Green Configuration: Can AI Help Reduce Energy Consumption of Configurable Software Systems?

Norbert Siegmund
Leipzig University

Johannes Dorn
Leipzig University

Max Weber
Leipzig University

Christian Kaltenecker
Saarland University, Saarland Informatics Campus

Sven Apel
Saarland University, Saarland Informatics Campus

***Abstract*—Reducing energy consumption of IT-systems is fundamentally important for saving cost and reducing CO_2 emissions. A largely untapped potential arises from the configuration options a software system provides to adapt it to the application scenario, workload, and underlying hardware. As applying methods from artificial intelligence (AI) and machine learning (ML) has been a success story for performance optimization, it is tempting to expect similar benefits for energy.**

We review proposed and potential techniques from AI for reducing energy consumption and discuss why energy, unlike performance, requires an approach that is closely intertwined with other software-engineering (SE) methods. We explain the limits of pure AI/ML methods when focusing on the source code and outline a conceptual framework for combining SE methods and ML to build white-box energy models. This way, researchers and practitioners are guided towards promising techniques, including explicit modelling of uncertainty of energy estimates and identification of causal relationships of configuration options to energy leaks.

REDUCING ENERGY CONSUMPTION of IT systems is of paramount importance for our economy and society as it reduces CO_2 emissions, saves energy cost, and often comes also with performance improvements. A significant optimization potential for reducing energy consumption that has not been fully tapped arises from *software configurability*. Almost all non-trivial software systems today are configurable, including operating systems, database systems, video encoders, compression and data-science libraries. Typically, these systems provide a multitude of tuning knobs (configuration options or

Published by the IEEE Computer Society
© IEEE

parameters) to tailor their functional and non-functional behavior. From a user perspective, one can use configuration options to adapt the system to its hardware platform, workload, and usage scenario, enabling potentially huge energy savings, as has been confirmed already in several studies [1]. From a developer perspective, developers of software systems are more and more concerned with the energy consumption of different parts of the system [2]. A key challenge for energy-efficient configuration concluded in a recent literature study is: "(i) to locate portions of source code that can be optimized [developer perspective] and (ii) choose suitable parameters and configurations to reduce energy consumption [user perspective]" [3]. Empirical studies have found that 59 % of performance issues are related to configuration errors, 88 % of these issues require fixing the code [4].

In a long-term endeavor, we have investigated how methods from subsymbolic artificial intelligence (AI), specifically machine learning (ML), can help users find energy-optimized configuration and developers identify configuration-related code regions for energy debugging. In this article, we discuss different kinds of ML techniques and how they can be applied to different use cases in energy optimization of configurable software systems. We highlight the need for techniques that explicitly address the inherent uncertainty of energy measurement and modelling. We argue that ML alone is insufficient answering the questions of developers, especially when causal relationships need to be revealed (e.g., energy leaks caused by specific configuration options). Only when multiple information sources are combined, including program analysis, benchmark design, and system modelling techniques, we can provide accurate energy estimates of software configurations and point developers to configuration-dependent code regions of causal interest.

An important lesson that we learned and want to share is that special measures need to be taken into account when dealing with energy consumption of software. Simply transferring folklore wisdom from performance engineering is not sufficient. Energy consumption is a viscous property. It cannot be instantaneously traced from a code statement to a power draw. It is prone to multiple indirections, depending on the hardware, operating system, and the environment. In general, energy consumption is difficult to measure: The measurement must be either conducted at the level of individual hardware components, which imposes challenges of synchronization, or for the system as a whole, which is often inaccurate and imprecise. All these aspects have severe consequences on the choice of the ML technique, on the applicability of ML in different use cases, and on the need of additional software-engineering (SE) methods to be combined with ML. Here, we focus on the software side, keeping other influential factors, such as the operating system, hardware components, and workloads controlled. However, as we will show later, techniques from transfer learning are in reach to transfer energy models to a different context.

To summarize, we provide:

- An overview of different ML techniques for estimating energy consumption for different use cases in the area of configurable software systems (user and developer perspective);
- Results of applying a range of learning algorithms demonstrating the need to handle uncertainty and causality;
- Open research challenges for which ML alone is not able to solve the problems of software engineers in reducing energy consumption.

What is already possible with ML?

As said previously, we can approach energy reduction of software systems from two perspectives: (i) what the user can gain from an optimal system configuration, and (ii) what the developer can achieve with an energy-aware development of configurable code. Figure 1 illustrates the two perspectives. To provide quantitative statements about the configuration space, such as "What is the most energy-efficient configuration?", require an accurate estimate of a configuration's energy consumption. Here, pure methods from ML, which take a black-box perspective, are sufficient and cost-effective. However, to provide qualitative statements, we need to know which code regions are affected by configuration options and how strong the effect is with respect to energy consumption. Finding the configuration-related code regions requires advanced methods from software engineering, such as static and

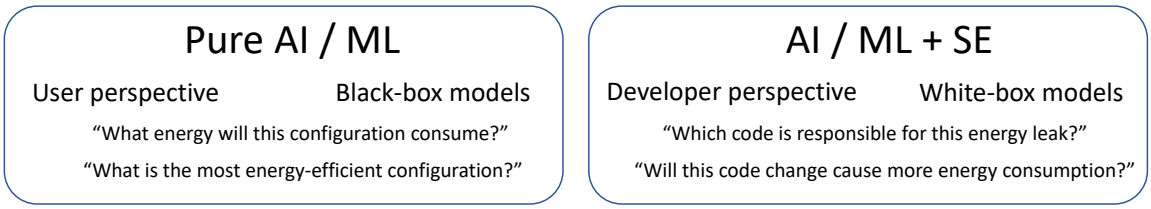

Figure 1. Different perspectives on configuration lead to different questions. Quantitative statements are best given by black-box models learned with pure ML. Qualitative statements require inner knowledge of the system, ultimately combining software engineering and ML methods to provide white-box energy models.

dynamic data-flow analysis. Only once we know the relevant code regions, we can again resort to ML to learn energy models for the individual regions. Only then it is possible to answer questions like “Where in the code and under which configuration do I lose energy?”. In what follows, we go through the two perspectives in more detail.

User Perspective: ML for Energy Optimization

Many software systems, libraries, and frameworks offer a wide range of configuration options, enabling the user to improve the system’s efficiency towards performance and energy consumption. However, the often large number of configuration options (ranging from dozens to thousands) tends to be mostly ignored by users, leaving significant optimization potential untapped [5]. The main issue is that users typically are not aware of the implications of configuration options on energy consumption or what the best configuration to minimize energy consumption is. The main reason is that, from the user perspective, the software system is a black box. Fortunately, ML can come to the rescue.

Accuracy is the key:
Estimating energy consumption of software configurations and finding optimal configurations have been extensively studied in recent years. It has been shown that the most accurate models are classification and regression trees, random forests (as ensemble method), and neural networks [6], [7]. The rationale behind these methods is that energy consumption behaves non-monotonically with respect to the configuration space. That is, small changes in configurations can have an abrupt change in energy consumption. Formally, a configuration is a set of assignments to all

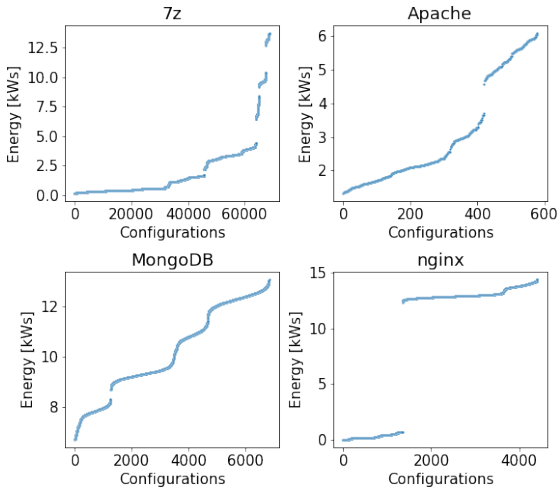

Figure 2. Energy stairs observed when ordering the configurations of four software systems according to their energy consumption. Similar stairs have been identified for performance earlier [8].

available configuration options from a certain domain (e.g., binary or numeric), that is $c = \{o_1, o_2, \ldots, o_n\}$, where n is the number of options and o_i is the value assigned to the i-th option. Figure 2 illustrates the non-monotonic energy consumption behavior for four real-world software systems: 7Z (file compression), APACHE (Web server), MONGODB (database system), and NGINX (reverse proxy). Energy has been measured using a power distribution unit. When ordering software configurations according their energy consumption, we observe several stairs. Such stairs have been found earlier for performance by Oh et al. [8]. They explain why learning techniques that mathematically support step functions are able to model energy consumption better than techniques that rely on smooth linear functions.

Having a model that can estimate the energy

consumption of every individual configuration is the first step towards optimization. Such a model allows a user to ask whether a configuration is 'better' than another and to what extent. However, to identify an optimal configuration or one that is close, we can either use the stairs to guide an iterative optimization process [8] or employ computationally expensive optimization methods, such as genetic optimization [9], on top of these (surrogate) models. Although such methods can lead to configurations with estimated low energy consumption, the correctness of this estimate is unclear. In other words, while we may be correct for 90 % of the estimates, we may be totally off in the remaining ones, causing a dilemma for the user: a switch to an supposedly better configuration from an already good configuration comes with the risk of being totally wrong. As these methods provide no means of risk assessment and no measure of *uncertainty* for their estimates, recent research has turned to the Bayesian world [10], [11], [12].

Uncertainty is the key for decision making:
A Bayesian model provides a measure of uncertainty along with an estimate. Applied to performance, Bayesian approaches are able to model uncertainty as first class citizen, paving the way to uncertainty-aware optimization [10]. They provide guarantees, for instance, to self-adapt to a faster configuration [12]. Applied to energy consumption, we developed the tool P4 (*p*erformance *p*rediction via *p*robabilistic *p*rogramming) to model an option's influence on energy consumption (and performance) as a probability density function describing the probability of an option having a specific influence on the system's energy consumption [11]. A configuration's energy consumption is, therefore, the combination of all probability density functions of selected configuration options, and itself a (joint) probability density function.

Figure 3 shows an example of how P4 models the energy consumption behavior of the Apache Web server. With such a model, it is now possible to obtain confidence intervals to provide probabilistic guarantees. For instance, we may be able to state that, with a confidence of 95 %, the energy consumption of a certain configuration is in an estimated range.

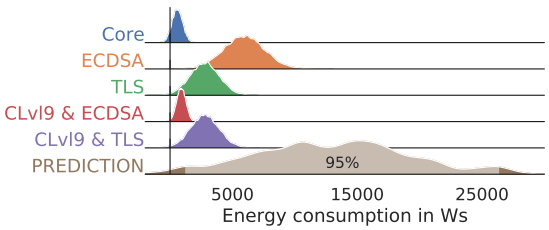

Figure 3. Illustration of P4's prediction for a given configuration of Apache. P4's prediction is shown on the bottom line. Its uncertainty distribution results from the uncertainty distributions of options and interactions that are active in the given configuration.

While Bayesian methods can be an effective tool for users when deploying an energy model alongside the software system, we have not considered the software itself yet. As we will show next, here is the point where AI/ML alone is insufficient to contribute enough information to developers to improve energy consumption.

Developer Perspective: ML for Energy Root-Cause Analysis

Software systems may contain bugs or may have inefficient implementations causing extraordinary energy consumption or energy leaks, especially in the context of configuration [4]. Even in the absence of energy leaks, the understanding of the configuration-related system's energy behavior (i.e., which code regions consume what energy under which configuration) is a necessary precondition for software maintenance and code optimization. This is where the black-box perspective reaches its limits. It is not possible to detect energy leaks by just estimating energy consumption at the level of configurations, since every model assumes that the same statistical properties hold for any configuration as seen for the sampled training set. In fact, a configuration-dependent bug is an outlier and would need to be explicitly observed (i.e., sampled and measured). Moreover, a black-box model cannot make any statements about where in the code an option triggers the consumption of energy and to what extent. As it turns out, this is where ML alone is insufficient, and new information sources from program analysis, system modelling, and benchmark design need to be brought in.

Intermezzo: White-box performance models: Before diving into white-box energy models, we will shortly review recent advances in the area of white-box performance modelling, since there are similar challenges: First, any white-box tool or analysis needs to establish a tracing from configuration options to code regions, in which the options take an effect either by directly controlling functionality and control flow or by indirectly influencing the data flow. Second, the corresponding code regions needs to be measured to be able to attribute the change in execution time to individual options and their interactions.

There are two angles to approach this task: solely with methods from SE and a combined approach. Following the pure SE approach, Velez and others employ static [13] and dynamic [14] control-flow and data-flow analysis to track code regions that depend on configuration options. They instrumented the identified code regions with precise time measurements construct a performance model without ML. The key here is that learning becomes obsolete if measurements are conducted in a highly precise and controlled way. Although very accurate, this approach comes with its own cost: Precise data-flow analysis is time- and resource-demanding and often requires the code to be prepared. That is, one may need to rewrite parts of the code, introduce measurement bias due to instrumentation, cannot account for further energy-influencing factors as there is no learning process, and faces an inherent measurement imprecision as energy is usually not immediately consumed, but indirectly via hardware. A pure SE approach might not be applicable for energy consumption. To address these limitations, we can add a pinch of ML.

The problem that needs to be solved is to avoid costly data-flow analysis and imprecise code instrumentation while still establishing a tracing from configuration options to code regions. As the first to combine both words for white-box performance models, our key idea is to apply sampling and ML with method-level profiling. Again, we execute the software system under different configurations. Now, we profile each method and record its execution time. With these data, we learn a performance model *per method* and can infer, this way, which options influence which methods by which amount [15].

Performance as a proxy for energy consumption? Having white-box performance models at hand, can we just use these models as a proxy for energy consumption? The argument is: the longer the execution, the more energy it consumes. In theory, method-level performance models may be a way to also build method-level energy models. However, this whole argument rests on the assumption that performance correlates with energy consumption. That is, performance can act only as a proxy if, for instance, a longer method execution time also results in a total higher energy consumption of that method. Unfortunately, this is not always true, since caching, different hardware usage, and energy drain due to high temperatures for high CPU load can easily break the correlation. In fact, when analyzing the existing literature for performance–energy correlation, we find papers with a positive correlation [16], a negative correlation [17], or no correlation at all [18]. Hao et al. even state in their analysis of 6 Android apps: “The Pearson coefficients (r in Table I) are nearly zero across all applications, indicating that there is almost no linear correlation between execution time and energy usage.” [18]. These inconsistent findings point to a research gap. How do programming mechanisms (e.g., data structures, architectures, data-flow patterns, etc.) influence the performance–energy correlation. In the case we find such influences, at least, for a subset of programming mechanisms, we can use techniques from *transfer learning* to convert white-box performance models to their energy counterparts. Jamshidi and others have already shown the practicality of transfer learning for black-box models when the context of a configurable software system changes (e.g., different hardware, workload, or version) [19]. Combining transfer learning with method-level performance-influence models [15] and a correlation analysis, white-box energy models might be possible. Moreover, extending a once learned white-box model to other hardware systems, environment settings, and workloads might be achievable like for performance once the changes in their distribution profiles and characteristics are known.

Putting these ideas together, a reasonable path for white-box energy models is to follow the recent advances in white-box performance modelling and combine them with energy measure-

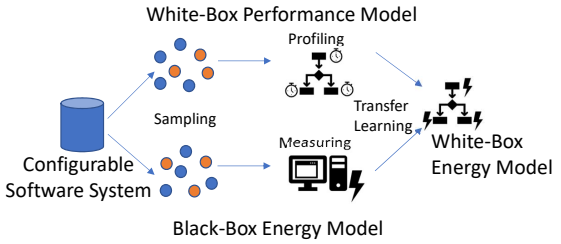

Figure 4. Combining SE methods and ML to enable white-box energy models.

ments to adjust and transfer the performance observations to approximate energy observations. Figure 4 outlines this path. It is important to note that all the necessary ingredients for this vision have been made available only recently. The upcoming research challenge is to put all pieces efficiently and reliably together and find effective ways to tune and validate the whole framework.

The Limit of AI: Energy Peculiarities

The path for achieving white-box energy models rests on rather strong assumptions, and it is unclear whether it works in most cases. The question is: Why can't we construct a white-box energy model directly with ML?

To answer this question, let us take a step back and review what is needed to build a white-box energy model. First, we need fine-grained energy measurements in space and time. That is, the measurements must be able to be traced back to individual code regions just like performance profiling or logging of individual methods. So, when entering a method or when sampling the method's call stack, we need to determine the current power drain and not just the drain of half a second ago. However, the frequency of sampling of the current power drain is below 100 Hz even for advanced measurement devices. Cost-intensive tailor-made solutions require specific hardware setups, such as in high-performance computing. Measurement lag makes locating the cause of energy consumption even at the method level difficult. Second, we need highly accurate power measurements within a milli-Watt range to build models at statement or method level. For performance, we can measure up to a nanosecond resolution or even count CPU cycles. For power drain, this resolution is often not available due to technical challenges. That is, there is an inherent uncertainty in the measurements we obtain. Third, we need a direct observable effect from code execution to power drain. However, energy might not be immediately consumed when executing a certain code statement. Often, there is a delay between a statement causing a certain power drain and when it becomes effective. This occurs, for instance, when electrical energy is transformed into heat due to heavy CPU load, or when hardware components act with a small time delay, or just due to caching effects. Naturally, this can get even worse when these effects overlap with measurement lag.

All these aspects clearly show that a direct application of ML methods will lead to inaccurate measurements, at best, or to misleading results, at worst. What we really need is to combine multiple methods from ML with knowledge about the software internals gained from software analyses. Specifically, it becomes clear that any approach to be successful must (i) explicitly account for uncertainty in the data and the measurement process, which points to Bayesian methods, and (ii) be able to locate the cause of an energy draining, which points to *causal modelling* [20].

Figure 5 exemplifies a causal model involving configuration options as independent variables activating the execution of different code paths, which cause the activation of hardware components and may interact with each other, all possibly causing a power drain. Obtaining such a model requires feature localization techniques using control-flow and data-flow analysis, from which we can obtain information what code region is affected by what (combination of) option(s). Moreover, we require an analysis of software patterns and architectures, since the usage of data structures, caching, and other resources have a profound influence not only on performance but also on energy consumption. Finally, being aware of the chain of causalities enables developers to ask questions of a new quality, such as *whether* changing my configuration causes an increased energy consumption and *why* is it so? This would enter a new stage of causal software configuration.

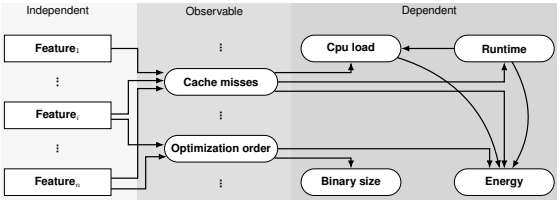

Figure 5. Causal graph showing energy consumption and performance.

Conclusion

We raised the question of whether methods from AI / ML can help reducing the energy consumption of configurable software systems. The current answer is that, only for cases of quantitative nature that require no deep insights, we can find more energy-efficient software configurations with ML techniques. As soon as we want to optimize the software itself, ML alone is insufficient. For this purpose, we need to establish traces from energy consumption behavior to individual code regions affected by configuration options, this way, being able to detect energy leaks and inefficient code. Establishing and maintaining traces requires fine-grained knowledge on where in the code specific configuration options are effective and how they influence energy consumption. Due to the low resolution and lag of energy measurements, as well as complex causal chains of energy consumption (e.g., software controls hardware which consumes energy), there are unsolved (possibly inherent) challenges to collect accurate data sets to be solely used by ML. Only when combining multiple sources of information, such as data-flow and control-flow analysis for building a white-box performance model and black-box energy models, ML methods can support developers. For these methods, reporting uncertainty of data and the model as well as uncovering causal chains are the key ingredients for obtaining insights on how to reduce energy consumption of a software system.

Acknowledgements

This work was supported by the German Research Foundation (SI 2171/2, SI 2171/3-1, and AP 206/11-1, as well as Grant 389792660 as part of TRR 248 – CPEC) and the German Federal Ministry of Education and Research (AgileAI: 01IS19059A and 01IS18026B) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".

REFERENCES

1. B. Herzog, F. Hügel, S. Reif, T. Hönig, and W. Schröder-Preikschat, "Automated selection of energy-efficient operating system configurations," in *Proc. Int. Conf. on Future Energy Systems*. ACM, 2021, pp. 309–315.
2. H. Malik, P. Zhao, and M. Godfrey, "Going green: An exploratory analysis of energy-related questions," in *Proc. Int. Working Conf. on Mining Software Repositories (MSR)*. IEEE, 2015, pp. 418–421.
3. S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: Techniques and tools," *ACM Comput. Surv.*, vol. 52, no. 4, 2019.
4. X. Han and T. Yu, "An empirical study on performance bugs for highly configurable software systems," in *Proc. Int. Symp. on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2016, pp. 1–10.
5. 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 *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 307–319.
6. H. Ha and H. Zhang, "DeepPerf: Performance prediction for configurable software with deep sparse neural network," in *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 2019, pp. 1095–1106.
7. 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, pp. 58–66, 2020.
8. J. Oh, D. Batory, M. Myers, and N. Siegmund, "Finding near-optimal configurations in product lines by random sampling," in *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 2017, pp. 61–71.
9. A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *Proc. Int. Conf. Automated Software Engineering (ASE)*. IEEE, 2013, pp. 465–474.
10. C. Trubiani and S. Apel, "PLUS: Performance learning for uncertainty of software," in *Proc. Int. Confón Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2019, pp. 77–80.
11. J. Dorn, S. Apel, and N. Siegmund, "Mastering Uncertainty in Performance Estimations of Configurable Software Systems," in *Proc. Int. Conf. Automated Software Engineering (ASE)*. IEEE, 2020, pp. 684–696.

12. C. Mandrioli and M. Maggio, “Testing self-adaptive software with probabilistic guarantees on performance metrics,” in *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 2020, pp. 1002–1014.
13. M. Velez, P. Jamshidi, F. Sattler, N. Siegmund, S. Apel, and C. Kästner, “ConfigCrusher: White-box performance analysis for configurable systems,” *Automated Software Engineering Journal*, vol. 27, pp. 265–300, 2020.
14. M. Velez, P. Jamshidi, N. Siegmund, S. Apel, and C. Kästner, “White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems,” in *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 2021, pp. 1072–1084.
15. M. Weber, S. Apel, and N. Siegmund, “White-Box Performance-Influence Models,” in *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 2021, pp. 232–233.
16. 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.
17. 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.
18. S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, “Estimating mobile application energy consumption using program analysis,” in *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 2013, pp. 92–101.
19. P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, “Transfer learning for performance modeling of configurable systems: An exploratory analysis,” in *Proc. Int. Conf. Automated Software Engineering (ASE)*, Oct. 2017, pp. 497–508.
20. A. Ibrahim and A. Pretschner, “From checking to inference: Actual causality computations as optimization problems,” in *Automated Technology for Verification and Analysis*. Springer, 2020, pp. 343–359.

Prof. Dr. Norbert Siegmund holds the Chair of Software Systems at Leipzig University, Germany. Prof. Sieg- mund received his PhD with distinction in 2012 from the University of Magdeburg. His research aims at the automation of software engineering by combining methods from software analysis and machine learning. His special interests include configurable software systems, performance and energy optimization, and SE4AI. Contact him at norbert.siegmund@uni-leipzig.de.

Johannes Dorn is a Ph.D. student and a research assistant at the Chair of Software Systems, Leipzig University, Leipzig, Germany. His research interests include uncertainty-aware performance-influence modeling and green computing. Dorn received his M.Sc. in 2018 from the Bauhaus University Weimar. Contact him at johannes.dorn@uni-leipzig.de.

Max Weber is a Ph.D. student and a research assistant at the Chair of Software Systems, Leipzig University, Leipzig, Germany. His research interests include energy consumption and performance modeling at method level of configurable software systems. Weber received his Master degree in 2018 from the Bauhaus University Weimar. Contact him at max.weber@informatik.uni-leipzig.de.

Christian Kaltenecker is a PhD student at the Chair of Software Engineering at Saarland University, Germany, under the supervision of Prof. Sven Apel. He received a Bachelor degree in 2014 and a Master degree in 2016 from the University of Passau, Germany. His research interests include sampling of software configuration spaces, performance prediction, and performance evolution of configurable software systems. One of his key contributions is the distance-based sampling strategy, which is an efficient and promising sampling strategy for sampling configurations of highly configurable software systems for performance prediction. Contact him at kaltenec@cs.uni-saarland.de.

Prof. Dr. Sven Apel holds the Chair of Software Engineering at Saarland University & Saarland Informatics Campus, Germany. Prof. Apel received his Ph.D. in Computer Science in 2007 from the University of Magdeburg. His research interests include software product lines, software analysis, optimization, and evolution, as well as empirical methods and the human factor in software engineering. Contact him at sven.apel@cs.uni-saarland.de.