Towards Feature-based ML-enabled Behaviour Location

Sophie Fortz
sophie.fortz@unamur.be
NADI, Fac. of Computer Science, University of Namur
Namur, Belgium

Paul Temple
paul.temple@irisa.fr
Univ. Rennes, CNRS, Inria, IRISA
Rennes, France

Xavier Devroey
xavier.devroey@unamur.be
NADI, Fac. of Computer Science, University of Namur
Namur, Belgium

Gilles Perrouin
gilles.perrouin@unamur.be
NADI, Fac. of Computer Science, University of Namur
Namur, Belgium

ABSTRACT

Mapping behaviours to the features they relate to is a prerequisite for variability-intensive systems (VIS) reverse engineering. Manually providing this whole mapping is labour-intensive. In black-box scenarios, only execution traces are available (*e.g.,* process mining). In our previous work, we successfully experimented with variant-based mapping using supervised machine learning (ML) to identify the variants responsible of the production of a given execution trace, and demonstrated that recurrent neural networks (RNNs) work well ($\geq$ 80% accuracy) when trained on datasets in which we label execution traces with variants. However, this mapping (i) may not scale to large VIS because of combinatorial explosion and (ii) makes the internal ML representation hard to understand. In this short paper, we discuss the design of a novel approach: feature-based mapping learning.

KEYWORDS

Variability Exploration, Feature, Machine Learning, Software Variability

ACM Reference Format:
Sophie Fortz, Paul Temple, Xavier Devroey, and Gilles Perrouin. 2024. Towards Feature-based ML-enabled Behaviour Location. In *18th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2024), February 7–9, 2024, Bern, Switzerland.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3634713.3634734

1 INTRODUCTION

Variability-intensive systems (VIS) constitute a unique category within the realm of software systems, distinguished by the ability to adapt their characteristics and behaviour through the activation or deactivation of specific features. These systems encompass a diverse range of applications, including Software Product Lines (SPLs) [1, 9], and business process families [10]. Yet, their behaviours are complex to analyse primarily due to combinatorial explosion and the absence of explicit variability-aware models [7]. In this context, locating variations is essential for any reverse-engineering endeavour [2]. Black-box testing techniques can also benefit from this information to *e.g.,* sample which variants should be tested first [7].

We previously proposed VaryMinions [4][1] to tackle the research question of identifying variant(s) that may have produced a given trace. VaryMinions relies on Recurrent Neural Networks (RNNs) mapping the occurrence of event messages to variants directly. Once the RNNs are trained, the learned mappings can be applied to new event logs. VaryMinions showed good performance, with a minimum of 80% accuracy in retrieving which variants could produce a specific event log.

Currently, to determine if a variant can generate a specific trace, we have to observe a substantial number of traces per variant to establish statistically significant patterns and draw confident conclusions. In practice, performing predictions across all potential variants requires enumerating and executing each variant, preferably multiple times. While this is doable for event logs associated with a limited number of variants [4], we may face combinatorial explosion while enumerating variants. Sampling is a common practice in such cases [7], though it may miss important configurations impacting the ML model confidence [12]. Mapping a trace to the entire configuration of a variant is quite demanding in terms of ML algorithms. We chose our Deep Learning (DL) models for their capacity to treat sequences. Yet, as in any DL model, interpretability is an issue, ultimately questioning the notion of learned features in the model's representation (latent space) [11]. Additionally, the cost of training and tuning these models is not negligible.

2 FROM VARIANTS TO FEATURES

To overcome these limitations, we shift our perspective from predicting which variants generate a specific execution trace to assessing the significance of (de)selecting particular features regarding that trace. We propose to independently predict the (de)selection of each feature, relying solely on the execution trace and the ground truth, represented as a vector of features, which can produce the trace. Each feature could be predicted using smaller and more interpretable ML models. But we should choose models that can output their confidence in their decision so that each feature can have three values: selected, deselected, or undetermined (*i.e.,* irrelevant in the current context).

Consequently, we need to train a set of small ML models, one for each feature. These models associate the presence of specific messages with the (de)selection of a given feature. As illustrated in the left segment of Figure 1, we have chosen Support Vector Machines (SVMs) as our ML models, although other options are worth

[1] the replication package is available at [5]

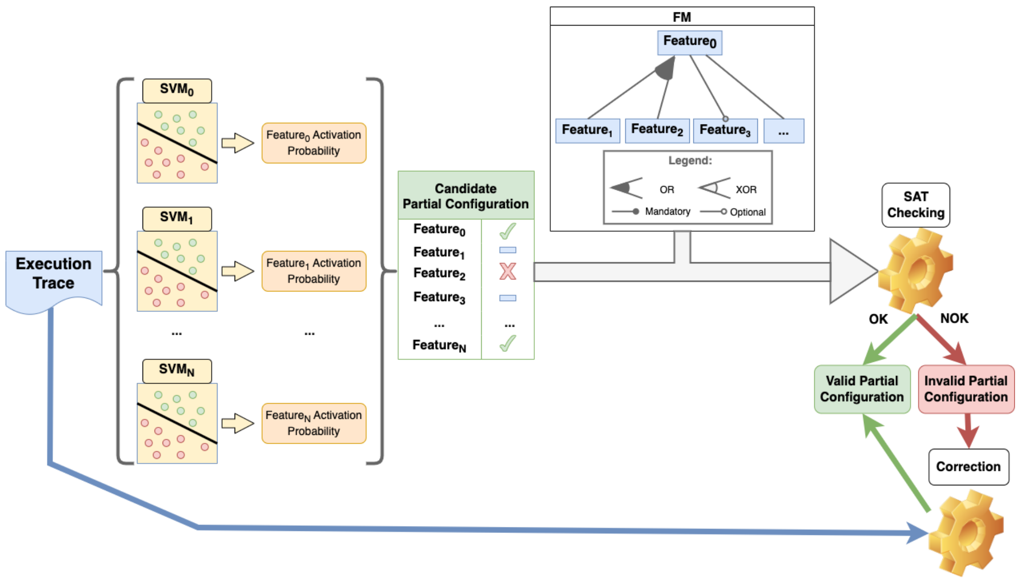

Figure 1: Conceptual representation of the feature-based approach during the prediction phase.

considering. For instance, the first SVM learns the (de)selection of the first feature, the second SVM handles the second one, *etc.*.

Once these models are trained, they can be applied to new execution traces. However, rather than providing a binary decision (selected or deselected), we intend to leverage the confidence level associated with the decision, represented as a floating value between -1 and 1, with 0 being the most uncertain case (*i.e.,* lying perfectly on the separation function). This more expressive approach justifies our choice of SVMs, as depicted in Figure 1. Based on this confidence score, we can establish thresholds for decision-making. A feature will be considered as selected if the confidence score falls between, for instance, 0.65 and 1, while it will be regarded as not selected if the confidence score ranges between -0.65 and -1. Any confidence score between -0.65 and 0.65 is deemed uncertain due to its proximity to the decision boundary, resulting in an "undetermined" decision. It's worth noting that the definition of these thresholds and their implications will require thorough evaluation.

The previous description, which includes the three possible outputs (selected, deselected, and "undetermined"), pertains to the prediction phase. Each model yields a confidence score for the (de)selection of its corresponding feature. Consequently, we obtain potential partial configurations (*i.e.,* when at least one feature is predicted as "undetermined"). In some scenarios, it may be beneficial to get a complete configuration, achieved by replacing all the "undetermined" with random (de)selected values. However, independent learning of (de)selection for features can introduce inconsistencies with the underlying feature model of the system, which is a problem that also arises when attempting to set values for "undetermined" features. Consequently, we need to use a solver to check that all constraints are satisfied. If any constraints are violated, we must develop strategies to rectify the configuration, such as through repair methods [3, 6, 8, 14]. As illustrated in the right part of Figure 1, this repair process should consider the input trace to ensure that the output of the repair procedure remains capable of executing the trace. Nonetheless, the initial output partial configuration provides the "form" (i.e., the selection and deselection of features) of a configuration, or set of configurations, having a high likelihood of generating the input execution traces. In case of errors or unexpected behaviour, we can swiftly focus on the most relevant features and their interactions.

3 CONCLUSION

In this paper, we highlighted the role of mapping choices when using machine learning to relate configurations with their execution trace. While our initial direct mapping performed well (*i.e.,* variant-based), it is challenging in terms of scalability, interpretability and computation. Learning a feature-to-trace mapping may alleviate these issues, but we must assess the proposed correction mechanism. We hypothesise that there is no perfect mapping and that we could eventually revisit the product line analysis cube [13] for ML tasks. Our long-term goal is to offer a detailed account of the trade-offs involved in the choices along the cube's edges and to help the community make informed mapping choices. Our first action will be the implementation of our approach. As mentioned before, we will have to face different choices such as which ML model to use or how to set the different thresholds to decide whether a feature is selected, deselected or if it does not matter. They will be considered as hyperparameters of the approach. We plan to evaluate our implementation on different subject systems to assess that our idea is applicable and whether it can surpass our previous work [4]. This implementation will help us understand how to set these hyperparameters so that we can provide guidelines.

ACKNOWLEDGMENTS

Sophie Fortz is supported by the Fonds de la recherche scientifique (FRS-FNRS) via a FRIA grant. Gilles Perrouin is an FRS-FNRS Research Associate. This research is partly supported by the EOS VeriLearn project under FRS-FNRS Grant No. O05518F-RG03, and the CyberExcellence project (No. 2110186) funded by the Public Service of Wallonia (SPW Recherche).

REFERENCES

[1] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation.* Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37521-7

[2] Wesley Klewerton Guez Assunção, Roberto Erick Lopez-Herrejon, Lukas Linsbauer, Silvia Regina Vergilio, and Alexander Egyed. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* 22 (2017), 2972–3016.

[3] Johanna Buchner and Marian Daun. 2016. Automated inconsistency detection and solution proposals in cyber-physical system networks. In *2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC).* IEEE, Manhattan, USA, 35–40. https://doi.org/10.1109/EITEC.2016.7503694

[4] Sophie Fortz, Paul Temple, Xavier Devroey, Patrick Heymans, and Gilles Perrouin. 2021. VaryMinions: leveraging RNNs to identify variants in event logs. In *Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution, Athens, Greece, 23 August 2021*, Apostolos Ampatzoglou, Daniel Feitosa, Gemma Catolino, and Valentina Lenarduzzi (Eds.). ACM, New York, NY, USA, 13–18. https://doi.org/10.1145/3472674.3473980

[5] Sophie Fortz, Paul Temple, Xavier Devroey, Patrick Heymans, and Gilles Perrouin. 2022. VaryMinions. https://zenodo.org/record/7492126. https://doi.org/10.5281/zenodo.7492126 Sophie Fortz is supported by the FNRS via a FRIA grant. Gilles Perrouin is an FNRS Research Associate..

[6] Patrick Franz, Thorsten Berger, Ibrahim Fayaz, Sarah Nadi, and Evgeny Groshev. 2021. Configfix: interactive configuration conflict resolution for the linux kernel. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* IEEE, Manhattan, USA, 91–100.

[7] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2019. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empir. Softw. Eng.* 24, 2 (2019), 674–717. https://doi.org/10.1007/s10664-018-9635-4

[8] Azadeh Jahanbanifar, Ferhat Khendek, and Maria Toeroe. 2021. Configuration of Complex Systems—Maintaining Consistency at Runtime. In *Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems: Communications of NII Shonan Meetings.* Springer, Berlin, Heidelberg, 199–224.

[9] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag, Berlin, Heidelberg.

[10] Marcello La Rosa, Wil MP Van Der Aalst, Marlon Dumas, and Fredrik P Milani. 2017. Business process variability modeling: A survey. *Comput. Surveys* 50, 1 (2017), 1–45. https://doi.org/10.1145/3041957

[11] Paul Temple and Gilles Perrouin. 2023. Explicit or Implicit? On Feature Engineering for ML-Based Variability-Intensive Systems. In *Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems* (Odense, Denmark) *(VaMoS '23).* Association for Computing Machinery, New York, NY, USA, 91–93. https://doi.org/10.1145/3571788.3571804

[12] Paul Temple, Gilles Perrouin, Mathieu Acher, Battista Biggio, Jean-Marc Jézéquel, and Fabio Roli. 2021. Empirical assessment of generating adversarial configurations for software product lines. *Empirical Software Engineering* 26, 1 (2021), 6. https://doi.org/10.1007/s10664-020-09915-7

[13] Alexander von Rhein, Sven Apel, Christian Kästner, Thomas Thüm, and Ina Schaefer. 2013. The PLA Model: On the Combination of Product-Line Analyses. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-Intensive Systems* (Pisa, Italy) *(VaMoS '13).* Association for Computing Machinery, New York, NY, USA, Article 14, 8 pages. https://doi.org/10.1145/2430502.2430522

[14] Yingfei Xiong, Hansheng Zhang, Arnaud Hubaux, Steven She, Jie Wang, and Krzysztof Czarnecki. 2014. Range fixes: Interactive error resolution for software configuration. *Ieee transactions on software engineering* 41, 6 (2014), 603–619.