# Transferring Performance across Software Systems Are we There yet?

Anonymous Author(s)

## ABSTRACT

Many research studies seek to predict the performance of software using machine learning techniques requiring large amounts of data. Transfer learning aims to reduce the amount of data needed to train these models. We can now apply transfer learning to different executing environments, software versions or variants, *etc.* We propose a minimal example of transfer learning across software systems using two video encoders (namely `x264` and `x265`) and discuss the open challenges to overcome for its general application.

**ACM Reference format:**
Anonymous Author(s). 2021. Transferring Performance across Software Systems Are we There yet?. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Software offers more and more options that users can (de)select to customize the system for their specific needs. With the exploding number of options *e.g.*, +2041 options in less than three years for the linux kernel [20], it becomes complex to accurately estimate the individual impact of options, difficult to predict software performance and unthinkable to measure exhaustively the configuration space of real-world systems. To overcome this problem, related work has proposed to train machine learning models *i.e.*, performance models [10], taking software configurations as input, training the model to link a configuration to a performance value and finally capable of predicting the performance of a new configuration. But this method has a cost [24]: since we feed the model with configurations and performance, we have to measure the performance of several configurations. Transfer learning techniques [20, 23] aim at reducing this measurement cost: if a performance model has already been trained on one executing environment, we can reuse this model to reduce the number of measures needed for a second executing environment [13], assuming that both environments are similar (*e.g.*, same operating system). We propose to apply transfer learning to software systems performing the same task, such as two different compilers (*e.g.*, `gcc` and `llvm`), two container managers (*e.g.*, `podman` and `docker`) or two video encoders (*e.g.*, `x264` and `x265`). Intuitively, the model trained on one software could be used -at least partially- to train the other performance model, as depicted in Figure 1 for `x264` and `x265`. This paper presents the first minimal example showing that under certain conditions, it is possible to transfer performance models between different software systems. We also discuss the limitations of our work and highlight the open challenges to face when scaling to other software systems.

Our contributions are as follows:

- An example of transfer learning across software systems;
- Data [1] and code [2] are publicly available.

[1]Dataset available at : https://zenodo.org/record/5662589
[2]Code available at : https://anonymous.4open.science/r/TL_cross_soft-855B/

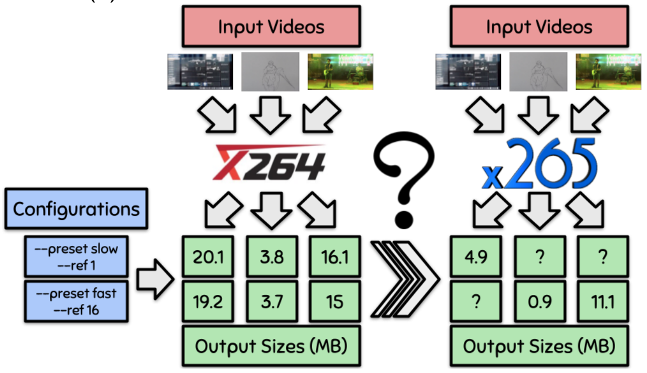


**Figure 1: Can x264 be used to predict x265 performance?**

Section 2 proposes a minimal example of transfer learning across systems; Section 3 details the threats to validity; Section 4 discusses our experiment and the requirements to transfer performance across software systems; Section 5 presents the related work; Section 6 concludes our paper.

## 2 TRANSFER LEARNING ACROSS SOFTWARE SYSTEMS: A PROOF OF CONCEPT

### 2.1 Research Questions

To get started with this topic, we first study the differences of performance between different software systems executing the same task. Are the distribution of performance consistent across similar software systems? If they are, we can probably use similarities between their performance distributions when training the model. We then address the main issue of paper : **Can we transfer performance from one software system to another?** How useful is it compared to non-transfer technique? How much do we gain, both in terms of measurements and accuracy? More specifically, we want to ensure that transfer learning does indeed outperform simple learning and avoid any instance of *negative transfer.*

To answer these questions, we design the following study.

### 2.2 Study Design

*2.2.1 Measure Performance.* First, we gather measurements about performance of software systems.

*Software Systems.* We select `x264` and `x265`,[3] two video encoders with different compression standards (resp. H.264 and H.265). Both are developed by VideoLAN, which makes it easier to find similarities between them (same options, same vision, *etc.*).

*Configuration Options.* Once the systems are selected, we search for common configuration options in their documentation. For instance, and according to their documentation, both `x264` and `x265` implement the features *−ref* and *−preset*. *−ref* could be set to 1, 8 or 16 while *−preset* can be fast or slow. Possible resulting

[3]See `x264` and `x265` webpages : https://www.videolan.org/developers/x264.html and https://www.videolan.org/developers/x265.html

configurations like (slow, 1) or (fast, 16) are accepted and valid for x264 and x265, as shown in Figure 1. In the end, we keep 35 configuration options common to x264 and x265 (out of resp. 118 and 168)).[4] We exploit these common features and make their values vary in order to generate a set of 3125 configurations working for both systems. We check the uniformity of the resulting distribution of options' values with a Kolmogorov-Smirnov test [21].[5]

*Input Data.* We select eight input videos extracted from the Youtube UGC Dataset [32], well-known in the community of video compression. For this selection, we vary the content (LiveMusic, Sports) and the resolution (360P, 480P) of videos.

*Performance Properties.* We then use x264 and x265 to transcode these eight videos from the mkv to the mp4 format. During each execution, we measure the percentage of cpu usage, etime the elapsed time in seconds and the size of the compressed video in bytes.

*Executing Environment.* We measure all performance sequentially on a dedicated (and warmed-up) server - model Intel(R) Xeon(R) CPU D-1520 @ 2.20GHz, running Ubuntu 20.04 LTS.

#### 2.2.2 *Compare Performance.*

Second, we show the differences between the distribution of performance properties of x264 and x265. As a measure of (dis)similarity, we display the Spearman rank-based correlation [15]. It is suited for our case since all performance properties are quantitative variables relative to the same configurations. If both encoders obtain the same rankings in terms of performance, the correlation is close to 1, and there is a good chance of getting good results with transfer learning. If they react differently to the same configurations, the correlation is close to 0 and the transfer might be challenging to achieve.

#### 2.2.3 *Transfer Performance.*

Third, and as displayed in Figure 1, we try to transfer the performance from x264 (*i.e.*, source software) to x265 (*i.e.*, target software). We use *Model Shift*, a transfer learning approach defined by Valov *et al.* [31]. The protocol should be read following Figure 2 :
1. First, it trains a shifting function, mapping the performance distribution of the source on the target software's performance distribution, 2. Then, it trains a performance model on the source software, 3. Finally, it predicts the performance distribution of the source software and applies the shifting function to the predictions, in order to estimate the performance of the target software system. We compare *Model Shift* to a simple baseline acting as a control approach, training a performance model directly on the target software, without using any measurement of the source. We call this baseline *No Transfer*. For both, we used a Random Forest algorithm [25] to predict software performance, without tuning its hyperparameters. We separate the dataset of the target into training and test, varying the size of the training set. In the evaluation, we compare and display the Mean Absolute Percentage Error (MAPE) [22] between the predicted values (*i.e.*, predicted by the approaches) and the real values (*i.e.*, measured on the test set of the target software). To reduce the variance induced by machine learning randomness, we repeated the process five times and display the average MAPE. We rely on the python library scikit-learn [27].[6]

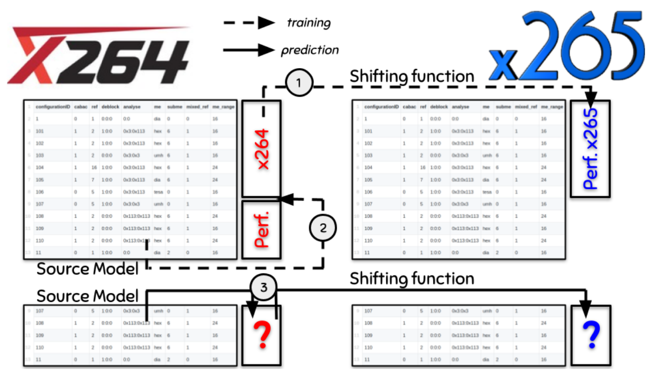


**Figure 2: *Model Shift*, an approach of transfer learning**

## 2.3 Evaluation

#### 2.3.1 *Compare Performance.*

Among the three performance properties, we can distinguish three cases : between x264 and x265, the cpu consumption has an average a correlation close to 0 (lower than 0.1), the elapsed time etime is overall positively correlated (about 0.75), and there is almost no differences between the sizes of the resulting mp4 videos (correlations close to 1). Transferring sizes is likely to be easy and transferring cpu consumption will probably be too difficult. The elapsed time etime is in between.

**Table 1: Correlations between the performance distributions of x264 and x265 for eight input videos**

| Video | cpu | etime | size |
|---|---|---|---|
| Animation | 0.05 | 0.74 | 0.98 |
| CoverSong | 0.0 | 0.73 | 0.98 |
| Gaming | -0.02 | 0.81 | 0.99 |
| Lecture | 0.07 | 0.75 | 0.98 |
| LiveMusic | 0.01 | 0.77 | 0.99 |
| LyricVideo | 0.14 | 0.73 | 0.96 |
| MusicVideo | 0.07 | 0.75 | 0.99 |
| Sports | -0.0 | 0.78 | 0.98 |

#### 2.3.2 *Transfer Performance.*

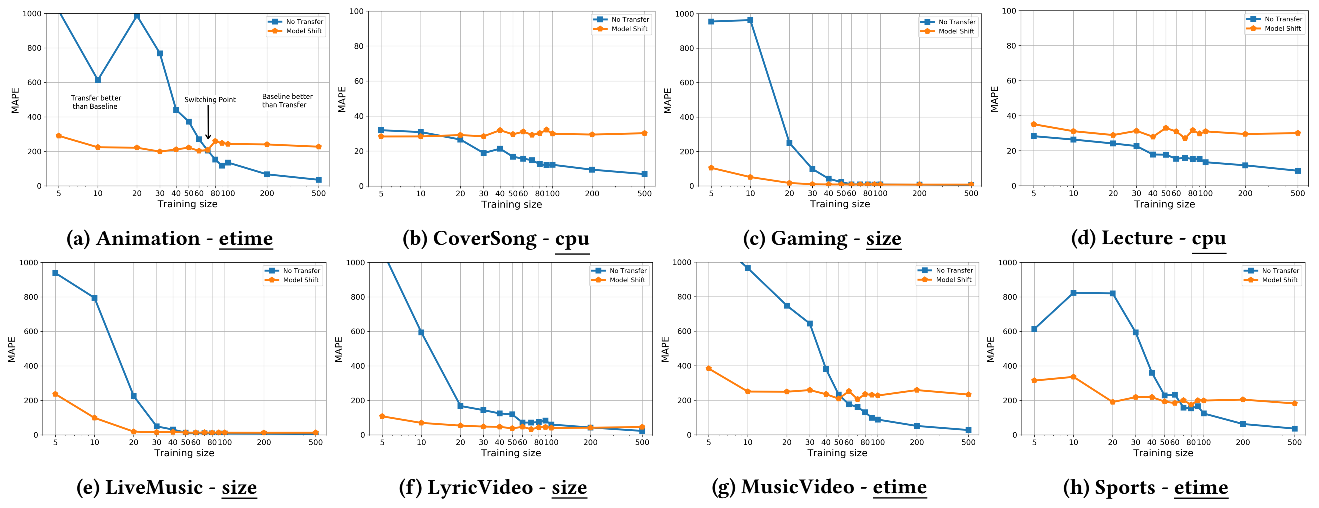
In Figure 3, we can notice that the results vary depending on performance properties:

- The transfer of cpu consumptions is almost always negative. For example, after 15 configurations in the training set for the CoverSong video, in Figure 3b, the baseline is always more accurate than transfer learning. For the Lecture video, in Figure 3d, it is even always a negative transfer.
- For etime, the transfer is cost-effective at first, until a given training size *e.g.*, 65 for Figure 3h. After this threshold, the error of the baseline keeps dropping while the errors stabilise for the transfer. For the Animation video, in Figure 3a, and for a budget of 500 configurations, *No Transfer* decreases to a MAPE of $35 \pm 1.3$ % and *Model Shift* stays at $220 \pm 21$ %.
- Finally, for the encoded sizes of videos, *Model Shift* is at least equivalent to the baseline whatever the number of configurations

[4]The list of selected configuration options can be consulted here : https://anonymous.4open.science/r/TL_cross_soft-855B/replication/x26x/README.md

[5]Results can be consulted at: https://anonymous.4open.science/r/TL_cross_soft-855B/replication/x26x/x264_x265_options.ipynb

[6]A description of our python environment can be consulted at: https://anonymous.4open.science/r/TL_cross_soft-855B/replication/requirements.txt



(a) Animation - etime (b) CoverSong - cpu (c) Gaming - size (d) Lecture - cpu

(e) LiveMusic - size (f) LyricVideo - size (g) MusicVideo - etime (h) Sports - etime

**Figure 3: Error (y-axis, lower is better) when transferring x264 to x265 performance depending on the training size (x-axis, log scale), for eight input videos and three performance properties**

and the video. This can be explained by the high correlations observed in Table 1. At first, the transfer is really outperforming the baseline *e.g.*, in Figure 3f, for 5 configurations, the transfer has an error of about 106 ± 64 % while the baseline amounts to 1071 ± 556 %.

**Conclusion.** The effectiveness of the transfer varies with the performance property we consider. Overall, it seems possible to outperform the *No Transfer* baseline when the budget (*i.e.*, the number of configuration in the training set) is low. As noticed by [12], the greater the correlation between performance distributions, the more accurate the transfer. As a concrete advice for developers; this correlation could be a cheap indicator to estimate *a priori* whether transfer techniques are adapted between two software systems.

## 3 THREATS TO VALIDITY

### 3.1 Internal

Due to the cost of measurement (58 days of system time), we did not measure performance more than once; the performance distribution of x264 and x265 could change with new measurements. To address this threat, we check the results for eight different input videos; given their consistency, we are confident that similar conclusions could be drawn by repeating the experiment. Another threat to validity is related to the use of machine learning algorithms, which leads to non-deterministic results. To mitigate this threat, we display the average result of five runs of the model.

### 3.2 External

In Section 2.2, we only select the configuration options common to x264 and x265, ignoring a large majority of features. This represents a potential threat to validity when generalising the transfer to the whole configuration space.

## 4 DISCUSSION

### 4.1 Limitations of our Work

Although this experiment is encouraging to explore this research direction further, it does not represent a realistic and operational case of transfer learning between software systems. The first limit to its generalisation is explained by the results of Section 2 : after a given training size, the interest in adding noisy measures -such as source data- decreases. This learning size is a switching point, as shown by Figure 3a: before this point, it is preferable to use transfer learning and after this point, we should switch to simple learning. This point seems difficult to estimate *a priori i.e.*, without any measurement. This may depend on the complexity of the configuration space: the more complex the configuration space, the more configurations are needed to accurately predict the software performance. And the more configurations needed to be accurate, the larger the switching point. It is therefore difficult to know when to apply transfer learning and when to switch to non-transfer learning. A second limitation is related to the performance properties. If a non-functional property is not consistent between software systems *e.g.*, with a correlation of zero, we believe it will perform poorly when using transfer learning. The third limitation is induced by the choice of software systems. We have deliberately simplified our example; since x264 and x265 are developed by the same team, the two systems share a common set of configuration options with the same range of values. Randomly selecting two systems from the same domain would make the experiment more complex to conduct.

### 4.2 Open Challenges

Specifically in this section, we place ourselves in a more general case. What might be the open challenges to overcome when transferring performance between software systems?

*4.2.1 Find Relevant Software Systems.* We cannot guarantee the effectiveness of transfer learning for every pair of software systems. What is difficult is to know whether transfer will work for a given pair of software systems. The only reasonable assumption we make is to choose software in the same domain: video encoders like `vp9`, `x264` or `x265`, compilers like `gcc`, `llvm` or `clang`, container managers like `podman` or `docker`, learning libraries like `theano`, `pytorch` or `tensorflow`, text editors like `emacs`, `gedit` or `vim`, *etc.*

#### 4.2.2 Choose the right Approach.

In our experiment, we are only testing one approach, namely *Model Shift*, but there are many other transfer learning techniques (see Section 5 - Transfer Learning). The challenge here is to determine the best approach to use for a pair of software systems. Empirical studies comparing different approaches on representative pairs of systems would be of great help in addressing this challenge.

#### 4.2.3 Map Configuration Spaces.

The configuration spaces will often be very different between the source and the target systems, making the sampling of configurations difficult in practice. We give several examples of such differences, illustrated with `vp9`, `x264`, and `x265`,[7] that should be dealt with :

- The same feature is implemented in the source and the target with two different names *e.g.*, *−level* for `x264` and *−level-idc* for `x265`;
- The same feature is implemented in the source and the target, but a value is only implemented in one software system *e.g.*, unlike for `x265`, the motion-estimator feature *−me* of `x264` does not implement the 'star' pattern search;
- The feature of one system encapsulates one feature (or more) of the other *e.g.*, activating *−fullrange* in `x264` is equivalent to choose *−range full* for `x265`;
- The feature is only implemented in one software system *e.g.*, *−rc-grain* in `x265` does not exist for `x264`;
- The feature does not have the same default value for the source and the target *e.g.*, *−qpmax* is set to 51 by default for `x264` and set to 69 by default for `x265`;
- Both software systems do not have the same requirements or feature interactions *e.g.*, for `x265`, passing a input video in the yuv format does not work unless you specify *−input-res*, while it does for `x264`;
- The same feature is implemented in the source and the target, but the scale of the values differ between the source and the target: the constant rate factor *−crf* goes from 0 to 63 for `vp9` and from to 0 to 51 for `x264` and `x265`, which is problematic when comparing a value (*e.g.*, *−crf* = 35) that will not have the same meaning for all systems. We could even imagine a situation where the values of a feature are increasing for one system and decreasing for the other.

Mapping configuration spaces is still challenging, but could be envisioned using recent advances in the variability community. We propose a very simple protocol below:

(1) *Align features* [19, 30] between systems;
(2) Meticulously model both configuration spaces *e.g.*, with feature models [5, 14];
(3) Analyse [8] and instrument [1, 2] them in order to create a resulting feature model generating configurations accepted by both systems.

In addition, and as acknowledged by [17], the study of configuration options also requires domain knowledge, which is also true when mapping the configuration spaces of different software systems. If the objective is purely to predict performance, heterogeneous transfer learning may be a possible black-box alternative (see Section 5) that infers the relationship between distinct configuration spaces.

[7]You can verify our illustrations with the lists of features:
https://cinelerra-gg.org/download/CinelerraGG_Manual/VP9_parameters.html for `vp9`,
http://www.chaneru.com/Roku/HLS/X264_Settings.htm for `x264`,
https://x265.readthedocs.io/en/2.5/cli.html for `x265`.

## 5 RELATED WORK

*Performance Comparison.* In many areas, related work compares the performance of different tools performing the same task [6, 9, 18, 26, 28, 29]. These empirical evidences could be relevant; if two software systems have similar performance distributions, they are good candidates for the transfer.

*Transfer Learning.* We list here few other transfer learning techniques that could be applied across software systems. Jamshidi *et al.* define Learning to Sample (L2S) [13]: it combines an exploitation of the source and an exploration on the target to sample a list of configurations, improving the overall accuracy of the learning algorithm. As many other transfer learning works [4, 23], it is applied to transfer performance of executing environments. Martin *et al.* develop TEAMs [20], a transfer learning approach predicting the performance distribution of the Linux kernel using the measurements of its previous releases. Between two releases, related the same system but distant in time, one can consider that it is a simple case of transfer across systems. Krishna *et al.* implement BEETLE [16], that we could use to find one bellwhether software *i.e.*, a source software that lead to better transfer results whatever the target. João *et al.* propose Weighted Multisource Tradaboost [3]. Applied to our case, it would exploit multiple source software systems and give them different weight values based on their similarity with the target system. This approach can be useful to apprehend the wide diversity of existing systems.

*Heterogeneous Transfer Learning.* Heterogeneous Transfer Learning [7, 11, 33] (HTL) is an extension of homogeneous transfer learning handling the differences between the source and the target feature spaces (*a.k.a.*, configuration spaces when studying software variability). It creates a representation of the feature space, in between the source and the target, and finally transforms both feature spaces so they fit in this representation. Applied to the transfer across software systems, it would handle the changes of features between the configuration space of the source software and the configuration space of the target software. Moving to HTL would generalise our work to a more realistic case, where we can decide to select any feature of the source and the target systems.

## 6 CONCLUSION

This short paper presents a minimal example showing that it is yet possible to transfer performance between different software systems, using two video encoders (namely `x264` and `x265`). We also discuss the limitations of our work and highlight the open challenges to overcome when generalising our work.

## REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. 2010. Composing Feature Models. In *Software Language Engineering*, Mark van den Brand, Dragan Gašević, and Jeff Gray (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 62–81.

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2011. Slicing feature models. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 424–427. https://doi.org/10.1109/ASE.2011.6100089

[3] João Antunes, Alexandre Bernardino, Asim Smailagic, and Daniel Siewiorek. 2019. Weighted Multisource Tradaboost. In *Pattern Recognition and Image Analysis*, Aythami Morales, Julian Fierrez, José Salvador Sánchez, and Bernardete Ribeiro (Eds.). Springer International Publishing, Cham, 194–205.

[4] Joaquín Ballesteros and Lidia Fuentes. 2021. *Transfer Learning for Multiobjective Optimization Algorithms Supporting Dynamic Software Product Lines*. Association for Computing Machinery, New York, NY, USA, 51–59. https://doi.org/10.1145/3461002.3473944

[5] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering*, Oscar Pastor and João Falcão e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 491–503.

[6] Nataliya Boyko, Oleg Basystiuk, and Nataliya Shakhovska. 2018. Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library. In *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*. 478–482. https://doi.org/10.1109/DSMP.2018.8478556

[7] Oscar Day and Taghi M Khoshgoftaar. 2017. A survey on heterogeneous transfer learning. *Journal of Big Data* 4, 1 (2017), 1–42.

[8] Roberto Di Cosmo and Stefano Zacchiroli. 2010. Feature Diagrams as Package Dependencies. In *Software Product Lines: Going Beyond*, Jan Bosch and Jaejoon Lee (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 476–480.

[9] Rasmus Emilsson. 2020. Container performance benchmark between Docker, LXD, Podman &amp; Buildah. , 23 pages.

[10] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 301–311. https://doi.org/10.1109/ASE.2013.6693089

[11] Yuwei He, Xiaoming Jin, Guiguang Ding, Yuchen Guo, Jungong Han, Jiyong Zhang, and Sicheng Zhao. 2020. Heterogeneous transfer learning with weighted instance-correspondence data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4099–4106.

[12] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering* (Urbana-Champaign, IL, USA) *(ASE 2017)*. 497–508.

[13] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to sample: exploiting similarities across environments to learn performance models for configurable systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 71–82. https://dl.acm.org/doi/pdf/10.1145/3236024.3236074

[14] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. Feature-Oriented Domain Analysis (FODA) feasibility study. (01 1990).

[15] Maurice George Kendall. 1948. *Rank correlation methods*.

[16] Rahul Krishna, Vivek Nair, Pooyan Jamshidi, and Tim Menzies. 2020. Whence to Learn? Transferring Knowledge in Configurable Systems using BEETLE. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.2983927

[17] Jacob Krüger, Wanzi Gu, Hui Shen, Mukelabai Mukelabai, Regina Hebig, and Thorsten Berger. 2018. Towards a Better Understanding of Software Features and Their Characteristics: A Case Study of Marlin. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems* (Madrid, Spain) *(VAMOS 2018)*. Association for Computing Machinery, New York, NY, USA, 105–112. https://doi.org/10.1145/3168365.3168371

[18] Yue Li, Zhuo Zhang, Feng Liu, Wanwipa Vongsangnak, Qing Jing, and Bairong Shen. 2012. Performance comparison and evaluation of software tools for microRNA deep-sequencing data analysis. *Nucleic Acids Research* 40, 10 (01 2012), 4298–4305. https://doi.org/10.1093/nar/gks043 arXiv:https://academic.oup.com/nar/article-pdf/40/10/4298/25335127/gks043.pdf

[19] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2016. Variability extraction and modeling for product variants. *Software & Systems Modeling* 16, 4 (Jan. 2016), 1179–1199. https://doi.org/10.1007/s10270-015-0512-y

[20] Hugo Martin, Mathieu Acher, Luc Lesoil, Jean Marc Jezequel, Djamel Eddine Khelladi, and Juliana Alves Pereira. 2021. Transfer Learning Across Variants and Versions : The Case of Linux Kernel Size. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3116768

[21] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951), 68–78.

[22] Christoph Molnar. 2020. *Interpretable Machine Learning*. Lulu. com.

[23] Farnaz Moradi, Rolf Stadler, and Andreas Johnsson. 2019. Performance Prediction in Dynamic Clouds using Transfer Learning. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 242–250.

[24] Vivek Nair, Rahul Krishna, Tim Menzies, and Pooyan Jamshidi. 2018. Transfer Learning with Bellwethers to find Good Configurations. arXiv:1803.03900 [cs.SE]

[25] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. 2012. How many trees in a random forest?. In *International workshop on machine learning and data mining in pattern recognition*. Springer, 154–168.

[26] Chanhyun Park, Miseon Han, Hokyoon Lee, and Seon Wook Kim. 2014. Performance comparison of GCC and LLVM on the EISC processor. In *2014 International Conference on Electronics, Information and Communications (ICEIC)*. 1–2. https://doi.org/10.1109/ELINFOCOM.2014.6914394

[27] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[28] Juliana Alves Pereira, Carlos Souza, Eduardo Figueiredo, Ramon Abilio, Gustavo Vale, and Heitor Augustus Xavier Costa. 2013. Software Variability Management: An Exploratory Study with Two Feature Modeling Tools. In *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse*. 20–29. https://doi.org/10.1109/SBCARS.2013.13

[29] Richard Pohl, Kim Lauenroth, and Klaus Pohl. 2011. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 313–322. https://doi.org/10.1109/ASE.2011.6100068

[30] Sandro Schulze, Michael Schulze, Uwe Ryssel, and Christoph Seidl. 2016. Aligning Coevolving Artifacts Between Software Product Lines and Products. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-Intensive Systems* (Salvador, Brazil) *(VaMoS '16)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/2866614.2866616

[31] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proc. of ICPE'17*. 39–50.

[32] Yilin Wang, Sasi Inguva, and Balu Adsumilli. 2019. YouTube UGC Dataset for Video Compression Research. In *Proc. of MMSP'19)*. 1–5. https://doi.org/10.1109/mmsp.2019.8901772

[33] Qingyao Wu, Hanrui Wu, Xiaoming Zhou, Mingkui Tan, Yonghui Xu, Yuguang Yan, and Tianyong Hao. 2017. Online Transfer Learning with Multiple Homogeneous or Heterogeneous Sources. *IEEE Transactions on Knowledge and Data Engineering* 29, 7 (2017), 1494–1507. https://doi.org/10.1109/TKDE.2017.2685597