Transferring Performance between Distinct Configurable Systems: A Case Study
Luc Lesoil, Hugo Martin, Mathieu Acher, Arnaud Blouin, Jean-Marc Jézéquel

To cite this version:

HAL Id: hal-03514984
https://inria.hal.science/hal-03514984
Submitted on 6 Jan 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.
Transferring Performance between Distinct Configurable Systems: A Case Study

Luc Lesoil, Hugo Martin, Mathieu Acher, Arnaud Blouin and Jean-Marc Jézéquel
IRISA, Université de Rennes 1, INSA Rennes France

ABSTRACT

Many research studies predict the performance of configurable software using machine learning techniques, thus requiring large amounts of data. Transfer learning aims to reduce the amount of data needed to train these models and has been successfully applied on different executing environments (hardware) or software versions. In this paper we investigate for the first time the idea of applying transfer learning between distinct configurable systems. We design a study involving two video encoders (namely x264 and x265) coming from different code bases. Our results are encouraging since transfer learning outperforms traditional learning for two performance properties (out of three). We discuss the open challenges to overcome for a more general application.

ACM Reference format:
https://doi.org/10.1145/3510466.3510486

ACKNOWLEDGMENTS

This research was funded by the ANR-17-CE25-0010-01 VaryVary project.

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 three years for the Linux kernel [21], 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 a.k.a. 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 [25]: since we feed the model with configurations and performance, we have to measure numerous configurations. Transfer learning techniques [21, 24] 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 measurements 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 distinct software systems performing the same task, such as compilers (e.g., gcc and 11vm), container managers (e.g., podman and docker), etc. 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 across software systems. We also discuss the limitations of our work and highlight the open challenges to face when scaling to other software systems. Data1 and code2 are publicly available.

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. We gather measurements about the performance of two software systems.

Software Systems. We select x264 and x2653, two video encoders. x264 and x265 realize the same task, but with different compression standards (resp. H.264 and H.265). It has a profound impact on the visual quality algorithms internally implemented. Though x265

1Dataset available at: https://zenodo.org/record/5662589
2Code available at: https://anonymous.4open.science/r/TL_cross_soft-855B/
3See x264 and x265 webpages: https://www.videolan.org/developers/x264.html and https://www.videolan.org/developers/x265.html
has the ambition to borrow heavily from x264’s features, x265 is not directly based on x264 source code. x265 is fully developed in C++ (and assembly) whereas x264 is written in C. x265 also implements novel algorithms such as CU-Tree the successor to x264’s macroblock-tree. Overall, x264 and x265 are two distinct software projects (i.e., x264 is not a version or a fork of x265). Importantly, both are developed by VideoLAN, which makes it easier to find similarities between them (same options, same conventions, etc.). From this respect, x264 and x265 can be seen as a favorable yet challenging case of transfer learning across systems.

Configuration Options. 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 configurations like (slow, 1) or (fast, 16) are accepted and valid for x264 and x265. In the end, we keep 35 configuration options common to x264 and x265 (out of resp. 118 and 168)\(^2\). 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 [22]\(^2\).

Input Data. We select eight input videos [18] extracted from the Youtube UGC Dataset [33], 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 m4v to the mp4 format. For this execution, we measure the percentage of cpu usage, etime the elapsed time in seconds and the file size of the resulting 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. We analyze the differences between the distribution of performance properties of x264 and x265. As a measure of (dis)similarity, we compute their 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 to get 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 simple and state-of-the-art transfer learning approach defined by Valov et al. [32]. 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.

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 0.99 is overall positively correlated consumption will probably be too difficult. The elapsed time etime is likely to be easy and transferring cpu usage 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

<table>
<thead>
<tr>
<th>Video</th>
<th>cpu</th>
<th>etime</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation</td>
<td>0.05</td>
<td>0.74</td>
<td>0.98</td>
</tr>
<tr>
<td>CoverSong</td>
<td>0.0</td>
<td>0.73</td>
<td>0.98</td>
</tr>
<tr>
<td>Gaming</td>
<td>-0.02</td>
<td>0.81</td>
<td>0.99</td>
</tr>
<tr>
<td>Lecture</td>
<td>0.07</td>
<td>0.75</td>
<td>0.98</td>
</tr>
<tr>
<td>LiveMusic</td>
<td>0.01</td>
<td>0.77</td>
<td>0.99</td>
</tr>
<tr>
<td>LyricVideo</td>
<td>0.14</td>
<td>0.73</td>
<td>0.96</td>
</tr>
<tr>
<td>MusicVideo</td>
<td>0.07</td>
<td>0.75</td>
<td>0.99</td>
</tr>
<tr>
<td>Sports</td>
<td>-0.0</td>
<td>0.78</td>
<td>0.98</td>
</tr>
</tbody>
</table>

2.3.2 Transfer Performance. In Figure 3, we depict the results varying for different inputs and performance properties:

\(^1\)The list of selected configuration options can be consulted here: https://anonymous.4open.science/r/TL_cross_soft-855B/replication/x26x/README.md
\(^2\)Results can be consulted at: https://anonymous.4open.science/r/TL_cross_soft-855B/replication/x26x/x264_x265_options.ipynb
\(^3\)A description of our python environment can be consulted at: https://anonymous.4open.science/r/TL_cross_soft-855B/replication/requirements.txt

![Figure 2: Model Shift, an approach of transfer learning](image-url)
The transfer of cpu consumptions is almost always negative. For example, after 15 configurations in the training set for the CoverSong video, in Figure 3d, the baseline is always more accurate than transfer learning. For the Lecture video, in Figure 3e, 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 3c. 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 ± 1.3 % and Model Shift stays at 220 ± 21 %.

Finally, for the encoded sizes of videos, Model Shift is at least equivalent to the baseline whatever the number of configurations 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 3h, 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 is possible to outperform the No Transfer baseline for 2 performance properties (out of 3), especially when the budget (i.e., the number of configurations 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 systems.

3 THREATS TO VALIDITY

Due to the cost of measurements (58 days of system time), we did not measure performance more than once. Therefore, 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 reproducing 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. 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

Although our experiment is encouraging to further explore this research direction, it does not cover all possible cases of transfer learning between software systems. In this section, we identify three lines of research and for each (1) we point out the limitations of our study; (2) we discuss the open challenges to overcome when transferring performance between distinct software systems; (3) we describe the potential approaches and solutions.

4.1 Find Transferable Software Systems

4.1.1 Limitation. A threat to our study is the choice of x264 and x265. There might be other pairs of systems for which non-functional properties are dissimilar e.g., with a correlation of zero, the transfer learning might perform poorly. For x264 and x265 our experiments show that two non-functional properties out of three are positively and strongly correlated.

4.1.2 Open challenge. 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.

4.1.3 Possible Solution. For now, the only reasonable assumption we can make is to choose software systems within the same domain e.g., compilers like gcc, 11vm or crang, container managers like podman or docker, learning libraries like theano, pytorch or tensorflow, text editors like emacs, gedit or vim, video encoders like vp9, x264, or x265. Measures on how performance distributions differ (e.g., with correlations) across systems can provide a first indicator on whether transferring is worth.
4.2 When and How to Transfer?

4.2.1 Limitation. The results of Section 2 show some limits of transfer learning: 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 make an accurate prediction. And the more configurations needed to be accurate, the larger the switching point.

4.2.2 Open Challenge. It is challenging to know when to apply transfer learning and when to switch to non-transfer learning. These results could also be better (or worse) with other transfer learning techniques (see Section 5). The challenge here is to determine the best learning approach to use for a pair of software systems.

4.2.3 Possible Solution. Empirical studies comparing different approaches on representative pairs of systems would be of great help in addressing this challenge. Specialized learning algorithms, capable of handling distribution shifts across software systems, are also expected.

4.3 What to Transfer?

4.3.1 Limitation. Our choice of configuration options is another limitation of our study. We have deliberately considered a favorable case: since x264 and x265 have been developed by the same team, the two systems share a common set of configuration options with the same range of values.

4.3.2 Open Challenge. In general, the configuration spaces will often be very different between the source and the target systems, making the sampling of configurations difficult in practice. The challenge is to map the configuration space of the source to the configuration space of the target. We illustrate these differences between configuration spaces with vp9, x264, and x265:

- The same feature is implemented in the source and the target with different names e.g., –level for x264 and –level-ide 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., the feature –rc-grain of x265 does not exist for x264;
- The feature does not have the same default value for the source and the target e.g., –qpmx 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. For instance, the constant rate factor –crf goes from 0 to 63 for vp9 but from 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 configuration option are increasing for one system and decreasing for the other.

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.

4.3.3 Possible Solutions. Mapping configuration spaces could be envisioned using recent advances in the variability community. We propose a very simple protocol: (i) Align features [20, 31] between systems; (ii) Meticulously model both configuration spaces e.g., with feature models [5, 14]; (iii) Analyse [8] and instrument [1, 2] them in order to create a resulting feature model generating configurations accepted by both 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 existing relationships between distinct configuration spaces.

5 RELATED WORK

Performance Comparison. In many areas, related work compares the performance of different tools performing the same task [6, 9, 19, 27, 29, 30]. 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, 24], it is applied to transfer performance of executing environments. Martin et al. develop TEAMs [21], 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 could 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, 34] (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 combines 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 shows it is challenging yet possible to transfer performance between distinct configurable 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 the applicability of transfer learning across distinct systems.

REFERENCES


[34] 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