*Featured-Oriented Domain Analysis* (FODA) is the standard to represent high variability — i.e., a domain analysis method based on the concept of features used to describe the commonalities and variations of systems. With FODA, we specify the variability of functional requirements and generate configurations, i.e., the configuration space of a system. FODA-type VMs include simple feature models [18], Clafer models [19], *Extended VMs* (EVMs) [8], to name a few, but only a subset of them include numerical features [1]. In this work, we use *numerical VMs* that are very suitable to model parameters that influence energy consumption as numerical features (e.g., feature message size), and from now on in this paper, **VM will refer to numerical VMs**. Hence, VMs consist of a tree-like hierarchical structure where the nodes are Boolean or numerical features (i.e., variables) affected by the defined branches' cardinality (e.g., only 2 of the 3 optional branches can be selected). Additionally, VMs support cross-tree relationships, a combination of first-order logic and arithmetic equations (e.g., (#Antennas + #RJ45s) = 0 implies not Network).

On the other hand, the QAs specify the non-behavioural aspects under which the system operates [20] (e.g., performance, energy consumption, security level), where all the possible values form the QAs measured space. There are two main alternatives to measure the QAs of VMs: per feature and configuration. For simple QAs whose distribution profile is a linear equation (e.g., additive), measuring per feature is feasible — this is the case of EVMs [8,21,22]. However, complex non-linear QAs depending on many interacting features and external factors, like energy consumption, cannot be measured per feature. For instance, could we

define a static energy consumption value of a specific microprocessor independently of its usage and overall hardware? Instead, code energy footprint should be measured per configuration [23,24]; for example, code migration in Edge computing depends on other features like code size, communication protocol, and virtualisation mechanism, among others.

In this scenario, **we consider that energy consumption values are directly linked to complete configurations by their specific leaf nodes**, as the tree hierarchy naturally traces back the non-leaf features up to the root, as recently proposed in [10]. From an implementation point of view, complete configurations' QAs can be integrated into the VM [10,19] or be hosted in data storage repositories as databases [16,25–27]. Independently of the QA nature, the ***measured space*** **denotes the set of complete configurations for which we have the corresponding QA measurement**.

### 2.2. Interacting features and normality tests

The literature often assumes that individual features affect QAs homogeneously; in other words, they behave equally independently of other features in every valid configuration. However, often, that scenario is very unlikely [28]. Interactions are identified when one feature behaviour is influenced by the presence of another (or a set of others). Furthermore, interactions sometimes cannot be deduced from individual behaviours, hindering compositional reasoning. This circumstance leads to scalability problems, as the number of potential interactions in a system is exponential in the number of features. The higher the order of interactions, the harder to detect them (e.g., pairs are a quadratic increase, and triplets are cubic). We must highlight that interactions are not simple equations (e.g., adding the proportional energy consumption of features or taking the absolute peak).

In this scenario, to ensure that the features and interactions identified as the ones most affecting a QA are meaningful, we can check that the analysed samples are as generally distributed as the system — the null hypothesis. In the context of our problem, a statistical normality test tells if two sets of samples have equally distributed interactions. Unfortunately, most real-world data sets and relationships are non-normal, and large sets will always reject the null hypothesis [29]. To overcome this, the variability community uses non-parametric tests. The most common ones are the Student's t-test, the Unequal Variance t-test, and the *Mann–Whitney U test* (MWU). The most used is the Student's t-test, which is accurate for assumed normally distributed data. On the other hand, it is stated that the Unequal Variance t-test is superior to comparing the central tendency of 2 populations based on sets of unranked data. To sum up, the most accurately flexible is the MWU, independent of the sets' distribution, size and ranking. Furthermore, for ranked data, the MWU is superior to the alternatives [30]. Therefore, the **MWU is the best test to assess whether a particular feature or interaction influences a QA, as energy footprint**.

### 2.3. Sampling alongside machine learning

Traditionally, we find two domains that tackle the scalability issues raised, aiming to reduce the search space by avoiding the generation and QA measuring of every configuration. The first one is *sampling*, the selection and measurement of a subset of the search space formed by pairs [configuration, QA] and [feature, QA] [11]. As sampling is used in many areas, we can find many types depending on probabilities, knowledge and heuristics [31–33]. The second is *machine learning*, a vast conglomerate of predictive and self-learning techniques [13]. Some of them are empirically demonstrated to be effective with small learning sets [17], which is crucial to improve scalability by reducing the number of requested samples. We must highlight one of our interests: *Transfer Learning* (TL). TL registers knowledge obtained while solving specific problems and applies it to another similar problem. A simple example is extrapolating Android 11 insights to the Android 12 future analyses.

To cope with IoT/Edge/Cloud/B5G domain requirements and the difficulties of estimating their energy consumption, in this paper, we assume that **our method has to work with domain-unknown configuration spaces, with scattered measurements, by using some of these techniques**. But the question is, which of these techniques or a combination of them fits the requirements of our application domain IoT/Edge/Cloud/B5G.

## 3. Our objectives and goal

While numerous techniques and related works seem applicable to analyse large VMs, energy consumption imposes concrete requirements that constrain their accuracy. As that prevents us from directly replicating past works, we have set the following objectives:

- **O1. Derive energy consumers and interactions without predicting absolute values.** Existing approaches declare that they work with exact QA values (e.g., runtime). However, when measuring energy consumption, we need to deal with external factors (e.g., temperature) and prediction inaccuracies, which are only sometimes considered by some measuring tools, going against that precision statement [5]. Thus, we aim to help software developers understand how incorporating certain features into their configurations can positively or negatively impact their energy consumption rates [7]; bearing in mind the difficulties of achieving exact energy measure values.
- **O2. Not relying on initial domain knowledge.** Techniques that predict exact values require measuring specific samples to be accurate, which is impossible for complex QAs [9]. We need smart techniques to improve software product line analyses without requiring particular or initial domain knowledge.[2]
- **O3. Fast analysis.** Using brute-force or predictive models in large spaces takes more time than a developer's patience [11,34]. We need an approach that generates insights in mere minutes — a practical approach.
- **O4. Support a variable search space.** IoT/Edge/Cloud systems tend to evolve quickly (e.g., new API variants). In these situations, we need to request new samples and re-adjust predictive models to keep accuracy [35]. For example, an IoT device's new, more powerful release extends the space, probably causing a higher consumption of other features (e.g., encrypting). On the other hand, requirements imposed by developers (e.g., a programming language) will automatically reduce the space by disabling several related features (e.g., libraries and functions). So, the proposed approach needs to consider that the results of the energy consumption analysis of certain features will be affected by removing other features from the analysis.

Synthesising, **our goal is to design an interactive approach that, using sampling, learning, and some available measures at hand, provides energy consumption (or similar QA) insights**

[2] The notion of *without domain knowledge* is used in this work as the lack of domain experts who can guide the learning of QAs values, which will hinder most guided machine learning techniques. We only know the variability models and their reusable features, and cannot assume we have QA measures of any concrete number or type of configurations.

**to IoT/Edge/Cloud system developers in the range of minutes**. Consequently, developers can constrain the search space, quickly understanding how that may affect the energy consumption of other features and product configurations.

## 4. Motivated design choices

This section provides the motivations behind our selection of techniques, arguing why others are discarded due to breaking our objectives or simply because they are less suitable for the energy consumption QA. For instance, proposals addressing attributed VMs are discarded, as, in general, feature-level techniques do not scale to the configuration level, including the ones requesting a subset (e.g., sampling or learning). The reason is that the space formed by the different configurations is exponentially larger than the one formed by their respective features.

### 4.1. Product sampling techniques

The range of techniques that potentially matches O1 and O4 belongs to *probabilistic sampling*, which includes the ones employing some randomness (i.e., equal probability of selection) [36]. In that case, the most representative strategies are:

- **Uniform Random Sampling** (URS) [37]: We select samples randomly. Its accuracy depends on the uniform distribution of the search space and the randomiser.
- **Statistical Recursive Search** (SRS) [37]: It is a refined URS with recursion by fixing statistically meaningful features.
- **T-wise sampling** [38]: We should sample one valid configuration per $T$-tuple of features (i.e., discrete combination). While a thoroughness of one (i.e., $T = 1$) is one sampled configuration per existing feature, $T = 2$ selects one configuration per pair of existing features, and so forth. Depending on $T$, the accuracy and computationally demands vary in an increasing pattern; up to 2 is considered balanced, while 6 reaches exponential cost [39].
- **Distance-based Sampling** (DbS) [40]: It is a faster T-wise where we select fewer samples and up to $T$ interactions, based on a configurable distance metric and probability distribution. Distance-based metrics force sample diversity compared to random approaches.
- **Diversified Distance-based Sampling** (DDbS) [40]: It is a refined DbS where diversity is increased by prioritising configurations containing the least frequently sampled features and interactions.

We discarded the ones not offering experimentally demonstrated advantages over the rest. Based on [40], DDbS outperforms other strategies for incomplete and probably biased search spaces, also having a lower computational effort. On the other hand, SRS is likewise empirically demonstrated to be superior to URS [37] in terms of accuracy but with even higher computational effort. Hence, **we select SRS and DDbS as they are the best strategies based on different metrics while matching O2 and O4 (i.e., fast analyses without domain knowledge**).

Notice that we do not use probabilistic sampling like in the literature to generate a learning set — a requirement for predictive techniques that breaks O1. The most famous examples are Guo et al. [41], which uses them with classification and regression trees to generate performance models, and SMTIBEA, which adds the indicator-based evolutionary algorithm (IBEA) [42]. This usage would break O1 thru O3 as it requires specific configurations to be measured dynamically and on-demand, often run out of memory, does not terminate, or the calculation takes too much time [11]. Instead, we consider the samples as a simplified representation of the measured solution space at the configuration level and directly analyse them. In any case, the ideal scenario for the best accuracy is a uniformly distributed and representative sample set and a small homogeneous measured space to make accurate QA predictions — utopia in the real world, but standard in academia [43].

Other types of sampling could have been guided sampling (e.g., adapted progressive and projected sampling [44], L2S [45], and genetic sampling [46]), alongside their specific learning techniques. However, they would break O2 as they require specific initial samples to achieve low prediction error, runtime measuring, and small sizes [47].

### 4.2. Interacting feature analyses

Regarding interacting features analysis, based on reviews [28, 48], direct approaches break O4 due to the poor scalability as detailed in Section 2.2. Another conclusion is that detecting external interactions, such as those with QAs, requires heavyweight and dynamic analysis techniques. Large-scale T-wise sampling and coverage are supported in the YASA tool [49]. However, initial domain knowledge is needed for the heuristics, which breaks O2. The effects of worsening granularity levels are analysed in [50]. Fortunately, Jeho et al. [38] conclude that higher-order interactions are less likely to occur, advising a maximum of 3-wise coverage. This matches [12,51], which suggests pairwise coverage as sufficient, meaning that it is unnecessary to test triples, quadruples, etc. Additionally, it is stated that lower-order interactions increase the probability of higher-order interactions [28]. Consequently, **if we analyse up to pairs of features already affecting a QA, we can generate minimal sample sets matching O4. This ensures that we will cover meaningful feature combinations.**

### 4.3. Machine learning techniques

Among existing machine learning techniques, *lazy learning* approaches are model-free and have no training stages, thus relying on sample representativeness. In other words, lazy learners postpone the learning process until an analysis is requested, and in this moment they only use the available measured space. Lazy learning is the proper approach when we have no assumptions about the current shape of the search space (i.e., O2) [52].

The most popular lazy learning strategy is the *k-Nearest Neighbours* (*k*NN). *k*NN algorithm is commonly used for classification problems by assigning the mode, the value that appears most often, of the k-closest observations as the predicted value. However in SPLs, different configurations rarely have the same QA values, especially if they are real numbers, such as the runtime or energy consumption (e.g., two code executions rarely have the same run times). Consequently, we will use *k*NN as a regression technique by assigning the mean of the k-closest observations. *k*NN accuracy and performance depend on a few configurable parameters such as the distance metric, where the Euclidean is the baseline, and the *k*-value. Nevertheless, even for a modest *k* and a few measured samples, *k*NN performs sufficiently fast and accurate [53].

Further, we are interested in ranking the features based on their constant influence on a specific QA. The ranking is a list of features with an assigned likelihood that defines the order. Instead of purely relying on a single ranking analysis per execution, we can keep track of the frequency of feature detection to create or update the ranking by *transferring* that information into current analyses. However, past-experience counting will no longer be accurate as the variability model evolves and/or the user includes desired constraints. When this occurs, the configuration space, and in turn, the usage scenario, can change, so the

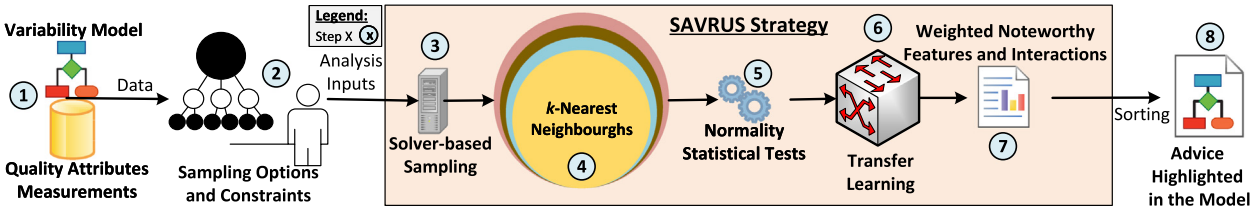


**Fig. 1.** SAVRUS approach overview highlighting meaningful components.

concluded general ranking will not always match the selected features. For example, a specific programming library could be highly affecting the energy consumption of a system, but in a specific sub-system, it could behave equally as the other alternatives, so the past ranking will not be accurate in the new analysis. Then, **we need a transfer learning technique where the source and target domains, and the analysis, are similar but not equal; this is studied in *Inductive TL*** [54]. Additionally, we can build the ranking based on the weights of a *Scoring Function* [55]. The score function is the gradient of features' logarithmic likelihood for the confident values of the function. Confident scores allow ordering a set of values to compare to one another easily and indicate the steepness of the function (i.e., the sensitivity to changes) [56].

As with sampling, the machine learning literature focuses on *prediction* [57], but unfortunately, this breaks objective O1. For instance, DECART [58] avoids verification samples, BEETLE takes advantage of the bellwether effect (group behaviour) [59], Jamshidi et al. [60] correlates performance models, and LOKI only considers the feature space [61]. Independently of that, off-the-shelf approaches do not necessarily succeed with our real-world requirements. Variability domains are complex due to the substantial constraints among features and data scarcity; measuring QAs in real-world systems could be complicated and tedious [17]. Additionally, better adjustments for specific domains degrade other domains' accuracy — typical in supervised learning techniques such as general linear regression, support vector regression, and *k*NN [62]. While unsupervised methods such as neural networks have the potential for domain-unknown approaches, to date, they underperform when applied to configurable systems [63].

## 5. SAVRUS approach

SAVRUS aims to quickly detect and rank interacting features that noteworthily affect QAs in models with domain-unknown QA spaces and dispersed QA values. Our approach integrates sampling, learning and statistical strategies without relying on training samples or specific on-demand measurements. While Section 5.1 provides the approach overview, the algorithm with the logic of the SAVRUS service and its implementation details are described in Section 5.2.

### 5.1. SAVRUS overview

Fig. 1 shows the SAVRUS overview, which consists of 8 sequential steps numbered from left to right. ① and ② are data and user inputs, ③ to ⑦ are the analysis strategy, and ⑧ is the final report. A user completely performs the sequence in each execution analysis.

We now detail a user perspective. Let us consider an IoT *application engineer* or her enterprise, which is worried about climate change, so they want to know what feature(s) of their software product line impact the energy footprint. Let us consider that an IoT/Edge product line exists and that the leading application engineer wants to deploy its devices and applications to be Ultra-Low Energy (ULE). However, the number of options is so large, complex and counter-intuitive that brute-force and on-demand predictive tools are inaccurate in the best case. She decides that the best choice in this situation is to use SAVRUS.

Consequently, she builds and measures the energy consumption of a small fraction of the options based on the time, components and tools she has available. Then, after configuring, partially or totally, the VM according to her case study and specifying some energy-related constraints (step ①), she executes SAVRUS with some parameters (i.e., sampling options and constraints ②) as a black box (steps ③–⑦) and receives an analysis report (step ⑧). She obtains a set of ranked features identified as energy consumers — they are potential candidates to be substituted for decreasing the energy consumption of the case study. She also obtains a graphical VM, where she visualises the alternatives that consume less energy and makes decisions (e.g., changing the operating system of some devices).

Going into details, ① represents the configurable system linked with some QAs measurements. As highlighted in Section 2.1, in practice means connecting a VM containing features and relationships with a QA repository implemented as a relational database that stores QA values for each configuration. These measurements presumably will be less than desired and not uniformly distributed. Still, in ① this is all we have since the development team does not dedicate more resources to energy footprint experiments. Step ② represents the application engineer inputs, which are the user requirements (i.e., selected features and quality constraints), the maximum number of samples to consider and the $k$ parameter of $k$NN. Generally, requirements (e.g., operating system = Android, energy consumption $\leq$ 3 Watt) with a higher number of samples and $k$ would produce better results at runtime cost. Steps ③–⑦ represent the SAVRUS strategy. The first part, step ③, is any sampling method. While we have used probabilistic sampling for domain-unknown analyses, SAVRUS is compatible with most guided, heuristic or solver-based sampling methods — although, the chosen method will impact the coverage and accuracy of the analyses.

Continuing, in step ④, we include $k$NN to try to provide an approximated value for the QA unmeasured samples if they are requested by step ③. If all the requested samples are already measured, $k$NN is not necessary (i.e., not executed). Additionally, if $k$-closer samples do not exist, the requested sample is discarded.

At this point, having just QA valued samples, step ⑤ is a statistical test that identifies the noteworthy and noticeably interacting features affecting a QA based on the sample data set. The notion of noteworthy features is coined in [64–66], where normality tests are used with a threshold of 95% confidence. The identified effects depend on the objective; they could be positive (i.e., improving/maximising a QA), negative (i.e., degrading/minimising a QA), or even more complicated (e.g., trade-offs).

Next, we use a TL method in step ⑥ to keep a dynamic track of the noteworthiness scoring of their features and their t-combinations. This enriches SAVRUS's current analyses based on previously detected noteworthy interactions. Hence, in ⑦, SAVRUS uses the updated scoring weights to rank feature combinations — the highest the weights and lowest the rank, the

strongest is the features-QA interaction. For example, if our objective was to minimise energy consumption, we could set the rank as negatively affecting it. Then, if the CPU of a system is at the top of the rank (i.e., rank 1), it must be the first to be replaced to reduce the system's energy footprint.

Finally, step ⑧ is the output of SAVRUS, which comprises: (i) an ordered list of features and interactions identified as notably affecting a QA, concerning a specific goal (i.e., minimising energy consumption); and, (ii) the VM provided as input in ① graphically highlighting and numbering features and interactions based on the previous list. While a developer pragmatically uses the list to improve the analysed system, the graphical VM provides an overview of the domain knowledge from the energy consumption point of view.

### 5.2. SAVRUS algorithm

**Algorithm 1:** SAVRUS analysis strategy

```
   Data: Variability Model VM, Configurations Quality
         Measurements CQMs, Ranking Scoring RS
   Input: Number of Samples #S, user Requirements Rs,
          Nearest Neighbour Parameter K
   Result: Ranked Worst Influencing Features and
           Interactions
1  if Rs then // User constrained the search space
2  |  VM ← Constraint(VM, Rs);
3  |  CQMs ← Constraint(CQMs, Rs);
   /* Generate #S valid configurations a solver
      and a sampling algorithm                    */
4  Samples = Sampling(VM, CQMs, #S);
   /* Unmeasured quality values are the distanced
      weighted mean of k near measurements        */
5  foreach S ∈ Samples do
6  |  if ¬S.QualityMeasurement then
7  |  |  S.QualityMeasurement ← KNearestNeighbours(K,S);
8  TUF ← TopUnderperfomingFeatures(Samples);
9  NoteworthyFeatures ← MannWhitneyU(TUF);
   /* Having detected influencing features, we
      analyse their pairs and QAs interactions    */
10 foreach F1 ∈ NoteworthyFeatures do
11 |  foreach F2 ∈ NoteworthyFeatures ∧ (F1 ≠ F2) do
12 |  |  if PairWiseMannWhitneyU([(F1,F2)], Samples) then
13 |  |  |  Append(NoteworthyFeatures,[(F1,F2)]);
   /* Influencing features and interactions imply
      recalculation of their scoring              */
14 TransferLearningScoring(RS, NoteworthyFeatures);
15 foreach F ∈ NoteworthyFeatures do
16 |  F.NoteworthinessWeight ← RS.F;
   /* Sort interactions/features by learned
      scoring                                     */
   Result: SortByWeight(NoteworthyFeatures)
```

The core of the SAVRUS approach is Algorithm 1, whose implementation can be found at.[3] The calling parameters *Data* and *Input* are steps ① and ②, respectively. At the same time, *Result* is the ranked list of identified affecting features — the arrow to step ⑧. In summary, ①, ②, and ⑧ are part of the front end. In the back end, the algorithm 1 shows the logic of the decisions taken to analyse configurations and QA measures (SAVRUS Strategy box at Fig. 1). The front-end is a web application, whereas the back-end is developed in PHP 8, and a MariaDB 10 SQL database stores the QAs' measurements. In this prototype, the VM supports Boolean and numerical features (i.e., numerical VMs). While in this subsection we present specific techniques, each step of the approach is modular and can be replaced by other alternatives (e.g., different sampling, learning and statistic techniques).

Lines 1 to 3 of algorithm 1 correspond to the analyst constraining the VM and QA. An example is that a SAVRUS user indicates that "the operating system must be Android" and that "the energy consumption must be less or equal to 3 Watts". Practically, constraining the VM means to indicate in step 2 those constraints, either graphically by clicking/tapping the required and excluded features or as text-based propositional clauses for more advanced relationships (e.g., *(Feature X and Feature Y) exclude Feature Z*). Then, the automated reasoner will always limit the configuration space by considering the user constraints as part of the VM logic. Additionally, while the QA constraints are indicated in the same format, they are routed to the database queries, and hence the database manager will further limit the measured space. If the automated solver requests a sample, it could be discarded if the QA measurement value does not match the user constraints. Alternatively, it can be replaced by a different one generated by a new run of the automated solver. Considering those feature selection constraints in the space, we obtain the number of measured samples requested by the analyst in the following line. While we have motivated SRS and DDbS strategies in Section 4, we could replace them with other alternatives without affecting the rest of the algorithm. We implement SRS and DDbS Clafer solver-based,[4] implying several solvers calls with loops and different search seeds. We aim to generate a discrete uniform distribution with fast (pseudo-random) procedures for the randomisation tasks used by several methods. For this, we use a combination of the Clafer self-randomiser and the PHP8 *openssl_random_pseudo_bytes()* function meant for cryptographic applications.

In the following lines, we implement the *k*NN and MWU techniques, based on the PHP-ML library.[5] In lines 5 to 7 *k*NN tries to approximate QAs values when the sampling method requests unmeasured samples. As a lazy learning approach, only when sampling requests unmeasured configurations, it generalise to calculate the [configuration, QA] tuples based on the mean of the k-closest measured ones. We can find several distance metrics already implemented in PHP-ML. In SAVRUS, we use the *Manhattan distance*,[6] also called the *City-Block distance*, as it is the recommended one in the literature for the variety of dimensions that the configurations of our application domain comprise [67]. To make the reasoner output and the PHP-ML distances compatible, note that the generated samples are complete configurations and that configurations are a set of features. Consequently, we transform samples as collections, as we can vectorise them as unordered feature sets with variable vector widths. We avoid the tree root feature, as it is shared by all the complete configurations, reducing the computational effort. Hence, the difference in size and the counting of features not present in the conjunction (i.e., counting the exclusive disjunction) increases the distance between samples. Regarding the *k* hyperparameter, while there are techniques to predict accurate ones, they require pre-training stages [68], which would break **O2**. Hence we decided to leave it user-defined, as equal to the number of samples, larger *k*s would potentially result in more accurate QA means but with a

[3] SAVRUS prototype: https://hadas.caosd.lcc.uma.es/savrus.

[4] Clafer Suite [19] is the state-of-the-art variability-solving reasoner composed of a modelling language, a numerical variability reasoner, a multi-objective reasoner, and a multi-dimensional interface. Given a VM in Clafer modelling language, the Suite can randomly generate parts of the respective configuration space and search for optimal configurations based on feature attributes.

[5] PHP-ML library: https://php-ml.readthedocs.io/en/latest/.

[6] Manhattan distance of $N$ dimension $= |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_N - y_N|$.

proportional increase in computational cost and time. Therefore, by default, we set $k$ as the minimum for a mean (i.e., 2), meaning assigning to the unmeasured samples the QA value of the mean of the two most similar measured configurations.[7] If $k$-closer neighbours are unavailable, the sample is discarded.

In line 8 SAVRUS identifies which features are shared among the worst performing configuration samples for the QA that we are analysing. For this, it starts by computing the average performance $\overline{P}(f)$ of configurations with a feature $f$, and the average performance $\overline{P}(notf)$ of configurations without $f$. Consequently, SAVRUS calculates the influence of a feature such as follows:

$$P\Delta(f) = \overline{P}(f) - \overline{P}(notf)$$

For QAs like runtime or energy consumption, where a higher QA value is underperforming, a $P\Delta(f) > 0$ means that $f$ degrades the QA performance of most of the configurations in the sample set. Further, SAVRUS generates a set with the features shared among the top $N$ underperforming configurations from the sample set. Based on [37], where a similar approach is used to find the best-performing features, it is experimentally concluded that an $N = 2$ is the most accurate. Elaborating, using only the worst configuration (i.e., $N = 1$) is misleading, as only some its features are potentially noteworthy. On the other hand, $N \geq 3$ is too constraining, as interesting features may not be in all configurations.

SAVRUS continues in line 9 performing an MWU test to confirm that those interacting features significantly affect a QA with a 95% confidence. In other words, we check if the influence of those potentially noteworthy features is statistically significant. Technically, confirming the null hypothesis, SAVRUS checks that the current set is sampled from the same distribution as a new random one containing those interacting features. In other words, it verifies that those interactions are expected for most of the configurations of the configuration space.

At this point, we have the measured sample set and the statistically noteworthy features set. In lines 10 to 13, we apply [28,51] by performing the same MWU tests to pair combinations of the noteworthy features set. Hence, while SAVRUS does not cover all possible interactions, it potentially identifies the most common and meaningful ones affecting a QA with the minimum computer resources. The loop ends with a pairwise noteworthy interacting feature set.

Finally, we keep track of the influence scoring, which is the composed likelihood of features affecting a QA based on the accumulated search spaces. We apply TL in lines 14 to 16. We follow a similar approach to TL anomaly detection with unsupervised instance selection [69], but in our case, the ranking scoring is shifted opposite of the anomaly. Anomaly detection involves identifying QA-valued configurations containing noteworthy features or interactions as outliers, meaning that their behaviour differs from what was expected. Suppose that in an analysis, we check if that noteworthiness conclusion holds for the $k$-close configurations in which they are present. If the effect of that feature or interaction in those exact configurations is an outlier, the conclusion does not hold. Hence, the scoring will decrease proportionally to the outlier scoring. Similarly, if the conclusion holds, the ranking scoring increments inversely proportional to the outlier scoring.

In this implementation, we rely on the PHP library Rubix ML.[8] Concretely, we make use of its *Local Outlier Factor* (LOF) method, which measures the local deviation of the density of an unknown sample, regarding its $k$NNs. One can identify the configuration with a substantially lower density than their neighbours by comparing the local density of a configuration to the local densities of its $k$NNs. As such, LOF only considers the $k$-neighbourhood of an unknown configuration, which enables it to detect anomalies within individual clusters of the measured configuration space. LOF can be executed as unsupervised or as semi-supervised learning. The unsupervised mode finds outliers within itself (i.e., the configuration space). On the other hand, the semi-supervised mode allows comparing the density of a given configuration versus the density of the $k$-closest neighbours in the measured configuration space. We execute the semi-supervised mode as we considers an initial weight of 0 (i.e., not affecting) for each feature and interaction and constantly update the weights.

As implicitly mentioned, we make use of $k$NN again, but this time within the LOF method, which is a sequence of:

1. Defining the distance metric, the $k$ hyperparameter, and the algorithm used to compute the NNs. In this version, we keep using the Manhattan distance and $k = 5$ for the same reasons detailed above. Regarding the specific algorithm, Rubyx ML implements the *k-d Tree* and the *Ball Tree* algorithms, from which we choose the latter. The reason is that the Ball Tree algorithm works well with various dimensions since the partitioning schema does not rely on a finite number of 1-dimensional axes aligned splits such as with *k-d* Tree. Ball Tree is a binary spatial tree that partitions the measured configuration space into successively smaller and tighter configuration clusters.
2. Calculating the $k$th distances of the samples. As mentioned, we chose the Manhattan distance again.
3. Pulling $k$NN configurations from the measured configuration space that contains the noteworthy feature or interaction to rank. If $k$-closer configurations do not exist, the scoring of that feature or interaction will not be updated.
4. Calculating each samples' Local Reachability Density (LDR) by estimating the distance at which a sample can be found by its neighbouring configurations. Mathematically, LDR is calculated as the count of the configurations in the specific $k$NN set for a sample over the Reachability distance of the sample to all the values in its $k$NN set. Formally, the LDR of a configuration sample X:

$$\mathrm{LDR}(X) = \frac{\#KSamples}{\sum_{Y=KSamples[1]}^{\#KSamples[N]} \mathrm{Reachability}(X, Y)}; \text{ where}$$

$$\mathrm{Reachability}(X, Y) = \mathrm{Max}\{k\mathrm{Distance}(Y), \mathrm{Distance}(X, Y)\}$$

5. Calculating the final LOF value of each sample. It is calculated as the sum of the LRD of all the samples in the respective $k$NN set, multiplied by the sum of the Reachability distance of all the samples in that $k$NN set, all divided by the square of the size of that $k$NN set. Formally, the LDR of a configuration sample X:

$$\mathrm{LOF}(X) = \frac{\sum_{Y=KSamples[1]}^{\#KSamples[N]} \mathrm{LDR}(Y) * \sum_{Z=KSamples[1]}^{\#KSamples[N]} \mathrm{Reachability}(X, Z)}{(\#KSamples)^2}$$

6. Defining a threshold to obtain conclusions based on the resulting LOF values. Based on the literature [70], if the measured configuration space is tight, clean, and uniform, LOF values above 1 mean that how a feature or interaction performed in the sample is an outlier and cannot be generalised. However, in cases like emergent domains like ours where the measured configuration space is probably sparse, with a varying density, and with local fluctuations in a local cluster, samples are considered outliers with LOF values above 2.

[7] While in classification problems the $k$ hyperparameter must be an odd number to ensure no ties in the voting, this restriction does not apply to regression problems.

[8] Rubix ML library: https://rubixml.com/.

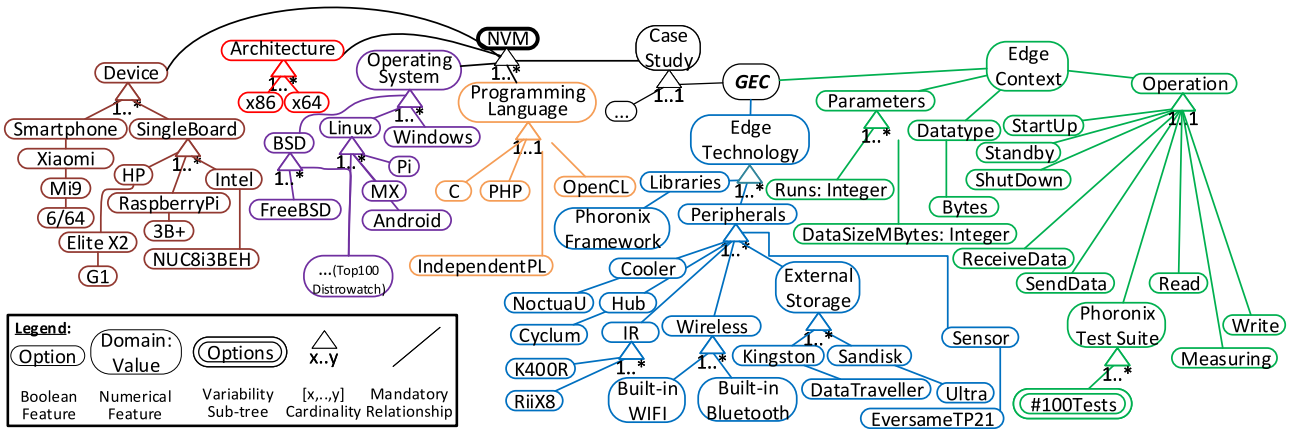


**Fig. 2.** VM of the *Generic Edge Computing* (GEC) large case study.

To clarify, instead of feeding the LOF method with sets of features forming valid configurations as we did in the *k*NN step, it analyses a single set of QA measurements belonging to the set of valid configurations containing the feature or interaction to rank. The number of sets to analyse is the total number of features and interactions to rank.

Another clarification is about calculating a *k*-mean QA value in the *k*NN step and then calculating the LOF value that again uses *k*NN. First, they are two different and necessary instances of *k*NN. Second, their inputs and outputs are other. In the *k*NN step, the inputs are unmeasured configurations, and the analyses rely on the user-constrained measured configuration space. In the TL step, the inputs are single features and sets of interacting features, and the LOF analyses rely on the features-constrained measured space. On the other hand, the outputs are the *k*-mean value of unmeasured configurations and a ranking scoring, respectively.

The ranking scoring goes from 0 to 1, meaning not affecting and always affecting, respectively. As mentioned, features and valid interactions start with weights of value 0 (i.e., generally not affecting) by default. On each SAVRUS execution, if any of them are detected as statistically noteworthy, the TL step with the LOF method is executed. Then, the weights are updated depending on the respective LOF values. If the LOFvalue of a feature $X$ is above the anomaly threshold, 2 in our case, it is normalised and subtracted from its scoring weight $w_X$ as $w_X \mathrel{-}= \text{LOF}(X) \div 100$. On the other hand, if the LOF value is below the threshold, it is added as $w_X \mathrel{+}= \text{LOF}(X) \div 100$. The weights cannot decrease/increase more than 0/1, respectively. Finally, features and interactions are ordered from 1 to 0, creating the noteworthiness ranking of features and interactions affecting a specific QA. This is the result and output of the algorithm.

## 6. A proof of concept

This section is a proof of concept that accomplish Section 3 by analysing an extensive IoT/Edge/Cloud system with our SAVRUS prototype. We present a case study scenario aiming to detect the features' influences on the energy consumption rate of that system. In the scenario, SAVRUS helps an IoT/Edge/Cloud developer to decide on specific configuration changes to reduce the energy consumption rate of that system while considering her recurrent requirements.

As mentioned in Section 5.2, the prototype supports SRS and DDbS strategies. They will be tested as, to our knowledge, they have yet to be confronted in the literature. We use the *Generic Edge Computing* (GEC) case study — a VM with a large configuration space we have designed to represent a regular IoT/Edge/Cloud system [71]. GEC VM is represented in Fig. 2, and it comprises six main branches:

- **Device**: Edge computing hardware, as single-board and small devices; in this evaluation, four were considered.
- **Architecture**: The microprocessor running architecture — the most common ones are x86 and x86_64.
- **Operating System** (OS): The running OS in which the software will be executed, where, besides the top 100 UNIX systems in use in 2021 yearly published by Distrowatch,[9] Microsoft Windows is considered. Only the latest updated version and the default kernel were considered. Additionally, we defined cross-tree constraints for unsupported OSs.
- **Programming Language** (PL): The PL in which the operation is coded; in our evaluation, just the ones of the executed benchmarks are considered.
- **Edge technology**: Our available libraries and peripherals are considered, including wireless communication, data storage, temperature sensors, remote controllers, etc.
- **Edge Context**: Three key branches are located at this level. **Parameters** contain the numerical features as natural numbers, which usually are input calculation parameters. **Datatype** represents the data types used in a specific operation. We used bytes for cases where several types are used simultaneously, like in benchmarks. **Operation** contains the tasks performed in an IoT device. Besides the common ones, such as starting or shutting down a device, the benchmark Phoronix Test Suite[10] operations are included in our evaluation. The suite ranges from battery power consumption monitoring for mobile devices to multi-threaded ray-tracing benchmarks and spans the CPU, graphics, system memory, disk storage, and motherboard components. While the suite comprises 403 tests, not all of them suit every OS or device of the VM, but, on average, 100 are compatible with each system. Representing all the OSs, operations and cross-tree constraints, Fig. 2 would not be graphically friendly. The complete Clafer VM can be downloaded in the footnote link.[11]

Following the planning process for the energy-efficient software [72], we measured GEC energy consumption rate QA in *Watts*,[12] triple checking with three professional tools: Watts Up Pro Portable Power Meter, Multimeter Eversame C, and Eversame

9 Ranking of UNIX systems in use in 2021 yearly published by DistroWatch: https://distrowatch.com/index.php?dataspan=2021.

10 Phoronix Test Suite: https://openbenchmarking.org/tests/pts.

11 GEC Numerical variability model download as Clafer model: https://hadas.caosd.lcc.uma.es/edgenvmfgcs.txt.

12 Watt is the derived unit of a joule per second and is used to quantify the energy rate.

**Table 1**
Detailed real-world variability models ordered by their search space size, of which GEC QA is incompletely measured.

| NVM | Description | #Booleans | #Numericals | Space | QA | #Measurements |
|---|---|---|---|---|---|---|
| Dune | Multi-grid solver | 11 | 3 | 2,304 | Equation solving time | 2,304 |
| HSMGP | Stencil-grid solver | 14 | 3 | 3,456 | Equation solving time | 3,456 |
| HiPAcc | Image analysis framework | 33 | 2 | 13,485 | Equation solving time | 13,485 |
| Trimesh | Triangle mesh library | 13 | 4 | 239,360 | Equation solving time | 239,360 |
| GEC | Generic edge computing | **552** | 2 | $\sim$**5.3** $*$ **10**$^{8}$ | Energy Consumption | 132,500 |

PowerMeter 2n1. The devices were highly cooled in a quiet and isolated room at all times to avoid external factors affecting the readings.

Clarifying, we will analyse GEC in four different scenarios, chasing four sets of results. In the first one, we run Clafer chocosolver[13] for the GEC VM to generate the entire solution space, checking that it is a correct VM. The second scenario is the perspective of a random GEC systems developer interested in improving its products' energy consumption by knowing which are the most efficient component alternatives (i.e., features). In this scenario, the developer acts as a SAVRUS user who interacts with its graphical web interface by running a sequence of SAVRUS instances with different analysis requirements. For the third perspective, we execute two sets of SAVRUS instances for a few VMs, including GEC. We focus on the averaged accuracy, coverage and runtime of each combination. In the last perspective, we evaluate the scalability of SAVRUS for colossal QA spaces like the one of GEC.

Consequently, we have written the GEC VM of Fig. 2 in Clafer modelling language comprising 552 Boolean and 2 numerical features with parent–children and cross-tree constraints. As introduced in Section 5, we used that VM as an input to Clafer chocosolver and ran its reasoning command generating a solution space of $\sim 5.3 * 10^8$ configurations in $\sim$36 h. GEC VM details are also specified in the last row of Table 1, where we can see that among all the VMs analysed in this work, GEC is the largest one. We can assume that a team measuring the energy consumption of some GEC configurations is likely only to measure a pre-configured benchmark suite and similar configurations. In other words, it is doubtful, if not impossible due to the size of the configuration space, that the team would physically have and actively run most of the compatible combinations of devices, OSs, and software and the proper energy rate measuring tools. Then, to approximate the GEC study to a real scenario, we performed just 132,500 different measurements for the QA energy consumption rate, which account for 0.25% of the total search space. Additionally, to faithfully reflect the likely scenario we have just commented on, this partially measured space includes random clusters of measured and unmeasured configurations for the QA energy consumption rate. As detailed in Section 5.1, we need to store these energy consumption rate measurements with their respective configuration references in a QAs values repository; in the current SAVRUS prototype the repository is a MariaDB database.

Evaluating fresh versus established instances of SAVRUS will not differ in detecting the noteworthy features and interactions. On the contrary, it could affect the resulting ranking of those features. The reason is elaborated in Section 5.2, and it is related to the TL module (i.e., step 7) where we initially set the TL scoring weights to 0 (i.e., not affecting). To prevent this and reflect actual usage of SAVRUS, we performed an initial set of 1000 SAVRUS runs with random user parameters generated by the PHP8 *openssl_random_pseudo_bytes()* like the strategy to use,

[13] Chocosolver is one of the two variability reasoners of Clafer Suite. Chocosolver is compatible with Boolean and numerical features and arithmetic constraints, and it can automatically generate between 1 and all possible configurations of a VM in partially random orders.

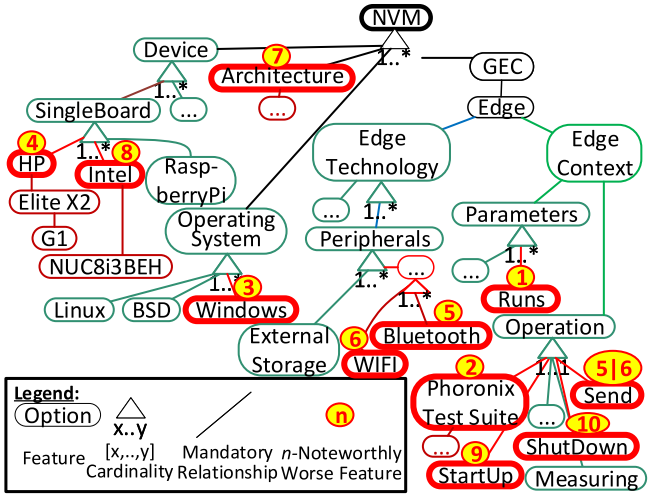


**Fig. 3.** Improving GEC energy consumption rate by switching the noteworthy red by alternative green features.

the number of samples, and feature constraints. After that, we conducted 100 runs of the SAVRUS prototype and averaged the results for a confidence level of 95% with a 10% margin of error [73]. In this second set of instances, $k$ parameter was left by default ($k = 2$).

> ***The GEC averaged ranked list of the most to the lesser influential top 10 noteworthy features and interactions affecting the energy consumption rate was:*** 1. Runs, 2. Phoronix Test Suite, 3. Windows, 4. (HP Elite X2) G1, 5. Bluetooth + Send, 6. Wireless + Send, 7. Architecture, 8. (Intel) NUC8i3BEH, 9. StartUp, and 10. Shutdown.

Additionally, that list is graphically represented, as shown in Fig. 3, indicating changes in the order of effect. Noteworthy features, coloured in bold red, are meant to be avoided to improve the QA values — i.e., to decrease the energy consumption rate in our study. Hence, the developer is implicitly advised to replace those with whatever possible alternative, which, just for simplicity, are marked as dotted green features in Fig. 3. The complete model is presented in Fig. 2.

Nevertheless, SAVRUS is an interactive approach that developers will normally execute several times while taking decisions based on previous executions. Thus, we now present a SAVRUS usage scenario running a GEC analysis of a developer that needs to understand which and how features affect the energy consumption rate of her system. In this scenario, we leave aside the algorithms options, like the number of samples, as we properly evaluate them in the next section. First, the developer loads in her favourite web browser SAVRUS, which welcomes her with the graphical interface of Fig. 4. Then, she clicks the *Analyse* button without selecting features or declaring numerical values (i.e., cross-tree constraints). This results in receiving a similar graphical list to Fig. 3 — an overview of GEC features and energy interactions. While so much data can be overwhelming by itself,

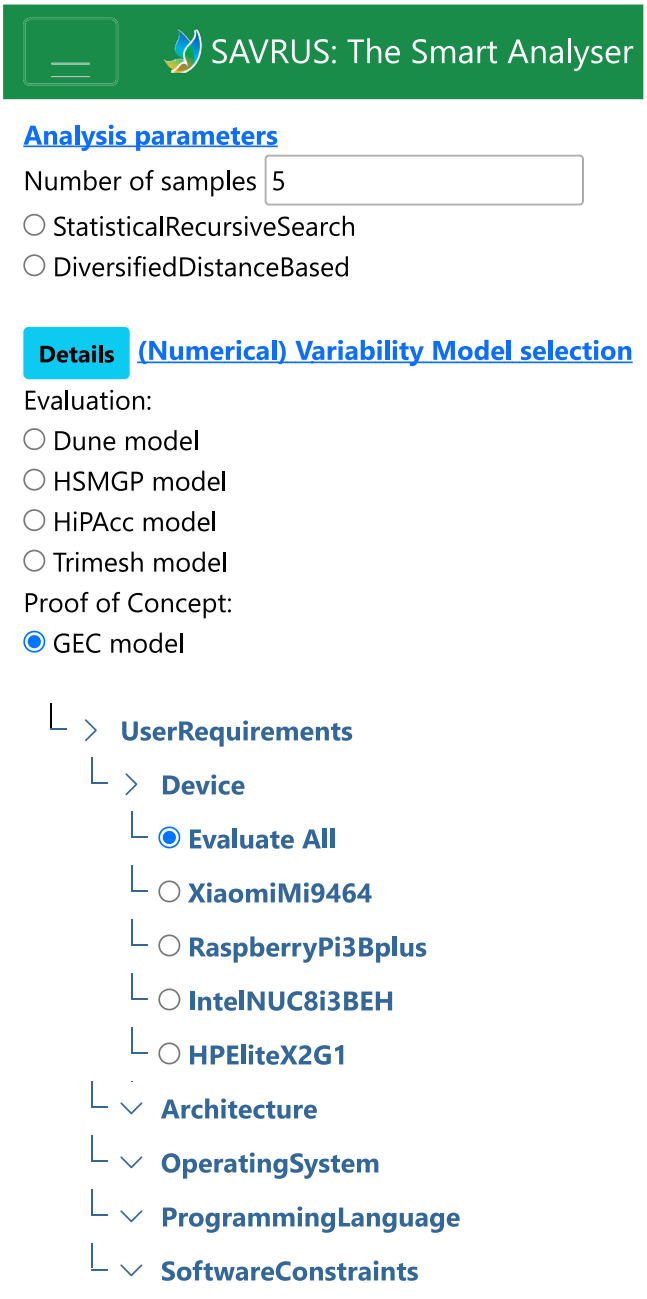


**Fig. 4.** The interactive graphical interface of the SAVRUS web prototype.

a quick look at the red indicators tells her that the *Edge Context* seems to be playing a significant role in the energy consumption of GEC, no matter the case. *Runs* is related to the number of times that a GEC operation is performed, while the *Phoronix Test Suite* features belong to benchmarking; hence both are useful to get insights into GEC but unsuited to be replaced. On the other hand, she understands that we should avoid constantly restarting a device, as *Shutdown* and *Startup* are energy consumers features, as well as *Send*ing data. Additionally, she can notice that the *measuring operation* was not considered noteworthy (i.e., did not negatively affect GEC energy requirements), which was essential to discard the observer's paradox.[14] After that, as she is interested

[14] Observer's paradox in our context: The quality measured tooling is unwittingly influenced by the quality requirements of the measuring tools.

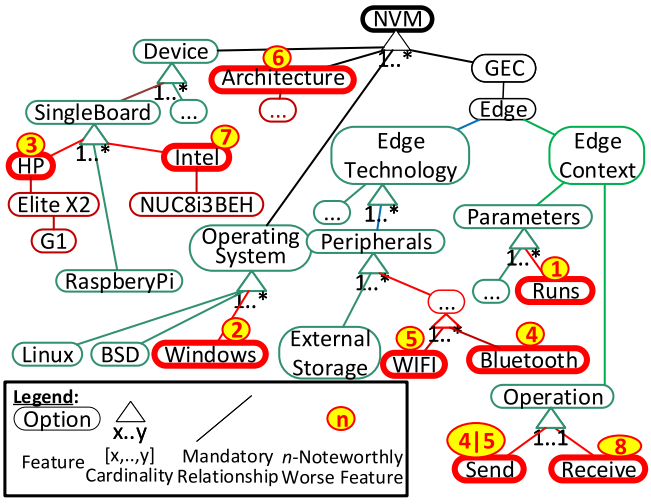


**Fig. 5.** SAVRUS results for GEC considering the features *Send* and *Receive* user requirements as variability model constraints.

in *Send* and *Receive* data with GEC, she constraints those operations in the second run of SAVRUS. The new results are visible in Fig. 5, where she sees a reduction in the number of noteworthy features detected, simplifying the application of the obtained insights. Based on that second run and the green-marked features, some actions that she can pursue to reduce the system energy requirement are: Switch the *Device* to a *RapsberryPi* and the *OS* to *Linux/BSD*; Use *External Storage* instead of a constant *Wireless* connection; Not worrying about most of the *peripherals*. By just switching the *Device* and *OS* to the green alternatives, she is saving between 5 and 27 watts, based on our measurements. That means a 16%–90% decrease in system's energy requirements of the system to perform the same task. On the other hand, based on the noteworthy red features, she should avoid using *Bluetooth* when wirelessly *Send*ing data, using *WiFi* instead. Just changing the *Wireless* interface of GEC to *WiFi*, she saved up to 1 watt in every possible scenario. Furthermore, she could keep performing newer runs of SAVRUS, hence obtaining deeper insights. As a final note, current mandatory features as *architecture* could be identified as noteworthy red features, yet irreplaceable in every scenario of the current model.

> ***Final results:*** The main features and interactions increasing the energy consumption rate in GEC, and aimed to change by an alternative, are *Windows OS*, larger devices such as *HP* and *Intel*, and last, *Send*ing *Wireless* data. Considering that we performed the analyses with just 3% samples of the 0.25% measurements of the total space, and we can reduce the energy consumption of GEC up to 90%, we conclude that SAVRUS is a functional approach that matches our objectives.

While the analysis results seem meaningful, they cannot be assumed as correct, as the SAVRUS approach has yet to be validated. To validate SAVRUS with GEC, we must measure its configuration space for the energy consumption rate ultimately, but that is impossible in a human time frame. Consequently, in the next section, we evaluate SAVRUS with different well-known and already studied VMs for a similar and completely measured QA.

## 7. Evaluation

In this section, we empirically demonstrate that SAVRUS generates meaningful results sufficiently fast by analysing two aspects of our prototype:

1. **SAVRUS validity.** *We analyse how reliable SAVRUS is concerning coverage and accuracy to identify feature influences based on incompletely measured spaces.*
2. **SAVRUS scalability.** *We analyse the relationship among the number of features, configurations and samples, and SAVRUS analysis runtime.*

We require VMs completely measured for a QA to demonstrate that SAVRUS results are correct empirically. As stated, it takes much time and resources to predict energy consumption — note that energy is always estimated, never measured. According to [24], large systems and VMs **completely** measured for any energy footprint are currently nonexistent and unlikely to happen. GEC is an example of this issue; thus we cannot use it to derive SAVRUS' accuracy.

As an alternative, we search for the most similar QA in which measuring is independent of complex expertise and tooling. According to Weber et al. [74], while the performance QAs **runtime** and energy do not necessarily correlate in every situation, they exhibit similar trends. Additionally, runtime performance is a well-known QA commonly used in the variability reasoning community and measured per configuration. According to that, we selected the four real-world VMs provided by authors of [75]: Dune [75], HSMGP [75], HiPAcc [75] and Trimesh [76]. Table 1 shows them ordered by the configuration space size in ascending order. To emulate the scenario in which SAVRUS is usually used, we purposely removed random chunks of measurements mimicking the issues of large VMs modelling energy-aware IoT systems (i.e., randomly spread measurements) like GEC. Then, we sequentially execute SAVRUS prototype 20 times. We repeat this process 5 times for each VM, degrading the measured space differently 5 times, reaching 100 analyses per VM.

The prototype runs on an Intel(R) Core i7-4790 CPU@3.60 GHz processor with 16 GB of memory RAM and an SSD running an up-to-date Ubuntu server 20.04 LTS X86_64 with the latest supported versions of NGINX web server, PHP 8.x, and MariaDB 10.x. $k$ parameter was again left by default ($k = 2$).

### 7.1. SAVRUS approach validity

For each analysis, we calculated and averaged the following metrics:

**Definition 1** (*Time*)**.** It is the averaged runtime in minutes ($m$) performed with our implementation of SAVRUS of $N$ analyses of a variability model $VM_x$ as:

$$\text{SAVRUS Time}(VM_x) = \frac{\sum_{i=1}^{N} Runtime_i}{N}$$

**Definition 2** (*Coverage*)**.** It is the averaged percentage of covered features $f$s and pairs of features $pf$s regarding the total number of them when running SAVRUS for $N$ analyses of a variability model $VM_x$:

$$\text{SAVRUS Coverage}(VM_x) = \left(\frac{\sum_{i=1}^{N} \#f_i + \#pf_i}{\#f_x + \#pf_x}\right) * 100$$

**Definition 3** (*Accuracy*)**.** After initially computing the correctly ordered list by a brute-force pairwise, we calculate the accuracy of SAVRUS for $N$ analyses of a variability model $VM_x$ an averaged percentage of correctly identified and ranked noteworthy features $rnf$s up to pairwise interactions. In other words, the accuracy is the probability of SAVRUS detecting and perfectly ordering the most meaningful feature-quality interactions.

$$\text{SAVRUS Accuracy}(VM_x) = \left(\frac{\sum_{i=1}^{N} \#rnf_i}{\#rnf_x}\right) * 100$$

**Table 2**
SAVRUS time, coverage and accuracy for SRS/DDbS querying 3% samples of the incompletely measured space of four real-world systems.

| Sampling | SRS | | | DDbS | | |
|---|---|---|---|---|---|---|
| *Model* | | | | | | |
| | Time | Coverage | Accuracy | Time | Coverage | Accuracy |
| *Dune* | 1.5 m | 73.2% | 86.7% | 1.3 m | 76.9% | 83.9% |
| *HSMGP* | 1.9 m | 72.4% | 85.4% | 1.4 m | 75.7% | 83.5% |
| *HiPAcc* | 3.7 m | 70.5% | 84.1% | 2.8 m | 73% | 83.3% |
| *Trimesh* | 7.3 m | 65.2% | 83% | 6.6 m | 68.7% | 82.8% |
| **Mean:** | **3.6 m** | **70.3%** | **84.8%** | **3 m** | **73.6%** | **83.4%** |

For results fairness and to keep runtime below 10 min in all cases, we fixed the number of samples per analysis to 3% of their respective search space. Nevertheless, a user could increase the number of samples to obtain better results at the cost of SAVRUS runtime. However, this is most noticeable for initial space explorations; if similar analyses have been previously executed, a heavy increment of the number of samples produces a large server overhead without proportionally improving its accuracy. Finally, to prevent SAVRUS server blocking, its current time-out is set to 30 min.

The averaged results of the $20 * 5$ executions per VM are presented in Table 2. We can see that SAVRUS runtime increases linearly and proportionally to the space size, reaching ∼7 min for Trimesh. The opposite occurs with coverage and accuracy, as they slightly decrease, although they keep around [65.2, 76.9]% and [83, 82.8]%, respectively. Hence, SAVRUS returned a correct ranking of noteworthy features and interactions affecting a QA above 80% of the runs; in other words, SAVRUS returned a partially mistaken ordered list less than 20% of the times. Those could be sufficiently good results for our purpose, i.e., to give energy footprint advice, considering we have a degraded search space and a relatively small sample set, domain unaware.

Regarding sampling, their means suggest that DDbS is slightly faster (i.e., −0.6 min) and has a higher coverage (i.e., +3.3%) than SRS, although SRS is more accurate (i.e., +1.4%) on average. It is interesting to highlight that while SRS has lower coverage than DDbS, it is more accurate; hence the covered SRS samples seem more meaningful than the ones covered by DDbS but at the cost of a higher runtime. That is also visible by looking at each model metric, where the differences are more stressed when the search space is smaller. DDbS accuracy generally seems more stable than SRS, which decreases faster for larger models. Consequently, the larger the search space, the more sense makes DDbS. Nevertheless, SRS is superior for all tested models if runtime is not considered.

> ***Validity Results:*** SAVRUS generates a correct ranking of noteworthy features and interactions 80% of the time for every incompletely measured model. This accuracy is inversely proportional to the space size and the number of samples. SRS is generally more accurate than DDbS, even with lower coverage. On the other hand, DDbS is comparatively faster, especially for larger spaces. Hence, the results are reasonably accurate without exploiting domain knowledge, matching O1/2/4.

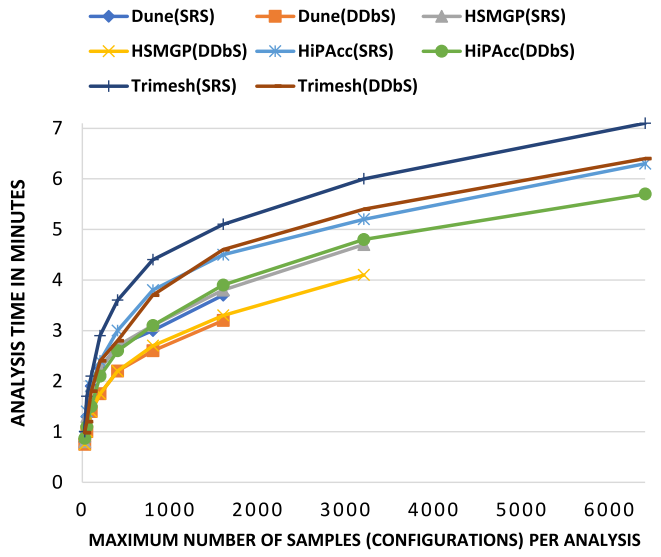


**Fig. 6.** SAVRUS scalability graph with regard to #samples.

### 7.2. SAVRUS prototype scalability

To test the scalability of SAVRUS implementation, we reused the same models used in the previous section. In this case, we degraded these models with an increasing number of samples from 25 to 6400, obtaining 9 sample set sizes, where each one doubles the former one. We have followed the same procedure of $5 * 20$ runs per case study (i.e., dotted marks), and the average of the results is shown in Fig. 6. We have tested 8 scenarios (i.e., 4 systems * 2 sampling strategies) for 9 different sample set sizes, but two of the VMs (Dune and HSMGP) were smaller than the larger sample sizes (3200 and 6400). So, the graph shows a total of 66 cases. The *X*-axis is the maximum number of samples per analysis, and the *Y*-axis is the execution time of the analysis.

The graph shows that, while there is a minimum workload of ∼1 min, the runtime of a SAVRUS analysis linearly increases proportionally to the number of samples. However, the curves tend to be smooth for larger sample sets. Additionally, the separation between the curves suggests that the execution time also increases based on the complexity and space size of the model. Like the characteristics described in Table 1, the relationship appears linear again. Finally, and matching RQ2 results, DDbS is slightly faster than SRS. Those differences tend to increase for larger sample sets and more complex spaces.

Regarding numbers, the execution time was kept below 7 min for the worst-case scenario: 6400 samples of the largest and most complex system — Trismesh. However, for standard sample sets, ∼2% of the total space, the execution time is always below 3 min for all cases and scenarios. For simplicity, the graph values refer to the full execution time of SAVRUS, but we have also measured its internals. We have identified that the most time-consuming components of SAVRUS are in the relevant order:

1. The database queries and related functionality, such as updating TL scoring weights or requesting *k* neighbours.
2. Solver-based functions. Clafer, as a Java application, has a significant initial workload, even for small sample sets.
3. Lastly, machine learning PHP8 libraries. PHP is not a mathematical-specific language, as it is meant for backends.

***Scalability Results:*** SAVRUS implementation presents an initial analysis runtime of ∼1 min, taking less than 3 min for comprehensible cases and scenarios, and with a worst-case of ∼7 min. We have empirically demonstrated that it has a linear relationship with the number and complexity of the samples, getting smoother based on how large the sample set is. Finally, the main bottlenecks are the database and the solver. Hence, linear scalability with reasonable base analysis times accomplishes O3.

## 8. Discussion and threats to validity

First, we tested SAVRUS with the wide configuration space of GEC, which is a system of our expertise, and where reducing energy consumption is particularly critical in the current times. In the process, we found interesting features interactions while highlighting and ranking by their effect strength, the ones notably affecting the QA energy consumption rate. This provided several energy-aware optimisation insights based on excluding the energy-hungry features while suggesting sustainable alternatives.

Then, we evaluated and compared the two implementations of our approach, DDbS and SRS. To answer our RQs, we validated SAVRUS effectiveness regarding (i) the quality of the features insights, (ii) the size of the sample sets concerning the space size, and (iii) the analyses times; in other words, we accomplished objectives 1–4 that motivated our research in Section 3.

SAVRUS generated a correct ranking of noteworthy features and interactions 80% of the time for every incompletely QA-measured model. Regarding SAVRUS performance, the current prototype has a base runtime of ∼1 min, taking less than 3 min for comprehensible cases and scenarios.

Between the two sampling implementations, and considering Table 2 and Fig. 6, DDbS was the most balanced alternative due to its speed, accuracy, and scalability, especially for large and complex systems. SRS is the most accurate alternative if the analysis time is not an issue for the developer.

In summary, with SAVRUS, we can find the quality influences of features and interactions in a system by analysing its unknown configuration space. While some techniques in SAVRUS have already been used in other works, they are adapted and combined in a novel sequence, as their traditional usage in feature-level or learning-based solutions will completely break our objectives. In other words, current works only considered quality analyses at the configuration level with (prior) domain knowledge. Furthermore, SAVRUS metrics indicate sufficient accuracy, scalability and adaptability to analyst requirements.

Hence, our contributions are summarised as identifying the challenges of large emergent systems, a reasoned selection of techniques, the modular SAVRUS approach, and the empirical evidence obtained by running a SAVRUS implementation with empirical data. With SAVRUS, developers can now get fast insights into features and quality interactions of large yet partially measured models at the configuration level without requiring specific domain knowledge, supporting features and quality constraints, and sampling method selection and adjustment. With those insights, developers optimise the systems by directly replacing the noteworthy red features and shielding the noteworthy green ones.

We evaluated our SAVRUS strategy with experiments that provide completely measured models to avoid conceptual errors. Regarding their **construct validity**, we analyse whether we use the proper data set. To represent an incomplete measures space of real-world Edge/IoT/Cloud systems, we properly degraded the completely measured spaces according to the objectives of Section 3. The degradation procedure is automatic and random and

was independently applied to the original spaces a few times. Consequently, we analysed the same space many times but degraded it differently to minimise the collateral effects that the degradation procedure could have on the results.

The **internal validity** is determined by the accuracy of SAVRUS results. Our specific implementation of SRS and DDbS methods could introduce some bias to the results. We aimed for an equal balance between the fast seed randomisation and the computationally expensive solver calls. We know that the selected sampling and learning methods may not be the best choice for all QAs and systems, but we are confident that they are proper for energy consumption and edge systems. Nevertheless, we carefully implemented the state-of-the-art while using open-source tools and libraries, which are widely tested in the referenced repositories. However, we did not only validate SAVRUS outputs but individually reviewed each strategy component (Fig. 1) to avoid hidden errors. In all cases, we repeated the analyses and presented averaged metrics to reduce a possible bias. Additionally, we remind that SAVRUS comprises a normality test within the process (i.e., MWU) with a 95% confidence, so the **conclusion validity** is reasonably good.

To evaluate SAVRUS **external validity**, we selected 4 VMs well-known by the community with a different number of features and space sizes. As highly configurable systems completely measured for energy consumption are uncommon, we searched for the well-known measured systems of Table 1, which share some issues in the scenarios of edge systems consuming energy. As those models are additionally degraded to mimic the exact issues of GEC, we are not entirely sure that the results of our evaluations can be generalised to all VMs and QAs. However, we are confident enough that SAVRUS could be extended for those domains in which it was partially inaccurate. Nevertheless, domains with disparate and narrow normal distributions could be challenging for SAVRUS to detect noteworthy features with sufficient confidence. A possible workaround could be to adjust (i.e., reduce) the confidence level in those scenarios.

Another issue is the level of knowledge and optimisation that a company could obtain using the SAVRUS approach. For example, if they are managing small VMs or an almost unmeasured space, we can expect an improvement under 90%, as the level of QA variability is likely small. However, by using SAVRUS, those companies could also benefit from other exotic QAs like sound, pollution, temperature, runtime, or airflow, among others. This is especially true if they share the same objectives described in Section 3, as other tools were not designed with them in mind. Consequently, SAVRUS coverage and accuracy are expected to be superior for these configuration-level QAs. Finally, reasoning approaches for feature-level QAs could rival SAVRUS in this kind of QAs. Notice that although SAVRUS covers both feature-level and configuration-level QAs, we designed SAVRUS to improve the analysis of configuration-level QAs.

While the prototype is online, we provide the 5 VMs of Table 1 in Clafer (.txt) format, their respective QAs measurements as CSVs, and the SAVRUS analyses and readings as a .xlsx all compressed in a *.zip* file data set downloadable in:

https://doi.org/10.5281/zenodo.6251045

**Limitations**. As with most of the related work, SAVRUS results are influenced by the quality of the input data. Wrongly constructed VMs, poorly measured QAs, and a biased data distribution certainly decrease SAVRUS accuracy. This could happen if a SAVRUS user does not choose adequate parameters for the desired analysis. An example could be a user indicating a maximum request of only 5 samples to analyse an large space; naturally, the results are expected to be inaccurate due to an insufficient amount of data analysed. Also, if a feature or an interaction was never measured in a configuration, specific insights are purely based on how their $k$-close neighbours perform. At the same time, if those neighbours are unmeasured, those missing features are discarded for analysis. Additionally, if the measured space size predominantly grows, the probability of needing the kNN step is proportionally reduced. While unlikely in practice, SAVRUS could theoretically suffer from the *Curse of Dimensionality*. Variability models in the Clafer language are defined as variability trees; thereupon, the full dimension that SAVRUS analysis could have is somewhere between the maximum depth of the tree plus the number of grouped features if mandatory or disjunction cardinalities are defined. As a worst-case scenario, we could calculate the dimensionality of Linux 2.6.33.3, one of the colossal systems with a configuration space of $\sim 3.90 \times 10^{1672}$. Its maximum dimensionality is $\sim$192 [77]. However, that maximum dimensionality is unlikely to arise in every analysis as the average depth of the branches is 2.7, and the total number of features is 6467. Finally, we need to remind the readers that SAVRUS accepts feature constraints from the user, exponentially reducing the configuration space size and, consequently, the probability of larger dimensionalities. In parallel, we can further discard this curse from the TL with the LOF step, as the measured configuration size is reduced further to the configurations comprising a specific feature, or set of interacting features, detected as noteworthy.

On the other hand, TL could suffer from another issue, which is the *Negative Transfer*. The definition of negative transfer is still under debate and is based on the algorithms, domain divergence and target data of the specific study [78]. Based on that, negative transfer in SAVRUS could arise if an extended sequence of past executions were based on a cluster of the configuration search space that behaves very differently from the ones in the current analyses. This would cause the resulting rankings to be shifted due to inaccurate scoring weights. However, it does not affect the detection of noteworthy features and interactions but their order (i.e., likelihood) in the ranking. Additionally, its probability is low, as that situation requires analysing one cluster many times and then analysing a cluster that performs very differently only once. A new sequence of analyses will properly readjust the scoring weights. This behaviour is expected from any TL approach in the early stages of learning a new cluster, and with experience, they correct for these effects.

While unexpected results could arise for colossal VMs and rare QAs, we are confident that our preliminary results will hold. Fortunately, current tests show linear trends for different cases and sample sets. From a developer's perspective, waiting a few minutes for analysis could be undesirable, but a more powerful computer and fewer samples certainly alleviate this issue. Finally, SAVRUS might identify all the alternatives of a parent feature as noteworthy affecting a QA. In other words, all the child features are equally affecting a QA. An example of this is the *Architecture* parent feature in the proof of concept of Section 6, where its two children (*x86* and *x64*) are notably increasing the energy consumption rate of any configuration whenever they are present. While it would be more interesting to directly suggest replacing a feature with a better-performing alternative, only concluding that all the other options affect equally is still a valuable piece of information. Nonetheless, modelling and quality measuring new features for SAVRUS will reduce this limitation in the long run.

## 9. Related work

This section completes the related work already discussed in Sections 2 and 4. While those sections introduced the techniques required to understand the motivation and contributions of our approach, this section discusses specific strategies dealing with the different issues considered in our approach.

Literature dealing with finding interactions without domain knowledge is rare, especially in energy-aware edge computing. As

it is critical, most works aim to reduce costs while maintaining performance. However, they share only some of our objectives simultaneously. [79] is a meta-review of predicting the energy consumption for software reuse by integrating data mining and artificial intelligence. Unfortunately, the authors conclude that all the reviewed solutions need more efficient prediction. MIGRATE is a three-step machine learning framework for intelligent energy profiling [80]. The two first steps of MIGRATE are incompatible with our domain unknown requirement: big data analysis, then supervised learning, ending with unsupervised learning. MoMo performs analyses dynamically (dynamic variability) [81]. However, the domain is well known beforehand and works with presumed absolute values, which breaks O2 and O1. Federated learning generates performance models based on aggregation learning providing a sophisticated balance between the accuracy and financial cost of wireless sensors [82], but this breaks all our objectives. With the same aim of optimising accuracy and economic cost, [83] uses orthogonal VMs with quality values at the feature level and a series of mappings and transformations compatible with the constraint satisfaction problem FAMA framework. Again, this solution breaks all of our objectives, as it works with absolute values and domain knowledge at the feature level. The mappings and analyses are neither reusable nor extendable for evolved models or new user requirements. The concept of deep software variability is used in [84] to exploit product lines with multiple concurrent layers. Instead of different layers, SAVRUS modelled them as separate branches. *Thermal-aware Scheduling and Tuning* (TaSat) propose a Pareto algorithm for performance, energy and temperature, which does not require specific measurements to perform accurately; however, its domain is specifically heterogeneous embedded devices [85]. A similar work is TOFFEE which uses a stochastic algorithm [86], but both solutions consider the quality space as well-known, breaking O2, O3 and O4.

Another approach proposes pre-defined templates based on the software architecture for energy-aware edge FPGAs [87]. These are regularly called energy models in the literature and tend to be used to detect worst-case scenarios like the tool Serapis [88], but they do not match our objectives. GreenScaler is a tool that contains automatic test generation for cyber–physical systems, and the results are stored in a local repository and used then to detect energy-aware configurations [14]. Finally, HADAS [89] tool generates tendency graphs of energy consumption for cyber–physical systems. The last two were the most promising for our domain. However, they partially break O1, and completely O2. As we can see, O2 is the most critical counter-objective of the current solutions, as far from considering it, recent works even exploited it.

In summary, **the main differences between the related work and our proposal are**: (1) Other proposals derive estimation models of feature level quality values for the generation of nearly-optimal configurations. Opposite to that, we provide an interactive approach to analyse QAs influences on QAs at the configuration level. Thus, we provide means to software engineers to understand the configuration space, which differs from predicting absolute values or exploiting the much smaller feature space; (2) Their usage of current learning methods requires substantial domain knowledge. However, our usage of sampling, statistical *k*NN and TL do not assume the entire QA space as granted; (3) Accurate estimation models are not directly reusable for highly variable search spaces, but this is mandatory for us considering the energy efficiency of IoT/Edge/Cloud systems; and (4) Software developers tend to use simple tools with fast insights for their feature selections, which usually would take days if new estimation models of large spaces need to be generated for every new feature, constraint or requirement.

## 10. Conclusions and future work

This paper presents the difficulties of analysing Edge Computing/Cloud/B5G systems to improve products regarding complex QA-like energy consumption. We presented the SAVRUS approach that: navigates through large and hard-to-measure spaces with domain-unknown QAs; works with just available data; and provides fast identification of features and pairwise interactions ranked by their importance. The SAVRUS strategy comprises 5 sequential steps: solver-based sampling, *k*NN, statistical tests, TL, and a weighted sorting of noteworthy features and interactions affecting a QA. We developed a prototype with SRS and DDbS for sampling, a statistical MWU test, and TL based on shifted LOF values. We tested SAVRUS applicability by analysing GEC (a vast and partially measured Edge Computing/Cloud/B5G system) to improve its energy consumption rate. Finally, we empirically demonstrated its linear scalability and sufficient accuracy by a series of analyses with 4 real-world highly configurable systems: Dune, HSMGP, HiPAcc and Trimesh.

For future work, firstly, we can study the possibility of reducing the *Curse of Dimensionality* by replacing in step 4 *k*NN with *Approximate Nearest Neighbours* (ANN) techniques like *Locality-Sensitive Hashing* [90]. Additionally, to increase SAVRUS scalability regarding the measured space size, we can include another sampling step (i.e., 3.2) to use a measured sample set to feed kNN and ANN to try to predict the QA values of the unmeasured sample set. In parallel, we can build in step 7 a scoring function by defining its aggregation function (e.g., the weighted sum of scores); we could rank the configurations by highly affecting a QA and use it to detect some of the effects of negative transfers. Further, we are studying how to extend the analysis of the T-wise interactions in step 3 (i.e. T-wise) in a scalable manner (e.g., self-guided sampling [91]). We could study the advantages and inconveniences of heuristically calculating the number of top configurations in step 5 with software threshold techniques [92]. We are also researching the advantages of integrating unsupervised machine learning techniques in SAVRUS' step 6. Last but not least, we plan to support multi-objective analyses (i.e., Pareto and trade-offs), like energy footprint and latency, in step 8. In any case, we welcome larger systems with different QAs to test them in SAVRUS and connect the prototype with other tools (e.g., repositories and micro-services analysis).

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

I have shared my data in https://doi.org/10.5281/zenodo.6251045 , and the tool is deployed online at https://hadas.caosd.lcc.uma.es/savrus.php.

## Acknowledgements

This work is supported by the European Union's H2020 research and innovation programme under grant agreement *DAEMON* H2020-101017109, by the projects *IRIS* PID2021-122812OB-I00 (co-financed by FEDER funds), *Rhea* P18-FR-1081 (MCI/AEI/ FEDER, UE), and *LEIA* UMA18-FEDERIA-157, and the PRE2019-087496 grant from the Ministerio de Ciencia e Innovación, Spain.

## References


[1] D.-J. Munoz, J. Oh, M. Pinto, L. Fuentes, D. Batory, Uniform random sampling product configurations of feature models that have numerical features, in: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A, ACM, Paris, France, 2019, pp. 289–301.

[2] T. Berger, S. She, R. Lotufo, A. Wasowski, K. Czarnecki, A study of variability models and languages in the systems software domain, IEEE Trans. Softw. Eng. 39 (12) (2013) 1611–1640, http://dx.doi.org/10.1109/TSE.2013.34.

[3] S. Robertson, J. Robertson, Mastering the Requirements Process: Getting Requirements Right, Addison-wesley, Boston, USA, 2012.

[4] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, D.O. Wu, Edge computing in industrial internet of things: Architecture, advances and challenges, IEEE Commun. Surv. Tutor. 22 (4) (2020) 2462–2488.

[5] K. Georgiou, S. Xavier-de-Souza, K. Eder, The IoT energy challenge: A software perspective, IEEE Embedded Syst. Lett. 10 (3) (2018) 53–56, http://dx.doi.org/10.1109/LES.2017.2741419.

[6] A. Mordahl, J. Oh, U. Koc, S. Wei, P. Gazzillo, An empirical study of real-world variability bugs detected by variability-oblivious tools, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, IEEE, New York, USA, 2019, pp. 50–61.

[7] L. Zhou, J. Li, F. Li, Q. Meng, J. Li, X. Xu, Energy consumption model and energy efficiency of machine tools: a comprehensive literature review, J. Clean. Prod. 112 (2016) 3721–3734, http://dx.doi.org/10.1016/j.jclepro.2015.05.093, URL https://www.sciencedirect.com/science/article/pii/S0959652615006617.

[8] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: O. Pastor, J.a. Falcão e Cunha (Eds.), Advanced Information Systems Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 491–503.

[9] A. Cañete, M. Amor, L. Fuentes, Energy-efficient deployment of IoT applications in edge-based infrastructures: A software product line approach, IEEE Internet Things J. 1 (1) (2020) 1, http://dx.doi.org/10.1109/JIOT.2020.3030197.

[10] D.-J. Munoz, D. Gurov, M. Pinto, L. Fuentes, Category theory framework for variability models with non-functional requirements, in: M. La Rosa, S. Sadiq, E. Teniente (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2021, pp. 397–413.

[11] T. Pett, T. Thüm, T. Runge, S. Krieter, M. Lochau, I. Schaefer, Product sampling for product lines: The scalability challenge, in: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume a, Association for Computing Machinery, Paris, France, 2019, pp. 78–83.

[12] S. Fischer, L. Linsbauer, A. Egyed, R.E. Lopez-Herrejon, Predicting higher order structural feature interactions in variable systems, in: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, CA, USA, 2018, pp. 252–263, http://dx.doi.org/10.1109/ICSME.2018.00035.

[13] E. Alpaydin, Introduction to Machine Learning, MIT Press, Massachusetts, USA, 2020.

[14] S. Chowdhury, S. Borle, S. Romansky, A. Hindle, GreenScaler: Training software energy models with automatic test generation, Empir. Softw. Eng. 24 (4) (2019) 1649–1692, http://dx.doi.org/10.1007/s10664-018-9640-7.

[15] R. Heinrich, A. Koziolek, S. Becker, R. Reussner, Infrastructure for modeling and analyzing the quality of software architectures, in: 2019 IEEE/ACM 2nd International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE, IEEE, Montreal, Canada, 2019, pp. 2–5, http://dx.doi.org/10.1109/ECASE.2019.00009.

[16] D.-J. Munoz, M. Pinto, L. Fuentes, HADAS: Analysing quality attributes of software configurations, in: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B, SPLC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 13–16, http://dx.doi.org/10.1145/3307630.3342385.

[17] C. Kaltenecker, A. Grebhahn, N. Siegmund, S. Apel, The interplay of sampling and machine learning for software performance prediction, IEEE Softw. 37 (4) (2020) 58–66, http://dx.doi.org/10.1109/MS.2020.2987024.

[18] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[19] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, A. Wasowski, Clafer: unifying class and feature modeling, Softw. Syst. Model. 15 (3) (2016) 811–845.

[20] M. Glinz, On non-functional requirements, in: 15th IEEE International Requirements Engineering Conference (RE 2007), IEEE, Delhi, India, 2007, pp. 21–26, http://dx.doi.org/10.1109/RE.2007.45.

[21] R. Olaechea, S. Stewart, K. Czarnecki, D. Rayside, Modelling and multi-objective optimization of quality attributes in variability-rich software, in: Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages, ACM, Denver, USA, 2012, pp. 1–6.

[22] N. Siegmund, S. Sobernig, S. Apel, Attributed variability models: outside the comfort zone, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, Paderborn, Germany, 2017, pp. 268–278.

[23] G. Pinto, F. Castor, Energy efficiency: A new concern for application software developers, Commun. ACM 60 (12) (2017) 68–75, http://dx.doi.org/10.1145/3154384.

[24] E. García-Martín, C.F. Rodrigues, G. Riley, H. Grahn, Estimation of energy consumption in machine learning, J. Parallel Distrib. Comput. 134 (2019) 75–88, http://dx.doi.org/10.1016/j.jpdc.2019.07.007, URL https://www.sciencedirect.com/science/article/pii/S0743731518308773.

[25] L. Olsina, G. Lafuente, O. Pastor, Towards a reusable repository for web metrics, J. Web Eng. 1 (1) (2002) 61–73.

[26] L. Brahimi, L. Bellatreche, Y. Ouhammou, A recommender system for DBMS selection based on a test data repository, in: J. Pokorný, M. Ivanović, B. Thalheim, P. Šaloun (Eds.), Advances in Databases and Information Systems, Springer International Publishing, Cham, 2016, pp. 166–180.

[27] P. Mohagheghi, R. Conradi, Exploring industrial data repositories: where software development approaches meet, in: Proceedings of the 8th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'04), IEEE, Oslo, Norway, 2004, pp. 61–77.

[28] S. Apel, S. Kolesnikov, N. Siegmund, C. Kästner, B. Garvin, Exploring feature interactions in the wild: The new feature-interaction challenge, in: Proceedings of the 5th International Workshop on Feature-Oriented Software Development, FOSD '13, Association for Computing Machinery, New York, NY, USA, 2013, pp. 1–8, http://dx.doi.org/10.1145/2528265.2528267.

[29] M. Happ, A.C. Bathke, E. Brunner, Optimal sample size planning for the wilcoxon-mann-whitney test, Stat. Med. 38 (3) (2019) 363–375, http://dx.doi.org/10.1002/sim.7983, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.7983, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.7983.

[30] G.D. Ruxton, The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test, Behav. Ecol. 17 (4) (2006) 688–690, http://dx.doi.org/10.1093/beheco/ark016, arXiv:https://academic.oup.com/beheco/article-pdf/17/4/688/17275561/ark016.pdf.

[31] M. Varshosaz, M. Al-Hajjaji, T. Thüm, T. Runge, M.R. Mousavi, I. Schaefer, A classification of product sampling for software product lines, in: Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1, ACM, Gothenburg Sweden, 2018, pp. 1–13.

[32] I. Etikan, K. Bala, Sampling and sampling methods, Biom. Biostat. Int. J. 5 (6) (2017) 00149.

[33] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, S. Apel, A comparison of 10 sampling algorithms for configurable systems, in: 2016 IEEE/ACM 38th International Conference on Software Engineering, ICSE, IEEE, Austin, USA, 2016, pp. 643–654.

[34] M. Feng, E. Peck, L. Harrison, Patterns and pace: Quantifying diverse exploration behavior with visualizations on the web, IEEE Trans. Vis. Comput. Graphics 25 (1) (2019) 501–511, http://dx.doi.org/10.1109/TVCG.2018.2865117.

[35] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.

[36] S. Baltes, P. Ralph, Sampling in software engineering research: A critical review and guidelines, 2020, CoRR arXiv:2002.07764, URL https://arxiv.org/abs/2002.07764.

[37] J. Oh, D. Batory, M. Myers, N. Siegmund, Finding near-optimal configurations in product lines by random sampling, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, IEEE, Paderborn, Germany, 2017, pp. 61–71.

[38] J. Oh, P. Gazzillo, D. Batory, T-wise coverage by uniform sampling, in: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A, Association for Computing Machinery, Paris, France, 2019, pp. 84–87.

[39] S. Vilkomir, O. Starov, R. Bhambroo, Evaluation of t-wise approach for testing logical expressions in software, in: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, IEEE, Luxembourg, Luxembourg, 2013, pp. 249–256.

[40] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, S. Apel, Distance-based sampling of software configuration spaces, in: 2019 IEEE/ACM 41st International Conference on Software Engineering, ICSE, IEEE, Montreal, Canada, 2019, pp. 1084–1094.

[41] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, A. Wasowski, Variability-aware performance prediction: A statistical learning approach, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, Silicon Valley, USA, 2013, pp. 301–311, http://dx.doi.org/10.1109/ASE.2013.6693089.

[42] J. Guo, J.H. Liang, K. Shi, D. Yang, J. Zhang, K. Czarnecki, V. Ganesh, H. Yu, SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines, Softw. Syst. Model. 18 (2) (2019) 1447–1466.

[43] J. Alves Pereira, M. Acher, H. Martin, J.-M. Jézéquel, Sampling effect on performance prediction of configurable systems: A case study, in: Proceedings of the ACM/SPEC International Conference on Performance Engineering, ICPE '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 277–288, http://dx.doi.org/10.1145/3358960.3379137.

[44] A. Sarkar, J. Guo, N. Siegmund, S. Apel, K. Czarnecki, Cost-efficient sampling for performance prediction of configurable systems (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, Lincoln, USA, 2015, pp. 342–352.

[45] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, IEEE, Lake Buena Vista, USA, 2018, pp. 71–82.

[46] J. Xuan, Y. Gu, Z. Ren, X. Jia, Q. Fan, Genetic configuration sampling: learning a sampling strategy for fault detection of configurable systems, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, Kyoto, Japan, 2018, pp. 1624–1631.

[47] S. Kolesnikov, N. Siegmund, C. Kästner, A. Grebhahn, S. Apel, Tradeoffs in modeling performance of highly configurable software systems, Softw. Syst. Model. 18 (3) (2019) 2265–2283.

[48] F. Ferreira, G. Vale, J.P. Diniz, E. Figueiredo, Evaluating T-wise testing strategies in a community-wide dataset of configurable software systems, J. Syst. Softw. 179 (2021) 110990, http://dx.doi.org/10.1016/j.jss.2021.110990, URL https://www.sciencedirect.com/science/article/pii/S016412122100087X.

[49] S. Krieter, Large-scale T-wise interaction sampling using YASA, in: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A, SPLC '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–4, http://dx.doi.org/10.1145/3382025.3414989.

[50] S. Khoshmanesh, R. R. Lutz, The role of similarity in detecting feature interaction in software product lines, in: 2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, IEEE, Montreal, Canada, 2018, pp. 286–292, http://dx.doi.org/10.1109/ISSREW.2018.00020.

[51] J. Sanchez, A review of pair-wise testing, (1) 2016, pp. 1–7, arXiv preprint arXiv:1606.00288.

[52] N.S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, Amer. Statist. 46 (3) (1992) 175–185, URL http://www.jstor.org/stable/2685209.

[53] S. Zhang, X. Li, M. Zong, X. Zhu, R. Wang, Efficient kNN classification with different numbers of nearest neighbors, IEEE Trans. Neural Netw. Learn. Syst. 29 (5) (2018) 1774–1785, http://dx.doi.org/10.1109/TNNLS.2017.2673241.

[54] Q. Zou, L. Lu, S. Qiu, X. Gu, Z. Cai, Correlation feature and instance weights transfer learning for cross project software defect prediction, IET Softw. 15 (1) (2021) 55–74, http://dx.doi.org/10.1049/sfw2.12012, arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/sfw2.12012, URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/sfw2.12012.

[55] M.A. Soliman, I.F. Ilyas, D. Martinenghi, M. Tagliasacchi, Ranking with uncertain scoring functions: Semantics and sensitivity measures, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 805–816, http://dx.doi.org/10.1145/1989323.1989408.

[56] Y. Ning, H. Liu, A general theory of hypothesis tests and confidence regions for sparse high dimensional models, Ann. Statist. 45 (1) (2017) 158–195, http://dx.doi.org/10.1214/16-AOS1448.

[57] L. Ochoa, O. González-Rojas, A.P. Juliana, H. Castro, G. Saake, A systematic literature review on the semi-automatic configuration of extended product lines, J. Syst. Softw. 144 (2018) 511–532, http://dx.doi.org/10.1016/j.jss.2018.07.054, URL https://www.sciencedirect.com/science/article/pii/S0164121218301511.

[58] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, H. Yu, Data-efficient performance learning for configurable systems, Empir. Softw. Eng. 23 (3) (2018) 1826–1867.

[59] R. Krishna, V. Nair, P. Jamshidi, T. Menzies, Whence to learn? Transferring knowledge in configurable systems using BEETLE, IEEE Trans. Softw. Eng. 1 (2020) 1, http://dx.doi.org/10.1109/TSE.2020.2983927.

[60] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, Y. Agarwal, Transfer learning for performance modeling of configurable systems: An exploratory analysis, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, Urbana-Champaign, USA, 2017, pp. 497–508.

[61] J. Dorn, S. Apel, N. Siegmund, Generating attributed variability models for transfer learning, in: Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, VAMOS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–8, http://dx.doi.org/10.1145/3377024.3377040.

[62] Y. Chen, Y. Gu, L. He, J. Xuan, Regression models for performance ranking of configurable systems: A comparative study, in: H. Miao, C. Tian, S. Liu, Z. Duan (Eds.), Structured Object-Oriented Formal Language and Method, Springer International Publishing, Cham, 2020, pp. 243–258.

[63] W. Fu, T. Menzies, Easy over hard: A case study on deep learning, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2017, Association for Computing Machinery, New York, NY, USA, 2017, pp. 49–60, http://dx.doi.org/10.1145/3106237.3106256.

[64] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Comput. Electr. Eng. 40 (1) (2014) 16–28, http://dx.doi.org/10.1016/j.compeleceng.2013.11.024, URL https://www.sciencedirect.com/science/article/pii/S0045790613003066, 40th-year commemorative issue.

[65] T. Khoshgoftaar, D. Dittman, R. Wald, A. Fazelpour, First order statistics based feature selection: A diverse and powerful family of feature selection techniques, in: 2012 11th International Conference on Machine Learning and Applications, Vol. 2, IEEE, CA, USA, 2012, pp. 151–157, http://dx.doi.org/10.1109/ICMLA.2012.192.

[66] J. Oh, D. Batory, M. Myers, N. Siegmund, Finding product line configurations with high performance by random sampling, in: UT in Austin Open Proceedings, Semantic Scholars, Austin, TX, USA, 2017, pp. 0–13.

[67] G. Baldini, D. Geneiatakis, A performance evaluation on distance measures in KNN for mobile malware detection, in: 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), 2019, pp. 193–198, http://dx.doi.org/10.1109/CoDIT.2019.8820510.

[68] S. Zhang, D. Cheng, Z. Deng, M. Zong, X. Deng, A novel kNN algorithm with data-driven k parameter computation, Pattern Recognit. Lett. 109 (2018) 44–54, http://dx.doi.org/10.1016/j.patrec.2017.09.036, URL https://www.sciencedirect.com/science/article/pii/S0167865517303562, Special Issue on Pattern Discovery from Multi-Source Data (PDMSD).

[69] V. Vincent, M. Wannes, D. Jesse, Transfer learning for anomaly detection through localized and unsupervised instance selection, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, pp. 6054–6061.

[70] J. Auskalnis, N. Paulauskas, A. Baskys, Application of local outlier factor algorithm to detect anomalies in computer network, Elektronika Ir Elektrotechnika 24 (3) (2018) 96–99.

[71] W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things, IEEE Access 6 (2018) 6900–6919, http://dx.doi.org/10.1109/ACCESS.2017.2778504.

[72] J. Mancebo, F. García, C. Calero, A process for analysing the energy efficiency of software, Inf. Softw. Technol. 134 (2021) 106560, http://dx.doi.org/10.1016/j.infsof.2021.106560, URL https://www.sciencedirect.com/science/article/pii/S0950584921000446.

[73] Creative Research Systems, Sample size calculator, 2019, https://www.surveysystem.com/sscalc.htm.

[74] M. Weber, C. Kaltenecker, F. Sattler, S. Apel, N. Siegmund, Twins or false friends? A study on energy consumption and performance of configurable software, in: Proceedings of the 45th International Conference on Software Engineering, ICSE '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 12.

[75] N. Siegmund, A. Grebhahn, S. Apel, C. Kästner, Performance-influence models for highly configurable systems, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 284–294, http://dx.doi.org/10.1145/2786805.2786845.

[76] M. Bauer, A comparison of six constraint solvers for variability analysis, Tech. rep., University of Passau, 2019.

[77] J.M. Horcas, J.A. Galindo, M. Pinto, L. Fuentes, D. Benavides, FM fact label: A configurable and interactive visualization of feature model characterizations, in: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B, SPLC '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 42–45, http://dx.doi.org/10.1145/3503229.3547025.

[78] Z. Wang, Z. Dai, B. Póczos, J. Carbonell, Characterizing and avoiding negative transfer, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019, pp. 11285–11294, http://dx.doi.org/10.1109/CVPR.2019.01155.

[79] A.K. Sandhu, R.S. Batth, Integration of artificial intelligence into software reuse: An overview of software intelligence, in: 2021 2nd International Conference on Computation, Automation and Knowledge Management, ICCAKM, 2021, pp. 357–362, http://dx.doi.org/10.1109/ICCAKM50778.2021.9357738.

[80] D. Tan, M. Suvarna, Y. Shee Tan, J. Li, X. Wang, A three-step machine learning framework for energy profiling, activity state prediction and production estimation in smart process manufacturing, Appl. Energy 291 (2021) 116808, http://dx.doi.org/10.1016/j.apenergy.2021.116808, URL https://www.sciencedirect.com/science/article/pii/S030626192100310X.

[81] T. Mangels, A. Murarasu, F. Oden, A. Fishkin, D. Becker, Efficient analysis at edge, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, IEEE, L'Aquila, Italy, 2017, pp. 85–90.

[82] C. Feng, Y. Wang, Z. Zhao, T.Q.S. Quek, M. Peng, Joint optimization of data sampling and user selection for federated learning in the mobile edge computing systems, in: 2020 IEEE International Conference on Communications Workshops (ICC Workshops), IEEE, Dublin, Ireland, 2020, pp. 1–6, http://dx.doi.org/10.1109/ICCWorkshops49005.2020.9145182.

[83] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, K. Lauenroth, Quality-aware analysis in product line engineering with the orthogonal variability model, Softw. Qual. J. 20 (3) (2012) 519–565.

[84] L. Lesoil, M. Acher, A. Blouin, J.-M. Jézéquel, Deep software variability: Towards handling cross-layer configuration, in: 15th International Working Conference on Variability Modelling of Software-Intensive Systems, VaMoS '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 8, http://dx.doi.org/10.1145/3442391.3442402.

[85] M.H. Alsafrjalani, T. Adegbija, Tasat: Thermal-aware scheduling and tuning algorithm for heterogeneous and configurable embedded systems, in: Proceedings of the 2018 on Great Lakes Symposium on VLSI, GLSVLSI '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 75–80, http://dx.doi.org/10.1145/3194554.3194576.

[86] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, X.S. Shen, TOFFEE: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing, IEEE Trans. Cloud Comput. 1 (2019) 1, http://dx.doi.org/10.1109/TCC.2019.2923692.

[87] A. Gamatié, G. Devic, G. Sassatelli, S. Bernabovi, P. Naudin, M. Chapman, Towards energy-efficient heterogeneous multicore architectures for edge computing, IEEE Access 7 (2019) 49474–49491.

[88] M. Couto, P. Borba, J. Cunha, J.P. Fernandes, R. Pereira, J. Saraiva, Products go green: Worst-case energy consumption in software product lines, in: Proceedings of the 21st International Systems and Software Product Line Conference - Volume A, SPLC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 84–93, http://dx.doi.org/10.1145/3106195.3106214.

[89] D.-J. Munoz, J.A. Montenegro, M. Pinto, L. Fuentes, Energy-aware environments for the development of green applications for cyber–physical systems, Future Gener. Comput. Syst. 91 (2019) 536–554, http://dx.doi.org/10.1016/j.future.2018.09.006, URL https://www.sciencedirect.com/science/article/pii/S0167739X18307295.

[90] A. Andoni, P. Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, in: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 459–468, http://dx.doi.org/10.1109/FOCS.2006.49.

[91] Z. Zhao, Q. Liu, W. Cao, D. Lian, Z. He, Self-guided information for few-shot classification, Pattern Recognit. 131 (2022) 108880, http://dx.doi.org/10.1016/j.patcog.2022.108880, URL https://www.sciencedirect.com/science/article/pii/S0031320322003612.

[92] S. Herbold, J. Grabowski, S. Waack, Calculation and optimization of thresholds for sets of software metrics, Empir. Softw. Eng. 16 (6) (2011) 812–841, http://dx.doi.org/10.1007/s10664-011-9162-z.