Twins or False Friends? A Study on Energy Consumption and Performance of Configurable Software

Max Weber*, Christian Kaltenecker†, Florian Sattler†, Sven Apel†, Norbert Siegmund*
* Leipzig University, Germany
† Saarland University, Saarland Informatics Campus, Germany

***Abstract*—Reducing energy consumption of software is an increasingly important objective, and there has been extensive research for data centers, smartphones, and embedded systems. However, when it comes to software, we lack working tools and methods to directly reduce energy consumption. For performance, we can resort to configuration options for tuning response time or throughput of a software system. For energy, it is still unclear whether the underlying assumption that *runtime performance correlates with energy consumption* holds, especially when it comes to optimization via configuration. To evaluate whether and to what extent this assumption is valid for configurable software systems, we conducted the largest empirical study of this kind to date. First, we searched the literature for reports on whether and why runtime performance correlates with energy consumption. We obtained a mixed, even contradicting picture from positive to negative correlation, and that configurability has not been considered yet as a factor for this variance. Second, we measured and analyzed both the runtime performance and energy consumption of 14 real-world software systems. We found that, in many cases, it depends on the software system's configuration whether runtime performance and energy consumption correlate and that, typically, only few configuration options influence the degree of correlation. A fine-grained analysis at the function level revealed that only few functions are relevant to obtain an accurate proxy for energy consumption and that, knowing them, allows one to infer individual transfer factors between runtime performance and energy consumption.**

I. Introduction

The energy demand of computing systems has been rapidly increasing for decades, with an ever growing number of users, devices, and applications. Recent developments in deep learning, big data analysis, and cloud computing further increase this demand [1], [2], [3]. While it is hardware that consumes energy, it is the software that controls how long and how intensive the hardware is used.

One powerful lever to optimize non-functional properties such as energy consumption and performance of software systems is *configurability* [4], [5], [6]. Recent work in this area demonstrates that a proper configuration can speed up a system's runtime performance by orders of magnitude [7], [5], [6]. Thus, it is not surprising that developers increasingly include configuration options in their code base [4], [8]. Clearly, configurability can serve as a means to optimize *runtime performance*, but can we reduce *energy consumption* of configurable software systems in a similar way?

The backbone of most optimization techniques in this area are surrogate regression models [5], [6], [9], which are able to estimate performance of a given software configuration. Constructing such a model usually requires extensive upfront measurements of a diverse set of software configurations to obtain a proper learning set [10]. While measurements are often straightforward to obtain for runtime performance and can even be obtained at the statement level (e.g., via profiling or code instrumentation [11], [12]), accurate and fine-grained energy consumption measurements are challenging for several reasons [13]: Measuring the energy consumption of software includes also the energy consumption of the underlying hardware and other running processes on the system, which gives rise to measurement uncertainty and bias. Worse, the physical measurement process has fundamental limitations regarding precision (measurement accuracy) and temporal resolution (sampling rate) [14]. In general, energy measurements are more prone to noise than performance measurements, because energy-measurement devices are more inaccurate compared to the internal clock that is used for performance measurements [15], [16], [17]. So, as a matter of fact, a direct measurement of energy consumption is complicated, time-consuming, and inherently noisy. Hence, a *cheap and accurate proxy measure* for energy consumption is clearly desirable. As different research studies [18], [19], [20] and practitioners guides [21], [22] suggest, performance might be exactly that proxy. In this vein, we reformulate our initial question:

> Can we use runtime performance as a proxy measure for energy consumption, and thus reduce energy consumption by performance tuning of software configurations?

The key is whether there is a *configuration-dependent correlation* between runtime performance and energy consumption. That is, do configuration options affect energy consumption in a similar way as they affect runtime performance? Although there is anecdotal evidence and isolated studies on the correlation of runtime performance and energy consumption (see Section III-B), answering this question is far from trivial and has not been studied in the context of configurability, despite its relevance for industry and society.

To answer our research question we conduct several experiments. First, we search relevant literature on the relation

between energy consumption and runtime performance to assess the state of the art about reported possible causes for an observed positive, negative, and absent correlation between energy consumption and runtime performance. This way, we obtain a broad picture about the validity of a performance proxy and ascertain whether configurability has been taken into account so far. Second, we conduct a series of experiments measuring runtime performance and energy consumption of a diverse set of configurable, real-world software systems analyzing the correlation of runtime performance and energy consumption across their configuration spaces. This includes the analysis of the whole configuration space as well as parts of it, down to the level of individual configuration options or interactions thereof. Finally, we conduct an experiment involving fine-grained function-level energy measurements to trace possible causes for correlation and non-correlation to the function level. That is, we selectively increase the resolution of our measurements from system level down to code level and investigate the energy–performance correlation of configuration options at each level.

We found strong correlations between performance and energy consumption for all subject systems averaged over the configuration space (i.e., reducing response time reduces energy consumption). However, when considering only a subset of (similar) configurations, we observe that this correlation may break down and even reverse. This can have severe consequences in scenarios when changing a running configuration to a similar one that reduces response time, energy consumption might even increase. We found that this behavior can be traced to individual configuration options and interactions that affect certain functions in the code such that the correlation may vary depending on which option is active. Moreover, we found that few functions exhibit a distinctive *transfer factor*, enabling us to transfer response times to energy consumption and vice versa depending only on a few options. Knowing these functions and the corresponding configuration options allows for the computation of these transfer factors and thereby for improving the applicability of performance as a proxy. Our analysis of the literature supports these findings not only by showing a mixed picture of performance–energy correlation, but also by pointing to common causes of correlation, such as caching, and multi-threading, all representing functionality that is often encapsulated and activated by individual options.

Overall, this paper makes the following contributions:

- We analyzed 75 studies that report on empirical findings about the energy–performance correlation of software systems, obtaining a mixed picture on whether and why energy consumption and runtime performance correlate.
- We conducted a series of experiments measuring performance and energy consumption of various configurations of 14 real-world software systems, analyzing how configuration options and their interactions affect the correlation between energy and performance. Based on the results of our experiments, we extract insights and deduce actionables for reducing energy consumption of configurable software systems.

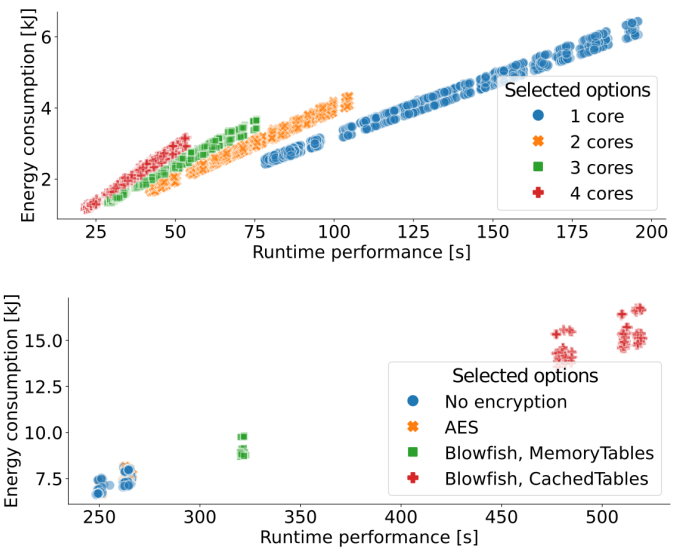

Fig. 1: Performance compared to energy consumption of all measured configurations of x264 (top) and HSQLDB (bottom). Colors and shapes represent different values of the selected configuration option.

- We traced the effects of configuration options on the correlation between energy consumption and runtime performance down to the function level, thereby facilitating understanding the causes, energy debugging, and optimization at the code level.
- We release all material, including measurement data (2.5 years of CPU time), scripts, and figures, on a publicly accessible supplementary Web page [1].

As such a study has not been conducted before, our findings contribute to the understanding of the *configuration-dependent correlation* between runtime performance and energy consumption of software systems. At the same time, identifying influential configuration options enables practitioners optimizing energy consumption and performance, only by measuring runtime performance. Tracing correlation variance at the system level down to functions provides developers a useful lever for energy debugging.

II. Correlation of Energy Consumption and Performance

The measure of energy consumption quantifies the amount of energy consumed that is required to resolve a task. It is calculated by integrating the power draw over time. It is reasonable to expect that longer execution times result in higher energy consumption. However, the relation between energy consumption and runtime performance is not straightforward [23], [21], [24], [25], [26]. Therefore, we distinguish three *modes of correlation*: *positive*, *negative*, and *no correlation*.

[1]Supplementary Web page: https://zenodo.org/record/7544891

Positive correlation means that, the longer a task runs, the more energy it consumes. Conversely, reducing execution time (i.e., improving performance) implies saving energy. That is, performance-optimal software configurations are likely to be also energy-optimal configurations.

Negative correlation arises when a higher energy consumption is associated with shorter execution times or vice versa. That is, there is a trade-off between the two, and one needs to decide to trade runtime performance for energy consumption.

Absent correlation implies that there is no dependency or observable relation between runtime performance and energy consumption. This is the case when, for the same amount of consumed energy, tasks require varying execution times or vice versa. Finding this mode of correlation in a configuration space enables users to optimize one measure (either runtime performance or energy consumption) without having to care about the other (i.e., there is no trade-off). This implies that runtime performance is no proper proxy for energy consumption.

The three modes of correlation are illustrated in Figure 1: For both subject systems, the set of all configurations follow the general pattern that longer execution times are associated with more energy consumption. Splitting the set of all configurations according to specific options can improve correlation between energy consumption and runtime performance, though. Figure 1 (top) shows option *Cores* of the video encoder X264. We can see that, deciding between the different settings for *Cores*, the resulting subsets of configurations approach a line in the plot. That is, for each setting of the option, we can use runtime performance as a proxy for energy consumption, though not globally. This is not always the case, though. Figure 1 (bottom) shows HSQLDB's options *Encryption* and *Tables*. Enabling *Blowfish* increases the execution time *as well as* the energy consumption. If we enable *Blowfish* and *MemoryTables* together, we see that all configurations are vertically arranged around 325 seconds. In this cluster, we can tune the energy consumption by changing configurations without degrading performance. This is an example of absent correlation.

III. Analysis of the State of the Art

As the first step to assess the relationship between runtime performance and energy consumption, we searched and analyzed the literature reporting on experiments that involve both sides. To this end, we systematically collected a set of papers, read them to identify relevant work, and analyzed the reported causes of correlation.

A. Methodology

First, we assembled an *initial* set of papers by searching for keywords in relevant software engineering venues (ICSE, ICPE, ESEC/FSE, and ASE), in particular, the keywords "*energy*", "*power*", and "*green*" in the title. We focused on recent work released between 2019 and 2022. The rationale is to set the starting point of the literature review on most recent software engineering work that copes with analyzing the energy consumption of software systems. In a second step, we extended the set of papers by those that have been cited by the papers already in our batch, following a *backward snowballing* method [27], [28]. This includes papers released before 2019 (the oldest from 1995) and papers from a wider range of journals and conferences (including TSE, MSR, CASES, MOBILESoft, HPCS). This way, we retrieved a set of 313 potentially relevant papers. Note that our objective here is not to conduct a formal or exhaustive literature survey, but rather to obtain a large and robust overview of performance–energy studies and to learn whether there is consensus in the literature about causes of performance–energy correlation and what factors strengthen and weaken a positive correlation.

We examined all 313 papers to identify those that discuss the relation of runtime performance and energy consumption. We deem a paper relevant if it measures energy consumption and runtime performance of software in combination. Furthermore, the paper in question has to analyze and discuss the link between the two. Here, a correlation analysis, a trade-off analysis, or a discussion of the relation are relevant inclusion criteria. Following empirical standards for literature studies [27], [28], two authors discussed whether the papers match our criteria. This step resulted in 76 relevant papers (out of 313 papers). From this set, we collected the reported explanations for why runtime performance and energy consumption correlate and abstract causes, which we have iteratively revised and refined, similar to card sorting.

B. Results

From the 76 papers that we have analyzed, we found 42 papers reporting a positive, 20 a negative, and 31 no correlation. Some of the papers reported multiple modes of correlation in their experiments.

From the relevant papers, we extracted the causes of a specific correlation mode to learn in which circumstances we can use runtime performance as a proxy for energy consumption right away. From the 76 relevant papers, 32 did not identify or discuss possible reasons for (absent) correlation. The remaining 44 papers identified and discussed, in total, 13 different causes of why (and why not) runtime performance and energy correlate. We summarized the eight causes related to configurability in Table I, showing the mode of correlation, the extracted cause, and excerpts from the literature.

Notably, the causes for positive and negative correlation overlap. For example, caching has been named to cause a positive *and* a negative correlation [33], [40]. Reported reasons for this contradiction are different setups and workloads.

Overall, our literature analysis reveals a mixed picture of the state of the art on the correlation between energy consumption and runtime performance. Many papers reported that runtime performance and energy consumption correlate, but often even in opposite directions. Fortunately, we found several reasons stated by the papers' authors of why they have observed a certain mode of correlation. Mostly, these reasons point to specific hardware devices, memory-processing trade-offs (e.g., via caching), and intensive IO operations. What is interesting is that all these aspects can often be controlled or, at least,

TABLE I: Overview of the three modes of correlation with the corresponding causes and excerpts from the literature.

Mode	Cause	Finding
Pos	CPU-intensive tasks [29], [30], [31], [32]	Rashid et al. found "a good linear regression of energy vs. time" and a "large part of the energy consumed is determined by the time performance" [30].
	Memory and caching [33], [34]	Subramaniam et al. found that "there is a correlation between power and performance related activity such as L2 data cache miss at a certain workload" [33].
	Tuning HW params [35], [36], [37]	Rauber et al. found that, using a larger number of threads, energy consumption as well as runtime increases when varying the number of threads and processor frequency [35].
Neg	CPU-intensive tasks [38], [39]	Oliveira et al. found that "performance is often not a proxy for energy consumption", which is why they concluded that "faster != greener" [38].
	Memory and caching [40], [41]	Michanan et al. found promising energy saving opportunities, but not always time savings, concluding that "even something very low-level like the cache system can impact the power-performance analysis significantly and unpredictably" [40].
None	Network comm. [42], [24], [43]	Malavolta et al. found that, for energy consumption, caching has no significant impact on leading a progressive Web application (PWA), whereas the loading time diverges, because the PWA has to be loaded through the network if it is not in the cache [42].
	IO-intensive systems [38], [44]	With their multi-core thread shuffling technique, Cai et al. achieve up to 56% energy savings without any performance loss for huge amounts of data [44].
	Tuning CPU&HW params [41], [25]	Pinto et al. conclude from their experiments varying numbers of threads, task division strategies, and workloads, that "being 'faster' clearly has little correlation with being 'greener' for concurrent programs on multi-core architectures" [25].

affected by configuration options in contemporary software systems. We conjecture that the different modes of correlation can also emerge in a single software system depending on the configurations that are executed. This would, however, make the proxy question more nuanced, as it would depend on the individual configuration options that are selected. We will shed light on this issue in our empirical study in Section V.

C. Related Studies

Closest to our paper are experiments reporting on variations of performance arising from a software's configuration [25], [33], [35], [34], [36]. The most complex experiment reported so far considered only five configuration options of a single software system [36]. This strongly indicates that there is only a limited coverage of configurability in the analysis of the correlation between energy consumption and runtime performance. Clearly, effects stemming from interactions among configuration options as well as different types of options cannot be studied with such experimental setups. Interestingly, we see that positive and no correlation are on par for these experiments [25], indicating that even for a small number of options, there might be good reasons not to use runtime performance as a proxy for energy optimization.

IV. Experiment Setup

In this section, we define our research questions and describe the measurement setup, including subject systems. We make all information on subject systems and experiment data including energy consumption and runtime performance measurements available on our supplementary Web site [2].

A. Research Questions

System-Level Correlation. To use runtime performance as a proxy measure for energy consumption, we have to evaluate how the two align with each other. Since the literature provides a mixed picture, we investigate whether both properties align over the whole configuration space or only in parts of it. So, we formulate the following two research questions:

$RQ_{1.1}$	*To what extent does runtime performance and energy consumption correlate across the configuration space?*
$RQ_{1.2}$	*Does this correlation stay constant across the configuration space or are there clusters of similar performing configurations that exhibit different correlations?*

Option-Level Correlation. Next, we are interested in to what extent individual configuration options (e.g., number of threads, cache size, encryption mode) affect the correlation between energy consumption and runtime performance. Changing a configuration usually corresponds to the (de-)selection of system functionality. By analyzing the correlation at the option-level, we gather evidence on whether runtime performance is a viable proxy for energy optimization for option-specific system functionality. This enables us to find (positively) correlating options and interactions, should they exist. So, we formulate the following two research questions:

$RQ_{2.1}$	*Can system-level correlations be traced to individual configuration options and interactions?*
$RQ_{2.2}$	*What is the fraction of options and interactions that have an effect on the correlation?*

Function-Level Correlation. While $RQ_{2.1}$ traces the modes of correlation to individual options, we want to pinpoint functions in which the correlation is influenced by configurations. The goal is to identify possible means for developers to use the execution time of a function as a proxy for energy consumption, thereby enabling code-based energy optimization. Such information has been requested by developers to have an actionable tool for a green software development [45], [46]. So, we formulate the next research question:

RQ_3	*Does configuration-dependent correlation exist at the function level?*

[2]Supplementary Web page: https://zenodo.org/record/7544891

B. Measurement Setup

We conducted all system-level runtime performance and energy measurements on two clusters, equipped with a dedicated energy measurement device. The first cluster uses Intel NUCs with i7-8559U CPUs, 32 GB DDR4-2666, and 500 GB NVME SSD. Every machine is plugged into a GUDE 8045-1 PDU (power distribution unit), measuring the power draw with an interval of 1 second per outlet. The second cluster consists of Intel Core i5-4590 machines having 16 GB RAM, 256 GB SSD, and depending on the subject system, a minimal installation of Ubuntu 16.04, Ubuntu 18.04, or Debian 9. Each machine is connected to a IPT iPower P1 PDU.

To measure energy consumption at the function level, we set up a dedicated single machine running an Intel Core i5-7500 with 4 GB RAM and 120 GB SSD, and a minimal installation of Ubuntu 18. This machine contains a 2.5 kHz energy measurement system [47], which is based on the INA226 power measurement chip. No other tasks were running during performance and energy measurements. We conducted pre-experiments to calculate the measurement bias (i.e., standard deviation of repeating runs). Based on these experiments, we repeated our final measurements 3 times, taking the average. When measuring client-server systems, such as Web servers and databases, we used multiple machines in parallel, where one acted as server and one or more as clients. In total, we invested 2.5 years of CPU time obtaining the largest performance–energy data set on configurable software systems we are aware of.

C. Subject Systems

To obtain a representative set of systems, we selected 14 subject systems from different application domains, of different sizes, using different programming languages, and with varying configuration spaces. In particular, we selected subject systems from related studies [12], [48], [10], [11], [5] that also measure energy consumption or runtime performance of configurable software systems. We opted for a mixed measurement setup that carefully balances measurement effort (in a feasible amount of time) and coverage of the configuration space. For small subject systems (BROTLI and LRZIP), we measured all configurations. For larger systems, such as JUMP3R and KANZI, we applied t-wise sample with t=2 to obtain a feasible set of configurations for measurements, including possible pair-wise interactions. For the remaining 10 systems, we have pre-selected a set of configuration options, whose documentations hint at an effect on runtime performance or energy consumption, this way, omitting debugging or help options.

D. Basic definitions

In our study, we consider a set $\mathcal{S}$ of subject systems, whereas each subject system $s \in \mathcal{S}$ provides both a set of valid configurations $\mathcal{C}_s$ and a set of configuration options $\mathcal{O}_s$. Each configuration $c \in \mathcal{C}_s$ assigns a value to each option $o \in \mathcal{O}_s$. Function $o(c)$ maps a configuration to the value of the corresponding configuration option o. The domain of this function $\text{dom}(o)$ (i.e., the possible values for a given option) is typically either binary ($o(c) : \mathcal{C}_s \to \{0,1\}$) or numeric ($o(c) : \mathcal{C}_s \to \mathbb{Z}$). The functions $\mathcal{E}_s : \mathcal{C}_s \to \mathbb{R}$ and $\mathcal{P}_s : \mathcal{C}_s \to \mathbb{R}$ map a configuration of the subject system s to its measured energy and runtime performance value, respectively. The average of all energy and performance values is defined as $\overline{\mathcal{E}}_s = \frac{1}{n_s}\sum_{i=1}^{n_s} e_i$ and $\overline{\mathcal{P}}_s = \frac{1}{n_s}\sum_{i=1}^{n_s} p_i$, where e_i and p_i are the energy and performance values of the ith configuration, and n_s number of all measured configurations $\mathcal{M}_s$.

TABLE II: Overview of subject systems, including application domain, programming language, lines of code (LOC), number of valid configurations ($|\mathcal{C}|$), and configuration options ($|\mathcal{O}|$).

System	Domain	Language	LOC	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{O}\rvert$
7Z	Compression	C++	164 977	68 640	10
APACHE	Web server	C	235 825	13 441	11
BROTLI	Compression	C	34 501	181	2
EXASTENCILS	Code Generator	Scala	71 240	86 058	10
HSQLDB	Database	Java	187 632	864	14
JUMP3R	MP3 encoder	Java	20 940	933 120†	14
KANZI	Compression	Java	21 805	1 984†	6
LLVM	Optimizer	C	7 675 240	65 536	16
LRZIP	Compression	C	15 475	5 184	12
MONGODB	Database	C++	4 918 542	6 840	15
NGINX	Web server	Go	147 246	4 416	14
POSTGRESQL	Database	C	944 473	864	7
VP8	Video encoder	C/C++	352 896	2 736	11
X264	Video encoder	C	68 475	3 840	11

† We used 2-wise sampling to measure a representative subset of configurations.

V. PERFORMANCE–ENERGY STUDY

To answer our research questions, we analyze the energy consumption and runtime performance of 14 real-world software systems presented in Table II.

A. System-Level Correlation

Operationalization: To answer $RQ_{1.1}$, we quantify the correlation between energy consumption and runtime performance of the entire configuration space using Pearson's correlation coefficient. Pearson's correlation coefficient states how well the mapping between runtime performance and energy consumption can be represented by a linear relation. To investigate whether the correlation holds in all or only some parts of the configuration space ($RQ_{1.2}$), we compute the correlation for slices of the data. Specifically, we split the performance dimension of each subject system s into 19 slices $\mathcal{L}_{s,1}, \mathcal{L}_{s,2}, \ldots, \mathcal{L}_{s,19} \subset \mathcal{M}_s$ of similar performance range (i.e., time interval) and compute the correlation of all configurations falling into that slice. Each slice represents $\frac{1}{10}$ of the runtime performance range between the slowest and fastest configuration. To be robust against unfavorable splits, we overlap each slice by half of the slice width, resulting in a total of 19 slices. For each slice $\mathcal{L}_{s,i}$, we compute Pearson's correlation coefficient $r_{s,i}$:

$$r_{s,i} = \frac{\sum_{c\in\mathcal{L}_{s,i}} \left(\mathcal{P}_s(c) - \overline{\mathcal{P}}_s\right)\cdot\left(\mathcal{E}_s(c) - \overline{\mathcal{E}}_s\right)}{\sqrt{\sum_{c\in\mathcal{L}_{s,i}} \left(\mathcal{P}_s(c) - \overline{\mathcal{P}}_s\right)^2 \cdot \sum_{c\in\mathcal{L}_{s,i}} \left(\mathcal{E}_s(c) - \overline{\mathcal{E}}_s\right)^2}}$$

The coefficient $r_{s,i}$ ranges from -1 to 1. A coefficient of 0 indicates absent correlation, 1 indicates a perfect positive correlation, and -1 a perfect negative correlation. We refer to r also as correlation value.

To quantify the extent of possible deviations of (sets of) configurations from the correlation, we additionally fit a linear function with all measurements available: $f_s : \mathcal{P}_s \to \mathcal{E}_s$ using linear regression. That is, given $p \in \mathcal{P}_s$, we ask how well can we predict $e \in \mathcal{E}_s$. This way, we learn how well runtime performance acts as an estimator for energy consumption. The rationale is that the extent and distribution of the prediction error maps the unrelatedness of runtime performance and energy consumption depending on the configuration space.

Results: Table III lists the results of the correlation analysis. Pearson's correlation for the whole configuration space is shown in the first column. The column for the slice correlation lists the number of slices that have a certain strength of correlation. Finally, the last two columns show the mean absolute percentage error (MAPE; a standard measure for model accuracy[9], [49], [11]) and its according standard deviation (StD) of the linear model fitted on all measurements.

TABLE III: Correlation between energy consumption and runtime performance. Green cells show an error below 5 %.

		Correlation modes						
System	Pearson	SN	MN	WN	MP	SP	MAPE	StD
7z	0.98	1	0	1	7	6	14.7	11.7
Apache	0.99	0	0	0	2	0	2.2	1.8
brotli	1.00	0	0	0	0	11	9.3	14.1
exastencils	0.98	0	0	0	6	13	5.7	4.8
HSQLDB	0.99	0	1	6	1	0	3.0	2.2
jump3r	0.99	1	1	2	0	6	10.0	8.2
kanzi	0.97	2	1	6	3	7	70.4	64.1
LLVM	0.99	0	0	0	0	19	0.7	0.5
lrzip	0.99	1	0	3	2	5	31.0	28.8
MongoDB	0.99	0	0	6	9	4	1.6	1.2
nginx	0.99	0	0	1	1	3	8.2	19.3
PostgreSQL	0.95	1	0	6	9	2	1.2	0.9
VP8	0.96	0	0	7	3	9	22.0	17.5
x264	0.96	0	0	3	6	10	10.3	7.1

Pearson's correlation of different configurations between energy consumption and runtime performance; SN: strong negative ($r \leq 0.7$); MN: moderate negative ($-0.7 < r \leq -0.3$); WN: weak or no ($-0.3 < r \leq 0.3$); MP: moderate positive ($0.3 < r \leq 0.7$); SP: strong positive ($r \geq 0.7$); MAPE: mean absolute percentage error of all configurations given runtime performance predicting energy consumption; STD: standard deviation of the distribution of prediction errors.

We observe a nearly perfect linear correlation across all subject systems when taking the whole configuration space into account. Even the lowest correlation value we found (x264) is still very large (0.963). To obtain deeper insights into the distribution of performance–energy correlation across the configuration space, we plot the energy consumption and runtime performance for all configurations (dots) in Figure 2 for a selected subset of systems (plots for all systems can be found at our supplementary Web page).

The key observation is that, although we observe a general trend of an increasing energy consumption with increasing response time, this trend does not hold for all configurations in

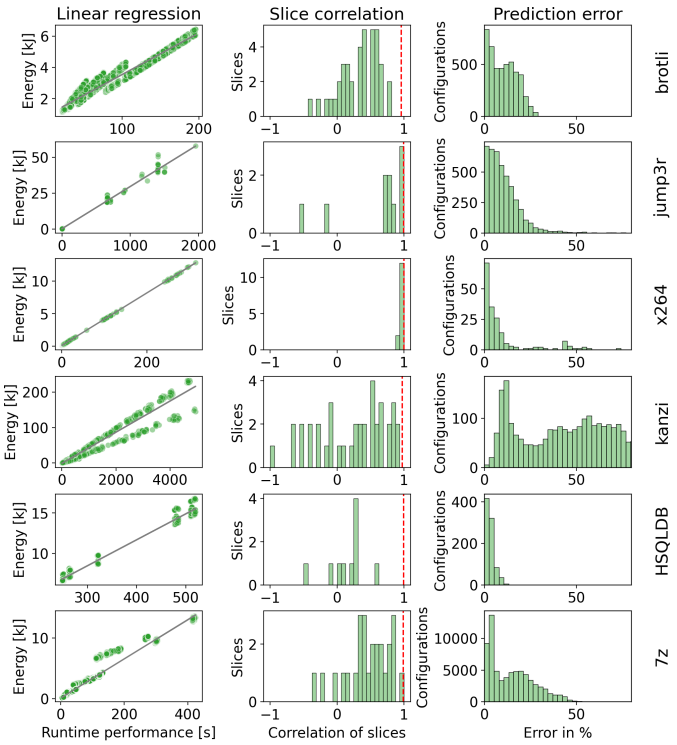

Fig. 2: Correlation of brotli, jump3r, x264, MongoDB, and 7z. Linear regression shows the fitted linear function for all measurements; Slice correlation shows a histogram for the number of slices of runtime performance data arranged according to their individual correlation coefficient; Prediction error depicts the MAPE for all configurations as a histogram.

the respective configuration spaces. Computing the correlation between energy consumption and runtime performance of subsets of configurations shows a very heterogeneous picture. For instance, for Kanzi, in 3 out of 19 slices, we observe a strong negative correlation, although the overall correlation is strongly positive (0.977). That is, for these three areas of the configuration space, increasing the response time, decreases energy consumption and vice versa.

In HSQLDB, none of the subsets of configurations have a strong positive correlation. This is an instance of Simpson's paradox, that is, detecting a strong correlation for all the data while simultaneously detecting no or opposite correlation for subsets of the same data [50]. This observation hints at a significant insight into system-level correlation: Changing configurations locally in the configuration space may not follow the global trend of a strong performance–energy correlation. Only brotli and LLVM seem to be consistent in the correlation of the subsets with the overall correlation.

Insight: Even a perfect linear correlation between energy consumption and runtime performance is no guarantee for having a good proxy for energy consumption for individual configurations. That is, energy consumption may still vary substantially for configurations with similar performance.

Actionable: To decide whether runtime performance is an appropriate proxy for energy consumption, it is necessary to inspect whether the correlation holds also for relevant slices of the configuration space. Considering the whole configuration space, runtime performance is a suitable indicator for energy efficiency, but being confined in a small subspace (e.g., by functional requirements or existence of a running configuration), runtime performance might become an unreliable proxy.

Looking deeper into the data (e.g., Figure 2, left column), we find that some configurations have a different energy consumption for a similar runtime performance and vice versa. To quantify for which software systems a performance proxy is suitable, we review the MAPE in Table III. For five systems, the linear model estimates energy consumption with a low MAPE: APACHE, HSQLDB, LLVM, MONGODB, and POSTGRESQL. This is because energy consumption and runtime performance values are close to the regression line across the whole configuration space (as shown in Figure 2). The other systems have a high MAPE (over 5 %) or a high standard deviation, which means that we cannot use runtime performance as a reliable proxy for many of the configurations of these systems [51]. Figure 2 (right column) shows the underlying distribution of the MAPE. We can see that there are groups of configurations that diverge substantially from the regression line (e.g., centered around 20 % error rate for 7z), which indicates multi-modal error distributions for some software systems.

Discussion: A configuration is often not selected without certain restrictions and functional requirements. Our results suggest that shrinking the configuration space can negatively influence the correlation value. In other words, the positive correlation might be an artifact of the combinatorially huge number of positively linked configuration decisions that get reduced when looking only at a subset of configurations. Strong indications for this are (i) configurations with a similar runtime performance (different groups) having no, weak, or even negative correlations, even if all configurations together have a very strong positive correlation; (ii) multi-modality in the error rate distribution, pointing to different degrees of correlation of subsets of configurations. Since a configuration is composed of the choices of several individual configuration options, we can already infer that some options must have a profound distinctive influence on the correlation value. To understand this effect better, we investigate next the role of individual configuration options and their interactions as possible drivers of a correlation between energy consumption and runtime performance.

B. Influence of Options and Interactions on the Correlation

Operationalization: To answer $RQ_{2.1}$, we quantify the correlation between energy consumption and runtime performance for individual configuration options using Pearson's correlation coefficient. Similar to $RQ_{1.2}$, we compute Pearson's correlation for individual slices, but now per option, to investigate whether the correlation of specific configuration options holds in the whole or only parts of the configuration space. For each configuration option $o \in \mathcal{O}$, we divide the configurations of each slice $\mathcal{L}_{s,i}$ into different sets such that each set contains only configurations in which the option has a fixed value.

Furthermore, we quantify the improvement (or deterioration) of prediction accuracy when dividing the set of all configurations by the values of individual configuration options. For this purpose, we fit a linear model $f_s^{o,v} : \mathcal{P}_s \rightarrow \mathcal{E}_s$ for all $o \in \mathcal{O}_s$ and all values $v \in \text{dom}(o)$ with each set of configurations of the individual configuration options. This way, we determine whether runtime performance is a reliable proxy for energy consumption when looking only at partitions of the configuration space whereby the partition is specified by an individual option's value. We can also identify highly correlating options and interactions with this method.

To this end, we compare the error of the global linear model using all configurations ($RQ_{1.2}$) with the error of the linear model of the best suitable option (lowest MAPE). A smaller error for a certain configuration option means that this configuration option explains some of the performance variance and thus fosters linear correlation. We use a threshold of 5 % as indicator for good models [51], [52]. To answer $RQ_{2.2}$, we report per subject system the number of configuration options that reduce the error below 5 %.

Interactions between configuration options are known to influence performance [51], [5]. We determine interacting configuration options by means of a qualitative analysis of the influence of configuration options. Specifically, we search for options with values that are locally clustered in certain areas of the performance–energy space, and we manually inspect whether a cluster can be isolated from all other configurations by selecting a combination of configuration options. Figure 1 illustrates this situation for HSQLDB. The Blowfish algorithm has a distinct characteristic that is easily separable from other values of Encryption. If we want to identify the cluster of configurations around 325 seconds, we need to select only the value MemoryTables from option TableType. Two of the authors conduct this analysis for all subject systems and all options to identify all interacting configuration options.

Results: In Table IV, we summarize our findings on the effect that individual configuration options and interactions have on the correlation between energy consumption and runtime performance ($RQ_{2.1}$ and $RQ_{2.2}$). The first column shows the mean value of the correlations per configuration option. Similar to $RQ_{1.1}$, we see a perfect positive correlation for all subject systems. The lowest correlation coefficient (BROTLI) is still strong (0.91).

The columns SN, MN, WN, MP, and SP of Table IV show the number of cases summarized over all 19 slices when a configuration options exhibits the corresponding correlation mode. Similar to $RQ_{1.2}$, we observe a mixed picture for most systems. We find even systems (e.g., MONGODB) for which all sets of configurations have a weak or moderate positive correlation. Compared to $RQ_{1.2}$, we observe that the

TABLE IV: Correlation between energy consumption and runtime performance based on configuration options for all subject systems. Green cells show an error below 5 %.

		Correlation modes							
System	Pearson	SN	MN	WN	MP	SP	MAPE	CO	CI
7z	0.98	33	5	86	156	207	7.9	1	3
Apache	0.99	0	0	23	16	54	4.0	0	0
brotli	0.91	7	0	2	20	122	6.2	0	0
exastencils	0.91	1	1	16	178	478	5.5	0	0
HSQLDB	0.97	1	15	77	42	9	2.8	0	3
jump3r	0.98	16	18	29	6	114	4.5	2	0
kanzi	0.96	12	13	50	42	117	23.7	4	2
LLVM	0.99	0	0	8	15	572	0.7	0	0
lrzip	0.98	11	2	40	48	87	7.1	1	3
MongoDB	0.99	10	9	156	224	84	1.5	0	3
nginx	0.99	0	0	36	35	73	4.3	0	0
PostgreSQL	0.99	0	0	3	32	129	1.2	0	0
VP8	0.97	5	2	51	150	141	4.7	2	3
x264	0.96	3	4	56	129	202	2.5	2	5

Pearson: mean value of the correlation coefficients for all values of different configuration options; SN: strong negative; MN: moderate negative; WN: weak or no; MP: moderate positive; SP: strong positive; MAPE: mean absolute percentage error of all configurations given runtime performance predicting energy consumption; CO (configuration option): number of configuration options that foster positive correlation; CI (configuration interaction): number of configuration options that are part of an interaction.

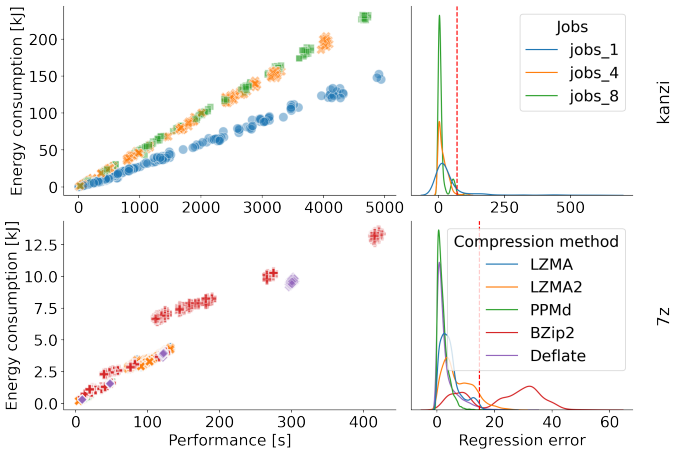

Fig. 3: Influence of configuration options on correlation for kanzi and 7z; Left: scatterplot of all configurations colored by different option values; Right: distribution of regression errors of all configuration options predicting energy consumption given runtime performance; Dashed red vertical line: regression error using all configurations.

correlation of configuration options per slice spreads more over the different modes of correlation. Hence, the strong correlation seen in $RQ_{1.1}$ can, at least, partially be attributed to an averaging global effect that does not hold when looking at individual slices.

Column MAPE in Table IV represents the highest achievable energy prediction accuracy when fitting individual models per configuration option on runtime performance data. Compared to $RQ_{1.1}$, the prediction error decreases for all subject systems. This indicates an option-specific correlation. For example, we observe a substantial drop in the error of kanzi from 70.4 % to 23.7 %. To illustrate this effect, we show in Figure 3 the configuration space of 7z. Here, we learn a separate model for each of the five compression modes of the option Compression method. Interestingly, when selecting BZip2, we observe a larger spread across the energy–performance space, leading to an increased error compared to the global model. This indicates that, for some values of configuration options, correlation is absent, and a performance proxy would fail when using one of these options.

lrzip and x264 exhibit error rates in which the error of the best model is substantially lower than the worst model and the average error over all models. The best regression model of x264, generated from the configuration option Cores, has an error of 2.5 %. That is, selecting a good configuration option for the split greatly improves model accuracy, suggesting that runtime performance is a good proxy. In Figure 1, we had shown the effect of dividing the configuration space into four sets, each with one value of Cores. We can clearly see that this split aligns the corresponding configurations along four different regression lines leading to more accurate energy estimates when considering each line individually, instead of averaging across them.

Answering $RQ_{2.2}$, we found the same pattern also for other systems: A small number of options (i.e., correlating options) is able to lower the prediction error. We report the number of these correlating options (CO) in Table IV. In total, there are 12 out of 155 configuration options that lower the error by more than 5 %. Across all systems, fitting a model per value of an option always reduces the error. That is, small to large deviations in correlation occur in all subject systems for the majority of options.

So far, we looked at individual configuration options, but we found also correlating interactions among options. That is, setting multiple configuration options to a certain value might affect the performance–energy correlation (CI in Table IV). In total, there are 22 configuration options involved in interactions that substantially affect the correlation. Notably, we were able to identify 14 options that have not been found by the previous analysis (CO in Table IV). This means, when taking interactions into account, not 12 but 26 options out of 155 lower the error rate. Interestingly, we found that seven subject systems have no interactions relevant for the performance–energy correlation and exactly those systems already have a low prediction error. This is a possible indicator that interactions may be responsible for changes in the performance–energy correlation for the other systems.

Discussion: Overall, we obtain a similar picture as in $RQ_{1.1}$ and $RQ_{1.2}$: The linear correlation across the whole energy and time range is nearly perfect for all systems. However, when correlating slices of data, we see all modes of correlation for different subsets of configurations by fitting the values of individual options. Furthermore, predicting energy consumption based on runtime performance of only a subset of configurations often reduces the prediction error. Interestingly, we also see that this applies only to a few options (12 out of

195 options across all systems). This is good news and bad news at the same time: It means that runtime performance usually works as a proxy for large portions of the configuration space and even for entire systems. The bad news is that, if it does not work, we need to identify the options causing the disruption of correlation. This can become costly as it requires sampling, measuring, and learning multiple sets of configurations.

Insight: In general, fitting a model mapping runtime performance to energy consumption for each value of an option always reduces the error across all subject systems.
Actionable: If the global error is too large (e.g., 70.4 % for KANZI), producing option-level prediction models can substantially reduce the error (e.g., 23.7 % for KANZI) and make runtime performance a reasonable proxy.

In our set of subject systems, there is one subject system that stands out with respect to prediction error: KANZI (see Figure 3). Even the best regression model has an error of 23.7 % (cf. Table IV). The reason for this large error is a high number of configurations with short runtime, such that only one or two energy measurements could be made in that time. So, the variance we learn with the regression models lies within the measurement deviations of the measurement device.

C. Function-Level Correlation

Operationalization: Answering RQ_3, we measure the runtime and energy consumption of all functions that are visited during program execution. To this end, we used the PERF profiling tool[3] for measuring the runtime within a function (self-time). Specifically, we used PERF's sampling-based profiling mode (with 1007 Hz sampling frequency) to reduce profiling overhead. To measure the energy consumption of the individual functions, we used a power measurement system with 2.5 kHz measurement frequency. For each function, we sum up the runtime values and integrate the power values to obtain the total runtime and energy consumption. The subject systems are again executed with different configurations. Because of the expensive measurement process, and since PERF is applicable only for C/C++ programs, we had to restrict the number of subject systems, as well as the number of different configurations to 200 per subject system. Due to the diversity of results of $RQ_{2.1}$, we selected BROTLI, X264, and LRZIP to include one typical candidate with a strong correlation (brotli), with mixed correlations at option level (lrzip), and with clearly separable option-level correlations (x264). To focus our analysis on relevant functions, we excluded functions that account for less than 0.1 % of the system-level runtime.

For each configuration, we compute the ratio between energy consumption and runtime to obtain singular transfer factor between both measures. The first column of Figure 4 shows the transfer factors of all configurations for three functions of LRZIP. If the transfer factors of all configurations are centered tight around one value, we can speak of an accurate performance proxy for the respective function. That is, the runtime of a function can be multiplied by a single factor to obtain the energy consumption. To test for a constant single transfer factor, we applied the dip test (Hartigans test of unimodality [53]) and calculated the coefficient of variation [54]. If the distribution of all transfer factors over all configurations is centered around one mode, and if the coefficient of variation is below 5 %, we can take the average transfer factor together with the runtime value as a proxy for energy consumption. Finally, we count the number of functions per subject system for which we identified a constant single transfer factor.

From $RQ_{2.1}$, we know that configuration options influence the relation between energy consumption and runtime performance at the system level. Therefore, we expect some functions to reflect this behavior, meaning that the distribution of transfer factors of some functions is expected to have a multi-modal distribution. To test whether the modes of the distribution can be assigned to individual configuration options, we split the set of configurations by the values of each option and repeat the test for a constant single transfer factor per value. We find a proxy (i.e., multiple transfer factors) if the values of the configuration option map the multi-modal distribution with a coefficient of variation less than 5 % per mode. We again count the number of functions per subject system for which we can successfully find multiple transfer factors that can be explained by a configuration option.

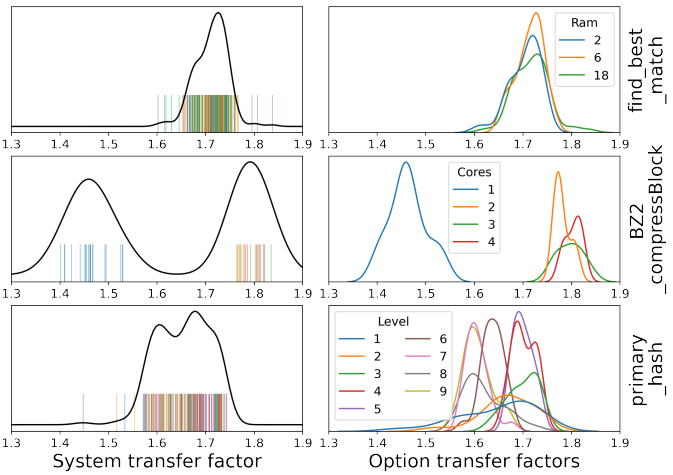

Fig. 4: Proxy for LRZIP for the functions find_best_match, BZ2_compressBlock, and primary_hash.

TABLE V: Number of functions of the transfer factor analysis including total number of functions ($|\mathcal{F}|$).

Name	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{F}_R\rvert$	$\lvert\mathcal{F}_S\rvert$	$\lvert\mathcal{F}_M\rvert$	$\lvert\mathcal{F}_W\rvert$
BROTLI	233	58	58	–	–
LRZIP	230	42	14	24	4
X264	391	93	–	43	50

$|\mathcal{F}_R|$: number of relevant functions; $|\mathcal{F}_S|$: number of single-factor functions; $|\mathcal{F}_M|$: number of multi-factor functions; $|\mathcal{F}_W|$: number of functions without transfer factors.

[3]perf: Linux profiling with performance counters. [Online: 2023, Jan]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

Results: In Table V, we summarize our results of the correlation at the function level, in which column $|\mathcal{F}|$ lists the total number of functions per subject system. We focus our analysis on those functions ($|\mathcal{F}_R|$) that contribute a share of, at least, 0.1 % of a system's runtime, which results in 58 (25 %), 42 (18 %), and 93 (23 %) functions for BROTLI, LRZIP, and X264. The analyzed functions make up a total execution time of 99.9 % per system.

Columns $|\mathcal{F}_S|$ and $|\mathcal{F}_M|$ list the number of functions for which we runtime multiplied with a constant transfer factor can act as a proxy for energy consumption. Column $|\mathcal{F}_S|$ counts functions for which a single transfer factor is sufficient. An example is find_best_match from LRZIP, plotted in Figure 4 first row: We can see that the transfer factors have a uni-modal shape, centered around 1.7. For BROTLI, we obtained a single transfer factor for all 58 performance-relevant functions ($|\mathcal{F}_R|$). This confirms our system-level findings of $RQ_{1.1}$, since BROTLIs configurations exhibit nearly a perfect correlation, as shown in the first row of Figure 2. While for LRZIP, there are 14 functions that can use the same single value as proxy, there is none for the 93 functions of X264.

Column $|\mathcal{F}_M|$ lists the number of functions for which we found multiple proxy values. We can get stable, precise proxy values if we selected the configuration options such that we properly split the multi-modal distributions. As an example, the second row of Figure 4 highlights function BZ2_compressBlock of LRZIP, for which we can observe two modes in the distribution of transfer factors. These two modes are controlled by option Cores. They apply when single threading or multi threading is selected in a configuration. In total, we found 24 functions for LRZIP and 43 functions for X264 with multiple transfer factors.

Column $|\mathcal{F}_W|$ shows the number of functions for which our classification was unable to find transfer factor that depend solely on a single configuration option. That is, interactions among options change the performance–energy transfer factor considerably. Overall, we were able to find proxy values for 139 out of 192 functions for all three subject systems.

Discussion: The fact that we are able to trace correlations observed at the system level down to the function level, is an intriguing result. We are even able to pin down individual configuration options that affect the correlation per function ($RQ_{2.2}$). That is, configuration options change the transfer factor between energy consumption and runtime performance for individual functions. This result has implications for future for energy hot-spot detection and code-level energy optimization, since we first have to identify the options for individual transfer factors and only then we can use performance as a proxy for energy consumption.

Insight: We are able to trace the effects of configuration options on the correlation between energy consumption and runtime performance down to function level, rendering not only system-level optimization via performance proxies feasible, but also energy optimization at code level.

Actionable: Since an exhaustive profiling approach, which includes energy measurements, is not feasible in practical settings, profiling should be applied only to influential options.

Note that, in our analysis, we concentrated solely on individual configuration options and their different values. Clearly, interactions between options may also affect the correlation, as we already found for $RQ_{2.2}$. This explains why we were not able to determine a proxy for half of the relevant methods for X264. However, even for functions that have been classified as without transfer factor (cf. column $|\mathcal{F}_W|$ in Table V), we can compute constant single transfer factors for a subset of the value ranges of an option. As an example, function primary_hash of LRZIP shows this aspect for option Level in Figure 4. Here, we can compute a transfer factor for 7 out of 9 values for this option. Only if option Level is set to 1 or 2, we see too large fluctuations, which might be caused by too short runtimes of the function or not considering interactions.

D. Implications for Practitioners

Our results have shown that runtime performance can be a viable proxy measure for energy consumption. However, our results have also shown that there are situations when the proxy measure becomes inaccurate or even misleading, and that a stakeholder needs to know when it is safe to use and when it needs more analysis to pin down an individual transfer factor per option. We suggest three scenarios on when and how to use performance as a proxy for energy consumption: (i) energy optimization through configuration, (ii) reconfiguration under performance constraints, and (iii) energy-efficiency improvement at code level.

Optimization: Our results from $RQ_{1.1}$ suggest that an initial optimization of energy consumption can be made with runtime performance data across. That is, it is reasonable to expect that a random sampling approach with runtime performance measurements on the entire configuration space provides a good indicator also energy-efficient configurations. However, this does not automatically mean that configurations with the same runtime performance have the same energy consumption. In fact, focusing on subsets of configurations clearly shows that, when optimizing energy consumption with performance constraints (i.e., sets of configurations with similar performance), their energy consumption can vary substantially. In this case, we need a learning approach based on energy measurements to quantify the influence of individual options and interactions on energy consumption (as found in $RQ_{1.2}$). Nevertheless, an-easy-to-follow heuristic is that the faster the runtime performance the lower the energy consumption.

Reconfiguration: In a reconfiguration scenario (i.e., a running application requires switching configuration options), we often are constrained by the current configuration, for example, since the running system relies on user-selected features. In this scenario, we choose a new configuration from a subset of the configuration space that is similar to the current configuration. Our results show that, in such cases, the correlation

often breaks down. This has profound implications: We cannot guarantee a similar (or even reduced) energy consumption even when performance might improve. In $RQ_{2.2}$, we identified the configuration options and interactions that have a distinctive (and possibly even contradicting) influence on the correlation. $RQ_{2.2}$ quantifies for the first time the effects of such configuration options. Following this setup in a practical setting would mean that, for a reconfiguration or a scenario with existing performance constraints (e.g., with given service-level agreements), we need actual energy measurements that enables the stakeholder to rate the influence of individual options.

Code-level energy consumption: When debugging or improving energy-intensive functions, developers need to trace energy consumption to concrete functions at the code level. RQ_3 provides some recommendations for this scenario. As shown in Table V, for most functions, a single proxy value or a configuration-option-specific proxy can be found with manageable effort, combining fine-grained energy measurements with performance profiling data. The important takeaway message is that, for the majority of functions, a single energy measurement is enough because of the constant energy–performance transfer factor of those functions. However, some functions' energy consumption need to be measured repeatedly, because we could not determine option-specific proxy values for these. Overall, our energy–performance correlation analysis has shown that runtime performance measurements can serve as a solid proxy for energy consumption and may be used in CI pipelines for automated regression detection.

E. Threats to Validity

Selecting Pearson's correlation measure imposes a threat to construct validity since other metrics such as rank-based correlation measures might result in other correlation values. Since measurement noise may influence both runtime performance and energy consumption in different ways, even small changes in runtime performance and energy consumption can substantially influence the ranking. Thus, comparing the ranks of runtime performance and energy consumption using rank-based correlation measures is more error-prone, such that we resort to the more robust correlation measure.

To increase external validity, we selected 14 configurable software systems from different domains. These ranges from throughput-intensive applications (e.g., compression tools, MP3 encoder, code optimizer) to server applications (Web server, databases). The configuration spaces contain 2 to 16 configuration options, some of which are numeric configuration options. Although this setup produces the largest configuration-focused data set for runtime performance and energy consumption we are aware of, there is no guarantee that our results generalize to other systems or application domains. However, since we already see a diverse picture in our data, we argue that our results hold also for other real-world applications.

There are also possible threats to validity regarding the literature analysis and the selected sampling strategies, which we discuss in Section III-B and Section IV-C.

VI. Conclusion

Understanding the relation between energy consumption and runtime performance of configurable software systems provides insights into whether runtime performance (which is easy to measure) can act as a proxy measure for energy consumption (which is more difficult to measure). An analysis of the literature draws a mixed picture with even shared causes for positive, negative, and no correlation between the two measures. We found that configurability has not been studied so far, possibly explaining some of the diverging observations.

We measured energy consumption and runtime performance of 14 real-world software systems, investing 2.5 years of CPU time, thereby building the largest set of combined energy-performance measurements we are aware of. We found that the correlation between energy consumption and runtime performance indeed depends on individual configuration options and interactions thereof: Despite an observed strong positive correlation over the whole configuration space, when confined to a subset of configurations, the correlation, and with it, the suitability of a performance proxy for energy consumption, can break down. Possible reasons are Simpson's Paradox and the presence of few, but influential options that dominate the correlation. Applying our findings in practical settings would mean that, in a greenfield scenario, runtime performance can be used as a reasonable proxy for energy consumption. In contrast, in a reconfiguration scenario, configurations with a similar performance can exhibit a negative correlating energy behavior, so we may need to determine the energy influence of individual configuration options to reliably forecast energy consumption of the changed configurations. By tracing the effects of configuration options on the correlation between energy consumption and runtime performance down to the function level, we are able to identify the root cause for configuration-dependent energy hot spots. This might be exactly the cost-effective tool needed for energy-aware software. However, identifying suitable methods and proxy values is not trivial and may require some upfront investment. Our insights indicate that these investments should come from an focused option-level sampling process.

Acknowledgments

Apel's and Siegmund's work has been funded by the German Research Foundation (SI 2171/3-1, SI 2171/2-2, AP 206/11-1, AP 206/11-2, and Grant 389792660 as part of TRR 248 – CPEC). Siegmund's work has been supported by the Federal Ministry of Education and Research of Germany for the "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig" (ScaDS.AI).

References

[1] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Modern Deep Learning Research," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09. AAAI Press, 2020, pp. 13 693–13 696.

[2] E. Kan, W. K. Chan, and T. Tse, "EClass: An Execution Classification Approach to Improving the Energy-Efficiency of Software via Machine Learning," *Journal of Systems and Software*, vol. 85, no. 4, pp. 960–973, 2012.

[3] J. Mendonça, E. Andrade, and R. Lima, "Assessing Mobile Applications Performance and Energy Consumption Through Experiments and Stochastic Models," *Computing*, vol. 101, no. 12, pp. 1789–1811, 2019.

[4] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, You Have Given Me Too Many Knobs!: Understanding and Dealing with Over-Designed Configuration in System Software," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 307–319.

[5] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-Influence Models for Highly Configurable Systems," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2015, pp. 284–294.

[6] J. Oh, D. Batory, M. Myers, and N. Siegmund, "Finding Near-Optimal Configurations in Product Lines by Random Sampling," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2017, pp. 61–71.

[7] P. Jamshidi and G. Casale, "An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2016, pp. 39–48.

[8] L. Passos, R. Queiroz, M. Mukelabai, T. Berger, S. Apel, K. Czarnecki, and J. Padilla, "A Study of Feature Scattering in the Linux Kernel," *IEEE Transactions on Software Engineering*, 2018.

[9] H. Ha and H. Zhang, "DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE / ACM, 2019, pp. 1095–1106.

[10] C. Kaltenecker, A. Grebhahn, N. Siegmund, and S. Apel, "The Interplay of Sampling and Machine Learning for Software Performance Prediction," *IEEE Software*, vol. 37, no. 4, 2020.

[11] M. Velez, P. Jamshidi, N. Siegmund, S. Apel, and C. Kästner, "White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE / ACM, 2021, pp. 1072–1084.

[12] M. Weber, S. Apel, and N. Siegmund, "White-Box Performance-Influence Models: A Profiling and Learning Approach," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE / ACM, 2021, pp. 232–233.

[13] M. Dong and L. Zhong, "Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, 2011, pp. 335–348.

[14] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, "Energy metering for free: Augmenting switching regulators for real-time monitoring," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2008, pp. 283–294.

[15] D. Bedard, M. Lim, R. Fowler, and A. Porterfield, "Powermon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems," in *Proceedings of the IEEE SoutheastCon (SoutheastCon)*. IEEE, 2010, pp. 479–484.

[16] A. Hindle, A. Wilson, K. Rasmussen, J. Barlow, J. Campbell, and S. Romansky, "Greenminer: A Hardware Based Mining Software Repositories Software Energy Consumption Framework," in *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 12–21.

[17] P. Pramanik, N. Sinhababu, B. Mukherjee, S. Padmanaban, A. Maity, B. Upadhyaya, J. Holm-Nielsen, and P. Choudhury, "Power Consumption Analysis, Measurement, Management, and Issues: A State-of-the-Art Review of Smartphone Battery and Energy Usage," *IEEE Access*, vol. 7, pp. 182 113–182 172, 2019.

[18] M. Gamell, I. Rodero, M. Parashar, and S. Poole, "Exploring Energy and Performance Behaviors of Data-Intensive Scientific Workflows on Systems with Deep Memory Hierarchies," in *Proceedings of the Annual International Conference on High Performance Computing (HiPC)*. IEEE, 2013, pp. 226–235.

[19] Z. Zhou, J. Abawajy, F. Li, Z. Hu, M. Chowdhury, A. Alelaiwi, and K. Li, "Fine-Grained Energy Consumption Model of Servers Based on Task Characteristics in Cloud Data Center," *IEEE Access*, vol. 6, pp. 27 080–27 090, 2018.

[20] B. Bruce, J. Petke, and M. Harman, "Reducing Energy Consumption Using Genetic Improvement," in *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 2015, pp. 1327–1334.

[21] L. Cruz and R. Abreu, "Performance-Based Guidelines for Energy Efficient Mobile Applications," in *Proceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2017, pp. 46–57.

[22] A. Shye, B. Scholbrock, and G. Memik, "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE / ACM, 2009, pp. 168–178.

[23] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. Fernandes, and J. Saraiva, "Energy Efficiency Across Programming Languages: How do Energy, Time, and Memory Relate?" in *Proceedings of the ACM SIGPLAN International Conference on Software Language Engineering (SLE)*. ACM, 2017, pp. 256–267.

[24] M. Puzović, S. Manne, S. GalOn, and M. Ono, "Quantifying Energy Use in Dense Shared Memory HPC Node," in *Proceedings of the International Workshop on Energy Efficient Supercomputing (E2SC)*. IEEE, 2016, pp. 16–23.

[25] G. Pinto, F. Castor, and Y. Liu, "Understanding Energy Behaviors of Thread Management Constructs," in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, 2014, pp. 345–360.

[26] D.-K. Kang, G.-B. Choi, S.-H. Kim, I.-S. Hwang, and C.-H. Youn, "Workload-Aware Resource Management for Energy Efficient Heterogeneous Docker Containers," in *Proceedings of the IEEE Region 10 Conference (TENCON)*. IEEE, 2016, pp. 2428–2431.

[27] M. Felderer and G. Travassos, *Contemporary Empirical Methods in Software Engineering*. Springer, 2020.

[28] B. Kitchenham and P. Brereton, "A Systematic Review of Systematic Review Process Research in Software Engineering," *Information and software technology*, vol. 55, no. 12, pp. 2049–2075, 2013.

[29] H. Yang, Q. Zhao, Z. Luan, and D. Qian, "iMeter: An Integrated VM Power Model Based on Performance Profiling," *Future Generation Computer Systems (FGCS)*, vol. 36, pp. 267–286, 2014.

[30] M. Rashid, L. Ardito, and M. Torchiano, "Energy Consumption Analysis of Algorithms Implementations," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE / ACM, 2015, pp. 1–4.

[31] Y. Lyu, D. Li, and W. Halfond, "Remove Rats from Your Code: Automated Optimization of Resource Inefficient Database Writes for Mobile Applications," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2018, pp. 310–321.

[32] W. Baek and T. Chilimbi, "Green: A Framework for Supporting Energy-Conscious Programming Using Controlled Approximation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2010, pp. 198–209.

[33] B. Subramaniam and W.-C. Feng, "GBench: Benchmarking Methodology for Evaluating the Energy Efficiency of Supercomputers," *Computer Science-Research and Development*, vol. 28, no. 2-3, pp. 221–230, 2013.

[34] G. Agosta, M. Bessi, E. Capra, and C. Francalanci, "Dynamic Memoization for Energy Efficiency in Financial Applications," in *Proceedings of the International Green Computing Conference and Workshops (IGCC)*. IEEE, 2011, pp. 1–8.

[35] T. Rauber, G. Rünger, and M. Stachowski, "Performance and Energy Metrics for Multi-Threaded Applications on DVFS Processors," *Sustainable Computing: Informatics and Systems*, vol. 17, pp. 55–68, 2018.

[36] M. Kambadur and M. Kim, "An Experimental Survey of Energy Management Across the Stack," in *Proceedings of the ACM International*

Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA). ACM, 2014, pp. 329–344.

[37] H. Anwar, B. Demirer, D. Pfahl, and S. Srirama, "Should Energy Consumption Influence the Choice of Android Third-Party HTTP Libraries?" in *Proceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE / ACM, 2020, pp. 87–97.

[38] W. Oliveira, R. Oliveira, and F. Castor, "A Study on the Energy Consumption of Android App Development Approaches," in *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 42–52.

[39] R. Pereira, M. Couto, J. Cunha, J. Fernandes, and J. Saraiva, "The Influence of the Java Collection Framework on Overall Energy Consumption," in *Proceedings of the IEEE/ACM International Workshop on Green and Sustainable Software (GREENS)*. IEEE / ACM, 2016, pp. 15–21.

[40] J. Michanan, R. Dewri, and M. Rutherford, "GreenC5: An Adaptive, Energy-Aware Collection for Green Software Development," *Sustainable Computing: Informatics and Systems*, vol. 13, pp. 42–60, 2017.

[41] E. Calore, A. Gabbana, S. Schifano, and R. Tripiccione, "Software and DVFS Tuning for Performance and Energy-Efficiency on Intel KNL Processors," *Journal of Low Power Electronics and Applications*, vol. 8, no. 2, 2018.

[42] I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta, and K. Soltany, "Evaluating the Impact of Caching on the Energy Consumption and Performance of Progressive Web Apps," in *Proceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. ACM, 2020, pp. 109–119.

[43] M. Nakhkash, T. Gia, I. Azimi, A. Anzanpour, A. Rahmani, and P. Liljeberg, "Analysis of Performance and Energy Consumption of Wearable Devices and Mobile Gateways in IOT Applications," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems (COINS)*. ACM, 2019, pp. 68–73.

[44] Q. Cai, J. González, G. Magklis, P. Chaparro, and A. González, "Thread Shuffling: Combining DVFS and Thread Migration to Reduce Energy Consumptions for Multi-Core Systems," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE / ACM, 2011, pp. 379–384.

[45] H. Malik, P. Zhao, and M. Godfrey, "Going green: An exploratory analysis of energy-related questions," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 418–421.

[46] S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: Techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–33, 2019.

[47] D. Bedard, R. Fowler, M. Lim, and A. Porterfield, "PowerMon 2: Fine-grained, Integrated Power Measurement," *RENCI Technical Report Series*, pp. 1–15, 2009.

[48] M. Velez, P. Jamshidi, F. Sattler, N. Siegmund, S. Apel, and C. Kästner, "Configcrusher: Towards White-Box Performance Analysis for Configurable Systems," *Automated Software Engineering*, vol. 27, no. 3, pp. 265–300, 2020.

[49] Y. Shu, Y. Sui, H. Zhang, and G. Xu, "Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.

[50] C. Blyth, "On Simpson's Paradox and the Sure-Thing Principle," *Journal of the American Statistical Association*, vol. 67, no. 338, pp. 364–366, 1972.

[51] S. Kolesnikov, N. Siegmund, C. Kästner, A. Grebhahn, and S. Apel, "Tradeoffs in Modeling Performance of Highly Configurable Software Systems," *Software & Systems Modeling (SoSym)*, pp. 2265–2283, 2019.

[52] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-Aware Performance Prediction: A Statistical Learning Approach," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE / ACM, 2013, pp. 301–311.

[53] J. Hartigan and P. Hartigan, "The Dip Test of Unimodality," *The Annals of Statistics*, pp. 70–84, 1985.

[54] S. Kokoska and D. Zwillinger, *CRC Standard Probability and Statistics Tables and Formulae*. CRC Press, 2000.