Predicting the Performance of ATL Model Transformations

Raffaela Groner
raffaela.groner@uni-ulm.de
Institute of Software Engineering and Programming Languages, Ulm University
Ulm, Germany

Peter Bellmann
peter.bellmann@uni-ulm.de
Institute of Neural Information Processing, Ulm University
Ulm, Germany

Stefan Höppner
stefan.hoeppner@uni-ulm.de
Institute of Software Engineering and Programming Languages, Ulm University
Ulm, Germany

Patrick Thiam
patrick.thiam@uni-ulm.de
Institute of Medical Systems Biology, Ulm University
Institute of Neural Information Processing, Ulm University
Ulm, Germany

Friedhelm Schwenker
friedhelm.schwenker@uni-ulm.de
Institute of Neural Information Processing, Ulm University
Ulm, Germany

Matthias Tichy
matthias.tichy@uni-ulm.de
Institute of Software Engineering and Programming Languages, Ulm University
Ulm, Germany

ABSTRACT

Model transformation languages are special-purpose languages, which are designed to define transformations as comfortably as possible, i.e., often in a declarative way. Typically, developers create their transformations based on small input models which systematically cover the language of the input models. This makes it difficult for the developers to estimate how the transformations would perform for a large and diverse set of input models.

Hence, developers would benefit from an approach for predicting the performance of model transformations based on just abstract characteristics of input models. Regression approaches based on machine learning lend themselves well to such predictions. However, it is currently unknown, whether and which regression approach is suitable in this context as well as how a model should be abstractly characterized for this purpose.

We conducted several experiments to analyze how well different machine learning methods predict the execution time of model transformations defined in the Atlas Transformation Language (ATL) transformations for distinct sets of model characteristics. As possible methods, we have investigated linear regression, random forests and support vector regression using a radial basis function kernel.

The results of our experiments show that support vector regression is the best choice in terms of usability and prediction accuracy for the model transformation modules covered in our experiments and is thus suited for a prediction approach. In addition, simple model characterizations based only on the number of model elements, the number of references, and the number of attributes are a suitable way to easily describe a model and to achieve decent prediction accuracy.

ICPE '23, April 15–19, 2023, Coimbra, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0068-2/23/04.
https://doi.org/10.1145/3578244.3583727

CCS CONCEPTS

• **General and reference → Performance**; • **Computing methodologies → Machine learning approaches**; • **Software and its engineering → Model-driven software engineering**.

KEYWORDS

performance prediction, ATL, model transformation, machine learning, linear regression, random forests, support vector regression

ACM Reference Format:
Raffaela Groner, Peter Bellmann, Stefan Höppner, Patrick Thiam, Friedhelm Schwenker, and Matthias Tichy. 2023. Predicting the Performance of ATL Model Transformations. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3578244.3583727

1 INTRODUCTION

Model-Driven Engineering is used in multiple areas and, particularly, in the area of cyber-physical systems [47]. Model transformations as technology are used, e.g., to translate models in one formalism into models in another formalism, e.g. transforming UML to Alloy [4], or to update models@run.time in order to trigger a reconfiguration of a self-adaptive system [71]. In the past, many different model transformation languages like the Atlas Transformation Language (ATL) [39], Henshin [59], QVTo [51] or Viatra [66] have been proposed.

Götz et al. have identified 15 different categories of advantages and disadvantages of model transformation languages in their recent Systematic Literature Review [27]. Performance is one of these categories, as researchers continuously work to improve the performance of the execution of model transformation rules. The importance of performance is also shown by the results of an interview study [28] that we conducted previously.

However, many of the interviewees mention not only the core performance of the execution as an important aspect but also would like to have an easy-to-use approach to be able to predict the performance of a model transformation for varying model characteristics, i.e., sizes and structures of input models. This is known as what-if analysis in the Performance Engineering research area [13, p. 78]. Our interviewees report manual and ad-hoc approaches for this prediction. This prediction is not a trivial task as model transformation languages are often declarative and do not describe the concrete execution steps but leave this as implementation and optimization detail to the execution engine.

In order to realize this vision of a what-if performance analysis for model transformations, we need to predict the execution time of a given transformation based on characteristics of the input model. The contribution of this paper is such a prediction approach. Particularly, we investigate how well different types of machine learning methods predict the execution time of model transformations specified in ATL based on different sets of model characteristics.

Specifically, we answer the following research questions:

RQ1: How well do different machine learning methods predict the execution time of an ATL transformation?

RQ2: How well do those machine learning methods perform for different feature sets of model characteristics?

We answer our research questions by comparing linear regression, random forest regression, and support vector regression using a radial basis function kernel. Our comparison is based on a set of real world models provided by Kögel and Tichy [43] and obtained by mining GitHub repositories, as well as ATL transformations from the ATL zoo [21].

With respect to **RQ1**, our results show that depending on the transformation module and characterization of a model the linear regression approach yields an acceptable mean absolute percentage error (MAPE) of 1.54%. The random forest regression and support vector regression provide good predictions depending on the model characterization, as they only misestimated by less than 4% in 75% of the predictions. With respect to **RQ2**, the best compromise of prediction quality and usability is the support vector regression with model characteristics describing the number of objects, the number of references and the number of attributes. Its resulting MAPE lies between 2.07% and 10.63%. But our experiments also demonstrate a limitation. Currently, we cannot predict the execution time of transforming attribute values having an arbitrary size with a high variance that is not depending on the structure of the model, e.g., comments defined as string attributes. In such a case our approach achieves in the best case a MAPE of 1,010.91%.

In the next section, we introduce the aims and foundations of predicting the execution time, particularly, the investigated features of model characteristics. We present the relevant background of the used machine learning methods as well as the employed prediction performance metrics in Section 3. In Section 4, we present the design of our experiments. The results of the experiments are presented in Section 5. After a discussion on related work in Section 6, we conclude and give an outlook on future work in Section 7.

2 PREDICTION

The prediction approach is the key component of a what-if analysis. This analysis provides a closer look at the behavior of a system or can assist in decision making [26, 54]. A developer can, e.g., examine how the execution time of an ATL transformation changes if the input model used consists of several thousands model elements. Then, the developer can decide whether the performance is sufficient, the transformation should be optimized, or the transformation should be replaced with another technology.

Table 1: Overview of the feature sets.

Feature Set / Feature	1	2	3	4	5	6	7	8
Number of model elements (only objects)	✓	✓	✓					
Number of references		✓	✓					
Number of attributes			✓					
Number of model elements per type				✓	✓	✓	✓	✓
⌀ Fan-In per model element type					✓	✓		
⌀ Fan-Out per model element type					✓	✓		
Number of attributes per attribute type						✓		
⌀ Fan-In per model element type and per reference type							✓	✓
⌀ Fan-Out per model element type and per reference type							✓	✓
Number of attributes per model element type and per attribute type								✓

With the help of a prediction approach developers can easily analyze the influence of arbitrarily large models of varying structure on the execution time. By describing the input model based on its characteristics for the prediction, the very time consuming manual creation of large and realistic models is no longer necessary. The manual creation of very large models is also very complex, since a model must usually satisfy semantic conditions to be realistic.

To realize a prediction, there are two requirements: 1) a function that maps given characteristics of a model to a predicted execution time and 2) a way to characterize a model that contains all the necessary information for a prediction. To fulfill requirement 1, we performed experiments to compare linear regression, random forests and support vector regression using a radial basis function kernel. We also examined different characteristics of a model, since a suitable way to characterize a model (requirement 2) based on which one can predict the execution time of a transformation is currently unknown to the best of our knowledge.

Based on our experience with model transformations and their performance, we systematically defined characteristics to describe a model. We incrementally defined metrics (features) that describe a model in increasing degrees of detail, and then used their union as feature sets. The idea is always to add another piece of information to a metric by adding structural or type information. We also considered possible performance metrics for transformations that we have summarized in our data set [32].

The feature sets in Table 1 are divided into three groups, which differ in the amount of type information they contain. Within each group the feature sets differ in the amount of structural information they contain. We use a simple meta model and model (cf. Figure 1) to illustrate the feature sets in Table 1. Note that the models we use are directed typed graphs specified by metamodels.

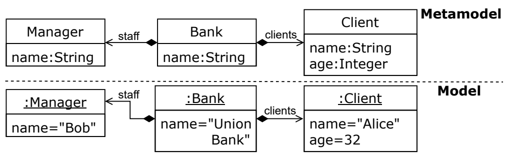

Figure 1: Metamodel and model of a simple bank.

The first group contains the feature sets 1 to 3, which cover increasing levels of structural information. Feature set 1 contains the total number of model elements regardless of which meta class they belong to. Feature set 2 extends that by the total number of references between the model elements irrespective of reference types. Feature set 3 then extends that by the total number of all attribute values. In our example (cf. bottom of Figure 1), the number of model elements is 3 (the client Alice, the manager Bob and the Union Bank), the number of references is 2 (*staff* between the manager Bob and the Union Bank and *clients* between the client Alice and the Union Bank) and the number of attributes is 4.

Group 2 contains the feature sets 4 to 6. These feature sets resemble the feature sets 1 to 3. However, they contain features for each type instead of a single feature, i.e., these feature sets contain three features each covering the number of instances of *Bank*, *Client*, resp. *Manager* for our example. Similarly, a feature, which contains the average Fan-In (incoming references) resp. Fan-Out (outgoing references) to characterize the references in more detail, is included in feature set 5 for each of the three types. Finally, feature set 6 extends that by containing the number of attributes for the type of each attribute, i.e., distinguishing *String* and *Integer* attributes in our illustrative example. For example, feature set 6 consists of 11 features for the input model at the bottom of Figure 1. Three features specifying that there is one bank, one client, and one manager in the model (these features are also part of feature set 4 and 5). Six features specifying the average Fan-In and Fan-Out of the bank, the client and the manager (these features are also part of feature set 5). The bank has an average Fan-In of 0 because it has no incoming references and an average Fan-Out of 2 because it references the client and manager. The client and manager both have an average Fan-In of 1, since they are referenced by the bank, and an average Fan-Out of 0. Furthermore, feature set 6 contains a feature for *String* attributes with the value 3 and a feature for *Integer* attributes with the value 1.

Group 3 contains the feature sets 7 and 8 that provide more details w.r.t. types of references as well as attributes. The model elements are not extended any further, since no additional type information is available for them. Feature sets 7 and 8 include features that differentiate the average Fan-In and Fan-Out for each reference type. In our illustrative example, features sets 7 and 8 include two features covering the average Fan-Out of the references *clients* and *staff*, respectively. This means in our example that for the Union Bank the average Fan-Out of the references *clients* is 1 and the average Fan-Out of the references *staff* is also 1. The feature sets in group 2 in contrast only contain one feature for the average Fan-Out of all references of the *Bank* element type. Feature set 8 extends feature set 7 by the number of attributes for the type of each attribute and the type of the associated model element, i.e., four features for *Manager.name*, *Bank.name*, *Client.age*, and *Client.name* result for our example. For the example model at the bottom of Figure 1, feature set 8 consists of 11 features. Three features specifying that there is a bank, a client, and a manager in the model (these features are also part of feature set 4, 5, 6 and 7). Two features specifying the average Fan-Out of the bank and two features specifying the average Fan-In of the client and the manager (these features are also part of feature set 7). One feature represents the average Fan-Out of the bank per reference type *clients* and one feature represents the average Fan-Out of the bank per reference type *staff*. Both of these features have the value 1 in our example. The average Fan-In of the client per reference type *clients* and average Fan-In of the manager per reference type *staff* have the value 1. Note, we do not include the average Fan-In/Fan-Out per model element type and per reference type for the model elements without any incoming/outgoing references, as this would result in feature sets containing a lot of features with a constant value of 0 (we discuss constant feature values in Section 4). As already mentioned, feature set 8 also contains 4 features representing the number of attributes per model element type and per attribute type.

The models we use also support inheritance between model element types. For example, we could extend our metamodel so that the types *Client* and *Manager* inherit from a model element type *Person*, which has an attribute *Person.name*. This would eliminate the need to define a corresponding attribute in the *Client* and in the *Manager* in the metamodel. Such a concept has naturally an impact on the performance of a transformation, because if the transformation considers, e.g., only model elements of the type *Person*, the number of possible model elements that are transformed is probably larger than if the transformation considers only model elements of the type *Manager*. This effect is reflected in our features by considering the inheritance hierarchy when quantifying the features for each model, e.g., "number of elements per type" is defined for the type being inherited from as well as for the type that inherits. Where the former feature is then a superset of the latter one. With regard to our previously described example, this means that we have one feature presenting the number of elements of the type *Client*, which has the value 1 (the client Alice) for our example model in Figure 1. We have another feature presenting the number of elements of the type *Manager*, which has the value 1 (the manager Bob). And we have a third feature presenting the number of elements of the type *Person*, which has the value 2 (the client Alice and the manager Bob) for our example model in Figure 1, since it represents all model elements of the type *Person* as well as all model elements that inherit from the type *Person*.

3 REGRESSION ANALYSIS METHODS

In this section, we first provide the definition of regression tasks. Subsequently, we summarize three popular regression approaches and define the metrics we use for evaluating the prediction quality.

3.1 Formalization

Let $X \subset \mathbb{R}^d$, $d \in \mathbb{N}$, be a d-dimensional data set. Furthermore, let $Y \subset \mathbb{R}$ be the set of target labels. By $N = |X| = |Y|$, we denote the number of elements in the sets X and Y. In a regression task, defined by the set $(X, Y) = \{(x_i, y_i)\}_{i=1}^{N} \subset X \times Y$, each data point

$x_i \in X$ is assigned to its corresponding target $y_i \in Y$.

The goal of each regression task is to find a mapping $f : \mathbb{R}^d \to \mathbb{R}$, such that $f(x_i) \approx y_i, \forall i = 1, \ldots, N$, i.e., such that the evaluation of f at data point x_i is *close* to its corresponding true target label y_i, for all data samples $(x_i, y_i) \in (X, Y)$ [52, p. 1-2].

In this work, the analyzed feature space reflects the different model characteristics. Therefore, each model is defined by an $x \in \mathbb{R}^d$, whereby the entries of vector x contain the features introduced in Table 1, such as the *number of model elements*. The set Y simply contains the corresponding *execution time*, for transforming each of the input models.

3.2 Ordinary Least Squares Linear Regression

One way of finding a mapping f with the properties described above is given by the ordinary least squares (OLS) method, which is also used in the field of machine learning. In the general OLS approach, the mapping f is obtained by minimizing the sum of the squared differences between the true target values and the approximated target values. Thus, we have to determine $\min \sum_{i=1}^{N} \Delta_i^2$, with $\Delta_i := f(x_i) - y_i, \forall i = 1, \ldots, N$. To this end, the *type* of the current regression function must be specified at first.

More precisely, in a *linear setting*, f is simply defined as $f(x) = \alpha_0 + \sum_{k=1}^{d} \alpha_k x^{(k)}$, $\alpha_k \in \mathbb{R}, \forall k = 0, \ldots, d$, whereby $x^{(k)}$ denotes the k-th dimension (i.e., the k-th feature of a data point $x \in X$). Let $\mathcal{X} \in \mathbb{R}^{N \times d+1}$ be defined as the matrix that contains all data points of X as row vectors extended by a leading one, i.e., the i-th row of matrix $\mathcal{X}$ is equal to $(1, x_i^{(1)}, \ldots, x_i^{(d)})$. Furthermore, let $\alpha \in \mathbb{R}^{d+1}$ be the vector consisting of the coefficients $\alpha_i, i = 0, \ldots, d$, and $y = (y_1, \ldots, y_N)^T$ be the vector consisting of the corresponding target labels. Then, to determine the regression function f, we have to solve the following equation system,

$$\min_{\alpha \in \mathbb{R}^{d+1}} \sum_{i=1}^{N} \Delta_i^2 = \min_{\alpha \in \mathbb{R}^{d+1}} \|\mathcal{X}\alpha - y\|^2, \tag{1}$$

whereby $\|\cdot\|$ denotes the Euclidean distance. In case the matrix $\mathcal{X}^T\mathcal{X}$ is *invertible*, the solution of Eq. (1), and hence the regression function, is given by $\alpha = (\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^T y$ [60, p. 52-53]. In this work, we will denote this approach simply by linear regression (LR).

3.3 Random Forests

In the field of supervised machine learning, Breiman et al. introduced the concept of classification and regression tree (CART) models [17], which are often simply denoted as decision trees. Thus, a decision tree is either a classification or a regression tree model.

The general training process of a decision tree can be briefly summarized as follows. Based on a pre-defined *split criterion*, e.g., the mean squared error, one of the features, i.e., input variables, is used to separate the initial training set into two subsets. The split criterion is used to obtain the *best* split to differentiate the training samples, with respect to their target values. Subsequently, each of the resulting subsets is divided analogously, until a pre-defined *stop criterion* is met. This procedure leads to a tree-based structure where each *node* is defined by its corresponding *split rule*, such as "is the statement *feature* $i \leq \theta$ true?", whereby $\theta \in \mathbb{R}$ is obtained during the training process. Each of the tree's branches leads to a terminal node, denoted as *leaf*. To predict the target label of a data point $z \in \mathbb{R}^d$, the features of z are fed to the decision tree, leading to exactly one of the leaves. For a regression tree, the target label of z is calculated as the average value of the target labels specific to the training samples that are included in the corresponding leaf.

It is common to apply ensembles of decision models [9, 45], due to a generally higher robustness and generalization ability. A regression ensemble is simply a set of regression models. A popular decision tree-based ensemble approach is the random forest (RF) method, which was introduced by Breiman [16]. In the basic RF variant, each ensemble member, i.e., decision tree, is trained in combination with a randomly chosen data- and feature subset of the fully available training set. More precisely, based on the training set $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$, for each ensemble member, we randomly draw $\tilde{N} \leq N$ data points. Moreover, for each of the resulting data subsets, we randomly draw $\tilde{d} \leq d$ features. The corresponding decision tree is then trained based on the obtained data- and feature training subset, in combination with the target labels of the randomly chosen data points.

To train a basic RF model, one has to specify the number of decision trees, the fraction of randomly drawn data points, and the fraction of randomly chosen features. In general, the ensemble member-specific training data points are sampled with replacement. This additionally ensures a variety in the size of the individually obtained, ensemble member-specific training sets. In general, the training subsets are randomly drawn according to the distribution of the initial, i.e., fully available, training set. This ensures that the individual training subsets reflect the initial data distribution.

To predict the value of a data point $z \in \mathbb{R}^d$, each of the ensemble members is evaluated in combination with z. The final prediction is obtained by averaging the outputs of each decision tree. Note that the RF approach does not provide any specific regression function.

3.4 Support Vector Regression

Another popular machine learning-based model used for classification and regression is the support vector machine, which was introduced by Vapnik [64]. Concerning support vector regression (SVR), the goal is to find a *smooth* linear function $f : x \mapsto \alpha^T x + \alpha_0$, which leads to the minimization of the term $\frac{1}{2}\alpha^T\alpha$ subject to the constraint $|y_i - (\alpha^T x_i + \alpha_0)| \leq \epsilon$, $\forall i = 1, \ldots, N$, whereby $\epsilon > 0$ is a pre-defined *error tolerance*. Since there is no guarantee for the existence of a function f that satisfies the aforementioned constraints, the idea of the SVR approach is to introduce so-called *slack variables*, denoted by $\zeta_i^-, \zeta_i^+ \geq 0$, which allow higher regression errors. This leads to the following optimization problem,

$$\begin{aligned}
&\min_{\alpha \in \mathbb{R}^d} \frac{1}{2}\alpha^T\alpha + C\sum_{i=1}^{N} \zeta_i^- + \zeta_i^+, \\
&\text{subject to } \forall i = 1, \ldots, N : \\
&y_i - (\alpha^T x_i + \alpha_0) \leq \epsilon + \zeta_i^-, \\
&(\alpha^T x_i + \alpha_0) - y_i \leq \epsilon + \zeta_i^+.
\end{aligned}$$

The parameter C regulates the penalty term with respect to the number of points violating the error tolerance.

Since the linear model is not always suitable for a given regression task, there are so-called *kernel functions* (*kernels*), which are used to replace the dot product and transfer the data into a higher-dimensional space. For instance, using the radial basis function kernel, the dot product between x and z, $x, z \in \mathbb{R}^d$, is replaced by $k(x, z) = \exp(-\gamma \|x - z\|^2)$, for some $\gamma > 0$. In our experiments we use the SVR approach with a radial basis function kernel.

3.5 Choice of Regression Analysis Approaches

There exists a plethora of different regression models. The current trend is dominated by (deep) artificial neural network (ANN) architectures [10, 46]. However, ANNs require a *sufficient* amount of training data to avoid *overfitting*. In addition, according to Saleh [58, p. 41], neural networks have a long training time compared to traditional machine learning approaches, apart from requiring an extremely large amount of training data. In particular, the need for input data would severely limit the usability of our prediction approach, as we discovered in our interviews [29] that there is already a lack of models to perform performance tests.

Due to the *small to high* amount of input models, as well as some low-dimensional feature spaces ($1 \leq d \leq 1,964$), we decided to focus on the LR, SVR and RF-based regression approaches. They are popular machine learning tools that are widely used in *simple* and *complex* pattern recognition tasks, such as in pain intensity recognition scenarios [8, 61].

3.6 Performance Evaluation Metrics

There exist different measures to determine the *quality* of regression models. To this end, let $Z = \{(z_i, y_i)\}_{i=1}^N \subset \mathbb{R}^d \times \mathbb{R}$ be a test set. Furthermore, let $\hat{y}_i$ denote the corresponding predicted values, specific to the data points z_i, $i = 1, \ldots, N$, obtained by the approaches LR, RF or SVR. We focus on the mean absolute percentage error (MAPE) [50] and on the absolute percentage error (APE) which are defined as follows,

$$\text{MAPE} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|, \tag{2}$$

$$\text{APE}_i = \left| \frac{y_i - \hat{y}_i}{y_i} \right|, \forall i = 1, \ldots, N. \tag{3}$$

Equation (2) and (3) imply that the MAPE measure and the APE measure are not defined if $y_i = 0$. However, note that, we want to approximate the execution time. Therefore, it holds, $y_i \neq 0, \forall i = 1, \ldots, N$. In order to assess the distribution of the predicted values, we use the 95th percentile of the absolute percentage error (P_{95}(APE)). We use the P_{95}(APE), since it is robust against outliers and provides an upper bound for the APE below which 95% of the values lie. Based on the suggestion from Hyndman and Fan [37], we used the median unbiased method of NumPy in case the percentile lies between two data points.

4 EXPERIMENTS DESIGN

The design of our experiments can be divided into five parts. We selected transformation modules and suitable input models. Then, we collected the data, planned the detailed setup of the experiments and defined evaluation criteria that we use to evaluate our results.

Transformation module selection: Our choice of transformation modules is influenced by three factors: 1) the transformations should be as different as possible 2) the structure of the input models transformed should be as different as possible and 3) enough real input models should be available to train the regression approaches.

Considering these factors, we use the transformation modules EMF2KM3 (*EMF2KM3*), Make to Ant (*MAKE2ANT*), ATL to BindingDebugger (*ATL2Debugger*), ATL to Problem (*ATL2Problem*) and ATL to Tracer (*ATL2Tracer*) from the ATL zoo [21]. We reimplemented the transformation Models Measurement (*EMF2Measure*) from the ATL zoo [21] to transform Ecore metamodels [1] instead of KM3 [40] models. We also had to partially modify some transformations to execute them. The necessary changes are documented in our supplementary material [30].

EMF2KM3 transforms Ecore metamodels [1] into KM3 [40] models. This is a simple translation from one model formalism into another one, for which we can use real Ecore metamodels from the data set from Kögel and Tichy [43]. *EMF2Measure* performs complex computations of metrics for Ecore metamodels. Thus, we can compare two different types of transformations on the same real input models.

MAKE2ANT transforms Makefiles into Ant files. Compared to Ecore metamodels, the transformed Makefile models are very flat and the model elements are less branched among themselves.

ATL2Debugger adds debugging instructions to an ATL transformation and *ATL2Tracer* adds tracing information to an ATL transformation. We included these two modules because they do not create new output models but update the given input models resp. ATL transformations.

ATL2Problem analyzes a given ATL transformation to identify non-structural errors. Thus, it implements a simple translation between two model formalisms by generating an error report model, but also contains complex analysis code.

Input model selection: For the model selection, we paid attention to choose real-world input models.

We used the data set from Kögel and Tichy [43] to obtain input models for *EMF2KM3* and *EMF2Measure*. This data set contains 31,799 Ecore metamodels including their version history. We always selected the newest version and excluded all models that could not be transformed or not be parsed due to unresolved dependencies. In total, we obtained 4,804 models as input for *EMF2KM3*. For *EMF2Measure*, we obtained 4,797 models. The different numbers result from the fact that *EMF2KM3* throws an exception for a model that can be transformed by *EMF2Measure*. And while performing *EMF2Measure*, 5 models caused stack overflow errors and for 3 models the execution did not terminate.

In order to obtain real Makefiles for *MAKE2ANT*, we mined GitHub via its REST API on the 22nd of April 2021 using the query *q=language:makefile*. The search API provides the first 1,000 search results for a query [25], which we filtered by parsable Makefiles and available suitable licenses. This way we received 247 real Makefiles.

We use the transformations available at the ATL zoo [21] as input models for *ATL2Debugger*, *ATL2Tracer* and *ATL2Problem*. We filtered out duplicates, ATL queries and ATL libraries, which are not transformed by these three transformations. This way we obtained 220 ATL modules that we use as input models.

Data collection: In order to avoid including model elements in our feature sets which are not touched by a transformation, we manually analyzed the transformation modules to identify the types of model elements, which are involved in the transformation. In this analysis, we have also taken into account the inheritance hierarchies specified by the corresponding metamodel. This means that if a model transformation touches a model element of a certain type, all types inheriting from it are also relevant types to be considered. To create our feature sets, we subsequently collected only the information on the model elements that belong to a relevant type. If we would collect information about all elements in an input model, the regression approaches could learn associations that do not exist because they have received information that has no impact on the execution time. The prediction could then depend, e.g., on EAnnotations, which are used with varying frequency in Ecore metamodels but are not considered at all in *EMF2KM3*. There are two things to note, first this manual analysis could be automated with a type inference for ATL transformations. Second, we can ignore the irrelevant model elements, since we only predict the execution time of the transformation and not the loading or the saving of the models.

Table 2: Overview of the modules and input models used.

Module	Relevant Model Elements		Execution Time [ms]	
	Min	Max	Min	Max
EMF2KM3	1	17,453	3.0	309.2
EMF2Measure	1	13,401	5.0	47,707.2
MAKE2ANT	1	196	0.9	2.16
ATL2Debugger	1	3,441	0.8	1,371.7
ATL2Tracer	15	7,574	13	2,484
ATL2Problem	20	11,674	4	10,104

After we determined for each transformation module the relevant model element types, we collected the data for our feature sets summarized in Table 1 using our own Eclipse Plugin (available here [30]). We monitored the execution times in ms with Java Microbenchmark Harness (JMH) [2]. The measured time values were rounded to zero, one or two decimal places, depending on the module, based on the reported errors in the result report of the JMH measurements to mitigate measurement errors.

Based on the obtained data, we can characterize the transformation modules and the input models as summarized in Table 2, in which the first double column (Relevant Model Elements) shows for each module used the lowest (Min) and the highest (Max) number of relevant model elements that we counted in the available input models. In the second double column (Execution Time [ms]), Table 2 shows for each module the smallest (Min) and the largest (Max) execution time that we measured for the respective available input models.

Experiments set-up: In the design of our experiments, we focus on two objectives: First, we compare the three different machine learning approaches on the basis of the feature sets we defined in Table 1. Second, we investigate whether our individual feature sets yield better prediction results than feature sets based on a feature selection method using the variance of features [15, p. 187].

For each combination of feature set and machine learning, we performed a 10-fold cross-validation evaluation [11], consisting of dividing the input data into 10 test-set-training-set pairs. We removed the features with a variance of 0 from the respective feature set for each run. Such features are a constant value over all models and, therefore provide no useful information for the prediction.

We re-ran our experiments using a feature selection method based on the variance of the features, to compare the resulting predictions to the results yielded by our feature sets. We used the union of all feature sets from Table 1 as the basis and calculated for each feature its variance. Afterwards, each feature is removed from the feature set whose variance falls below a given threshold value. Filtering features based on their variance is also called Variance Thresholding and is a basic technique for feature selection. This feature selection method is based on the assumption that features with a small variance have little benefit as input than features with a high variance [3].

In the following we use $\mathrm{VAR_F}$ to denote the variance of one feature and $\mathrm{VAR_{FS}}$ to denote the set of all variances for each feature in a feature set. We use a variance of 0 ($\mathrm{VAR_F}>0$) as a threshold, as well as the 75th ($\mathrm{VAR_F}>\mathrm{P_{75}}(\mathrm{VAR_{FS}})$), 85th ($\mathrm{VAR_F}>\mathrm{P_{85}}(\mathrm{VAR_{FS}})$), 95th ($\mathrm{VAR_F}>\mathrm{P_{95}}(\mathrm{VAR_{FS}})$), and 99th ($\mathrm{VAR_F}>\mathrm{P_{99}}(\mathrm{VAR_{FS}})$) percentile of the set of all variances of each feature from the union of all feature sets as further thresholds. This means we remove all features from the union of all feature sets with a constant value ($\mathrm{VAR_F}>0$). And for the other thresholds, we retain only X-% of the features with the highest variance removing those below the given variance threshold. Whereby we vary X in the different experiments between 25% ($\mathrm{VAR_F}>\mathrm{P_{75}}(\mathrm{VAR_{FS}})$), 15% ($\mathrm{VAR_F}>\mathrm{P_{85}}(\mathrm{VAR_{FS}})$), 5% ($\mathrm{VAR_F}>\mathrm{P_{95}}(\mathrm{VAR_{FS}})$) and 1% ($\mathrm{VAR_F}>\mathrm{P_{99}}(\mathrm{VAR_{FS}})$). We decided to use these thresholds since many of our features have a very small variance. For the calculation of the percentiles, we used the linear method provided by NumPy in Python since the use of the median unbiased method in combination with the 99th percentile resulted in an empty feature set for *MAKE2ANT*.

Note, reducing the feature set to the necessary features is always advisable, as this reduces the computational overhead and reduces the risk of overfitting, especially for linear regression approaches [41, p. 173-174]. We also normalized the values of the features in order to balance their influence, since some of them have large differences in their value ranges [44, p. 48].

We used Scikit-learn [53] to implement the three machine learning approaches. We did not further configure the LR approach. In the implementation of the SVR and the RF approach, we also optimizes their hyperparameters. To select the possible values of the hyperparameters C and γ for the SVR approach, we used the suggestions of Hsu et al. [35]. For the RF approach, we optimized the number of threes and the number of samples required to be at a leaf node using the default values from Scikit-learn and the values suggested by Probst et al. [57]. Probst et al. [57] mentions more hyperparameters than the two we optimized, but these are either not offered by Scikit-learn or the proposals lead to invalid values for some of our feature sets. We used a seed to reproduce our results and we provide the data and scripts in [30].

Evaluation criteria: To evaluate the results of our experiments, we evaluate the quality of the predictions based on the errors. For this purpose, we focus on the mean absolute percentage error (MAPE) in %, the absolute percentage error (APE) in % and the 95th percentile of the absolute percentage error ($\mathrm{P_{95}}$(APE)) in %. A good combination of feature set and approach is characterized by a high accuracy (small values for the MAPE, the APE and the P_{95}(APE)) of the predictions. In analyzing the results, we proceed in two phases. First, we consider the individual results for each combination of transformation module, feature set, and machine learning approach. The goal here is to check whether there are combinations that deliver particularly good results or are completely useless predictions. In the second phase, we consider the MAPE and the P_{95}(APE) over all modules (excluding *ATL2Tracer*, cf. our 3rd finding) for all combinations of machine learning approach and feature set. The aim in the phase is to compare the combinations of feature set and machine learning approach over all modules that yield the smallest values for the MAPE and the P_{95}(APE) to subsequently answer our research questions.

4.1 Threats to Validity

In this section, we discuss the major threats to validity of our study based on the guidelines by Wohlin et al. [72, p. 102-110].

Conclusion validity: There may be other approaches that provide better predictions than the ones we investigated. We chose the approaches examined, because we want an approach that works out-of-the-box so we can use it as a basis for our what-if analysis.

There may be a combination and encoding of model characteristics that can provide better predictions than those we have investigated. The feature sets we have defined are by no means a complete list of all possible combinations and encodings of model characteristics.

In order to ensure the repeatability of our experiments, we used a seed to create the test set-training set pairs and to configure the implementation of the RF approach.

Construct validity: The manual analysis of the transformation modules to identify the types of model elements, which are touched by a transformation, was performed by two authors independently and subsequently the results were compared, to avoid a biased selection of relevant types of model elements.

While we measured the execution times we always used the same software versions for Java, JMH and the Ubuntu OS, but we were confronted with some hardware issues. So the time measurements for *EMF2KM3* and *EMF2Measure* were done on one machine. The time measurements for *MAKE2ANT* and *ATL2Problem* were done on the same machine, but a hard disk was replaced. The time measurements for *ATL2Debugger* and *ATL2Tracer* were performed on a completely different machine. Overall, the results of our experiments are not affected by this, since we do not mix timing data from different machines for one transformation module.

In order to prevent a biased selection of input models, we used all input models available excluding only those which could not be transformed. In addition, due to 10-fold cross-validation, each input model was included exactly once in a test set.

External validity: We only used transformations from the ATL Transformation zoo. However, we paid attention to the fact that they used different types of input models and execute different types of transformations, to improve the generalizability of our results. However, we cannot guarantee that these different types of transformations represent the general use of ATL.

It is not clear whether our results for the best approach and feature set apply to other models and transformations as well. However, we consider various different types of transformation modules. In addition, we considered Ecore metamodels as well as models describing Makefiles, which differ strongly in their structure, since models describing Makefiles are by far flatter and model elements reference each other less often than in Ecore metamodels.

5 EXPERIMENTS RESULTS

In this section, we present the results of our experiments.

Tables 3 and 4 show the MAPE in % and the P_{95}(APE) in % of the predicted execution times for each transformation module and each combination of machine learning approach and feature set used. The row "All modules" contains the MAPE and the P_{95}(APE) over all modules used except *ATL2Tracer*. We explain the reason for this in our 3rd finding. The cells colored in green contain the lowest combination of MAPE and P_{95}(APE) for an approach and feature set combination per module. The cells colored in yellow contain the lowest combination of MAPE and P_{95}(APE) for an approach and feature set combination over all modules except *ATL2Tracer*. Four findings emerge from the data presented in the Tables 3 and 4:

1st finding: Most of the time the RF approach yields the best results but also the SVR approach yields good predictions.

2nd finding: The LR approach performs rather well for *EMF2KM3* and *MAKE2ANT*. This could be due to the fact that these two transformations simply transform an element of the input model into an element of the output model. The other transformations, perform more calculations and might traverse the input model several times. But there are also some combinations of the LR approach and feature sets resulting in extremely high values for the MAPE for *EMF2KM3*, *ATL2Problem* and *ATL2Tracer* (cells marked in orange in the Tables 3 and 4). There are two possible reasons for these extremely high values. Single extreme outliers can reduce the quality of the prediction or there is no linear relationship between the used feature set and the execution time. Our analysis of the predicted values for *EMF2KM3* shows that the extremely high MAPE values are the result of outliers. This can also be seen by the fact that the values for P_{95}(APE) is between 4.77 and 4.84% in the respective cells. Such outliers can occur, e.g., if the training set generated for a fold does not sufficiently represent the corresponding test set. For the remaining cells marked in orange, the predicted values rarely match the real ones and scatter strongly, indicating that there is no linear relationship between the execution time and the feature set used for these modules. The large deviation of the predicted values from the real ones can also be seen in the fact that the values for P_{95}(APE) are between 14,918.72 and 1.1e14% in the respective cells.

3rd finding: There is a limitation of our approach to predict the execution time for transformations which transform attributes, whose values can have an arbitrary unlimited size, and its size is unrelated to the features in our feature sets. In *ATL2Tracer* comments, which are string attributes, are explicitly copied. The size of comments depends on whether and what the developer of the input model has written. Hence, there is no combination of machine learning approach and feature set that yields decent results for *ATL2Tracer*. We repeated our experiments without transforming the comments and the RF approach with feature set $VAR_F{>}P_{85}(VAR_{FS})$ yields the best results (MAPE=2.0%, P_{95}(APE)=5.28%). We plan to explore how

Table 3: MAPE in % and P_{95}(APE) in % of the predicted execution time using the features sets 1 to 8.

Module	Ap-proach	Feature Set 1		Feature Set 2		Feature Set 3		Feature Set 4		Feature Set 5		Feature Set 6		Feature Set 7		Feature Set 8	
		MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)
EMF2-KM3	LR	3.05	6.67	2.7	6.38	2.17	5.6	1.94	4.84	1.54	4.17	1.59	4.48	1.5e7	4.77	9.5e9	4.84
	RF	1.51	4.19	1.22	3.24	1.12	3.05	1.43	3.58	1.44	3.86	1.13	3.12	1.5	3.93	1.53	3.95
	SVR	1.94	4.9	1.91	4.84	2.07	4.9	1.99	4.72	1.77	4.81	1.36	3.37	1.59	3.79	1.6	3.97
EMF2-Measure	LR	451.23	573.29	236.55	809.13	309.93	772.96	628.81	1, 594.13	958.5	2, 773.15	1, 114.0	3, 302.94	923.75	2, 952.88	1, 248.33	3, 188.14
	RF	23.96	39.88	8.5	21.56	7.0	15.27	11.42	18.92	7.15	10.64	7.22	10.4	17.59	11.42	15.95	12.38
	SVR	10.14	37.84	8.85	24.6	6.6	16.72	9.96	25.9	16.55	35.07	17.65	37.38	33.21	92.99	46.37	194.35
MAKE2-ANT	LR	3.0	9.2	2.46	7.35	2.21	6.01	1.84	6.52	1.68	5.01	1.68	5.01	1.68	5.04	1.68	5.04
	RF	2.81	9.8	2.29	6.58	1.9	4.89	1.26	3.25	1.19	2.79	1.13	2.8	1.28	3.2	1.28	3.2
	SVR	3.01	9.29	2.91	8.12	2.84	7.49	3.0	9.07	3.85	9.77	2.71	9.89	3.38	9.61	3.24	9.89
ATL2-Debugger	LR	493.26	2, 047.01	359.23	1, 403.32	369.81	1, 451.78	383.73	1, 634.01	400.87	1, 596.19	400.87	1, 596.19	416.79	1, 719.12	419.94	1, 715.83
	RF	11.1	37.01	6.71	21.38	6.82	21.57	5.13	13.63	5.12	13.96	4.98	12.95	5.1	14.27	5.1	14.0
	SVR	11.24	29.52	5.24	16.84	5.0	14.45	10.54	41.79	14.62	40.89	14.89	51.59	13.34	40.21	11.38	29.48
ATL2-Problem	LR	1, 279.76	3, 976.64	1, 391.7	4, 460.82	1, 013.48	3, 373.92	412.75	1, 176.63	2, 879.66	10, 538.05	4.9e9	1.1e10	2.5e13	1.1e14	1.2e13	4.4e13
	RF	13.83	53.29	14.19	53.09	14.03	49.26	11.38	37.75	11.95	41.57	10.38	36.55	157.67	669.36	217.74	447.72
	SVR	15.64	42.77	14.53	38.93	10.63	30.69	28.15	65.4	28.02	96.62	27.56	93.9	58.48	246.6	57.07	253.26
ATL2-Tracer	LR	1, 477.44	7, 194.49	1, 485.1	7, 194.85	1, 481.16	7, 160.47	1, 429.58	6, 772.25	1, 202.38	6, 889.34	1, 203.33	6, 881.9	1.3e14	14, 918.72	1.4e14	16, 386.54
	RF	1, 449.71	8, 230.44	1, 368.79	7, 196.96	1, 321.88	6, 606.88	1, 098.83	6, 264.37	1, 039.94	5, 309.84	1, 038.66	5, 203.76	1, 019.66	4, 998.52	1, 013.76	4, 981.58
	SVR	1, 778.13	9, 037.99	1, 630.03	8, 626.71	1, 630.08	8, 645.7	1, 170.31	7, 834.5	1, 107.94	6, 420.77	1, 108.67	6, 420.05	1, 043.71	5, 751.31	1, 035.09	5, 937.85
All modules	LR	249.81	559.77	149.06	674.0	175.16	648.95	311.18	1, 242.39	517.83	2, 096.92	1.0e8	2, 435.36	5.3e11	2, 380.19	2.6e11	2, 729.15
	RF	12.47	21.41	5.04	13.69	4.28	10.3	6.37	10.84	4.4	7.81	4.25	6.6	12.41	9.44	12.94	9.48
	SVR	6.29	22.36	5.51	16.49	4.45	11.49	6.47	14.48	9.54	19.56	9.84	19.98	17.84	37.15	23.91	65.74

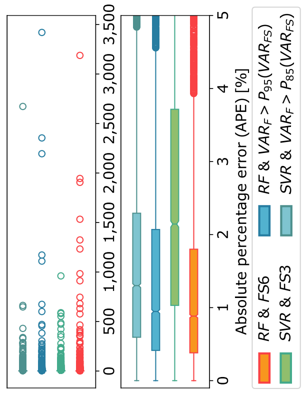

Figure 2: APE of all modules (excluding *ATL2Tracer*).

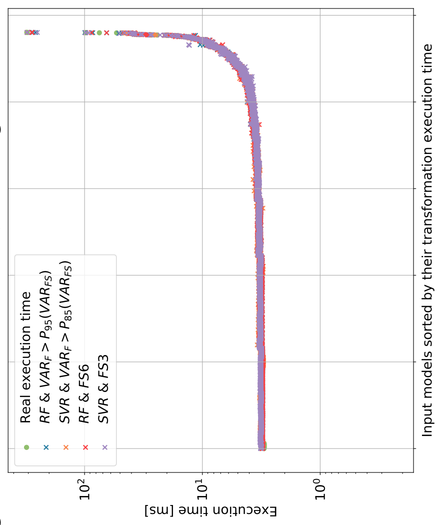

Figure 3: Real vs. predicted execution time of *EMF2KM3*.

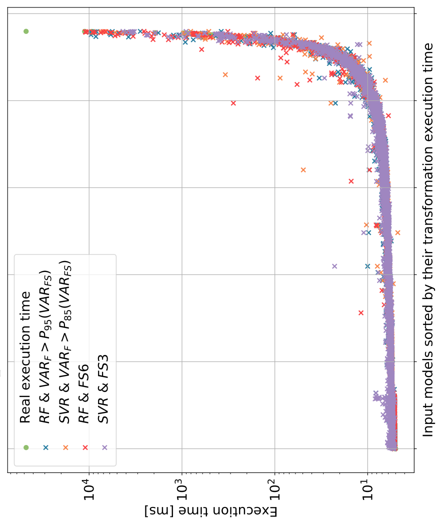

Figure 4: Real vs. predicted execution time of *EMF2Measure*.

we can deal with this issue in more detail in our future work. A solution would be to include the size of the strings in our feature sets, but we want to avoid simply adding a new feature for every possible constraint. Due to the fact that there is a sweet spot between the size of a feature set in terms of overfitting (usage of too large feature sets is prone to overfitting [41, p. 173-174]), as well as the usability and the quality of the prediction. The other modules also transform string attributes, but their size is similar for all models, thus their transformation is approximated as a constant overhead.

Table 4: MAPE in % and P_{95}(APE) in % of the predicted execution time using the variance to filter the features used.

Module	Approach	$VAR_F>0$		$VAR_F>P_{75}(VAR_{FS})$		$VAR_F>P_{85}(VAR_{FS})$		$VAR_F>P_{95}(VAR_{FS})$		$VAR_F>P_{99}(VAR_{FS})$	
		MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)	MAPE	P_{95}(APE)
EMF2-KM3	LR	1.79	4.78	1.63	4.56	1.92	5.13	2.18	5.7	2.12	5.42
	RF	1.08	3.03	1.08	3.03	1.08	3.03	1.1	3.03	1.28	3.54
	SVR	1.48	3.84	1.52	3.75	1.67	4.04	1.92	4.58	2.28	5.81
EMF2-Measure	LR	1, 078.01	3, 198.17	574.49	1, 749.6	575.49	1, 616.43	380.97	942.02	481.68	625.89
	RF	14.31	10.06	8.22	12.05	6.91	11.4	6.34	12.36	27.0	44.38
	SVR	47.0	189.41	7.55	11.01	4.27	7.89	6.04	14.3	12.01	41.05
MAKE2-ANT	LR	1.71	5.01	1.89	6.38	1.84	6.52	2.34	6.22	2.55	7.36
	RF	1.2	2.99	1.29	3.37	1.33	3.91	2.08	5.69	2.37	7.96
	SVR	3.73	9.71	3.43	9.76	3.12	8.9	2.82	7.69	2.65	7.33
ATL2-Debugger	LR	421.44	1, 641.25	387.98	1, 609.67	379.11	1, 499.09	370.21	1, 470.24	494.24	2, 050.46
	RF	5.11	13.27	5.12	13.91	5.05	13.8	8.6	29.82	11.24	37.06
	SVR	16.58	59.66	9.87	45.3	4.74	13.78	7.18	26.28	11.42	30.07
ATL2-Problem	LR	$7.0e12$	$3.8e13$	$1.5e12$	$7.3e12$	44, 034.14	42, 980.73	245.3	722.23	305.5	786.39
	RF	114.97	240.19	11.22	40.53	13.5	43.08	9.97	31.51	8.93	30.89
	SVR	53.2	229.22	34.3	125.12	26.58	93.21	20.76	76.23	8.24	23.09
ATL2-Tracer	LR	10, 138.09	32, 682.46	1, 414.92	6, 340.63	1, 187.59	6, 134.3	1, 465.1	7, 011.33	1, 480.47	7, 114.73
	RF	1, 010.91	5, 036.55	1, 015.0	4, 980.31	1, 049.39	5, 742.53	1, 144.09	6, 079.69	1, 337.63	7, 130.48
	SVR	1, 052.39	6, 170.99	1, 067.61	5, 920.41	1, 111.38	6, 496.49	1, 631.95	9, 211.29	1, 629.14	8, 633.83
All modules	LR	$1.5e11$	2, 555.34	$3.2e10$	1, 481.9	1, 219.02	1, 223.61	191.87	666.19	242.75	600.85
	RF	9.77	6.71	4.71	6.78	4.16	6.78	3.92	8.63	13.67	23.24
	SVR	24.19	66.05	5.26	9.37	3.52	6.71	4.38	10.72	7.15	22.99

In the following, we will exclude *ATL2Tracer*, because its data does not provide any further information.

4th finding: With respect to the best prediction results using feature selection based on variance (cf. Table 4), there is only a slight difference in quality from the best prediction results using one of our feature sets (cf. Table 3).

In the following, we take a closer look at the combinations of approaches and feature sets of the yellow marked cells in the Tables 3 and 4 since they provide the best results over all modules except *ATL2Tracer*. But we exclude the LR approach since our previous findings already show that it is not useful.

Figure 2 shows the absolute percentage error (APE) resulting from using the RF approach combined with feature set 6 (FS6) and $VAR_F>P_{95}(VAR_{FS})$ and the SVR approach combined with feature set 3 (FS3) and $VAR_F>P_{85}(VAR_{FS})$ for all modules (excluding *ATL2Tracer*). The y-axis shows the individual combinations and the x-axis the APE in %. The upper plot shows all values and bottom plot shows a zoomed in view. For each feature set-approach combination one box plot is displayed, showing the APE for all modules. Note that we trained the machine learning approaches for each module individually and did not use the same trained machine learning approach to predict the execution times of all modules. From Figure 2 one obtains various findings:

5th finding: The upper part of the plot shows that every combination of approach and feature set leads to some extreme outliers.

6th finding: The interquartile ranges are very small and the upper quantile is slightly less than 4%. This means that the approaches are misestimating by less than 4% in 75% of the predictions.

Figures 3 to 7 show for each module the actual execution time in comparison to the predicted ones.

7th finding: The plots show that the different feature sets provide quite similar predictions. But one can also see that longer execution times are usually not as well predicted as shorter ones. This may be due to the fact that our data sets consist mainly of small models, which do not lead to particularly long execution times.

Summary and discussion: Based on our results, we obtain the following answers for our two research questions. Concerning **RQ1**, we can conclude that the RF approach and the SVR yield decent results (cf. Tables 3 and 4). With respect to **RQ2**, we can conclude that the SVR approach in combination with feature set $VAR_F>P_{85}(VAR_{FS})$ yields the best prediction results (MAPE=3.52%, P_{95}(APE)=6.71%, over all modules except *ATL2Tracer*). In terms of usability for a what-if analysis we would recommend the SVR approach in combination with feature set 3, because a model can be described much easier only using the number of model element, the number of references and the number of attributes and the results are only slightly worse (MAPE=4.45%, P_{95}(APE)=11.49%, over all modules except *ATL2Tracer*).

Overall, our results are very good considering our small amount of training data for some transformation modules in the machine learning context. Prathanrat and Polpraser [56] use 909 records (our smallest data set contains 220 input models) to train a random forest in order to predict the performance of Jupyter notebook. The feature set used by Prathanrat and Polpraser consists of information about the execution, such as memory and CPU usage. They have optimized the hyperparameters and trained one machine learning model to use it for different programs. Subsequently, they obtained a MAPE of about 12.8%. Maros et al. [48] use, among other approaches, a random forest approach to predict the performance of cloud applications on Apache Spark. Depending on the experiment, they

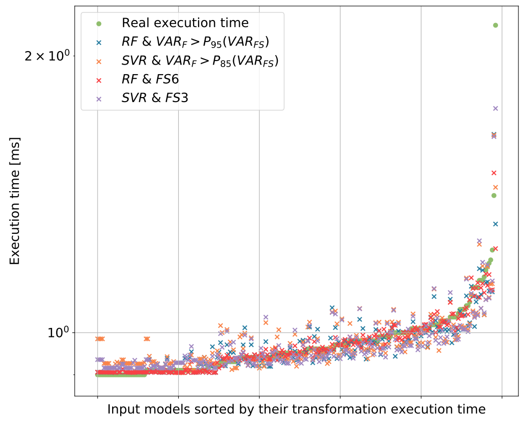

Figure 5: Real vs. predicted execution time of *MAKE2ANT*.

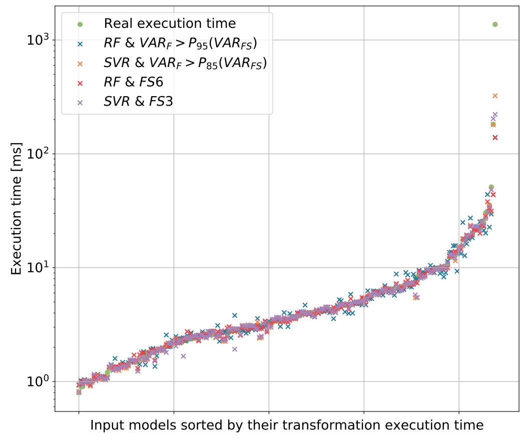

Figure 6: Real vs. predicted execution time of *ATL2Debugger*.

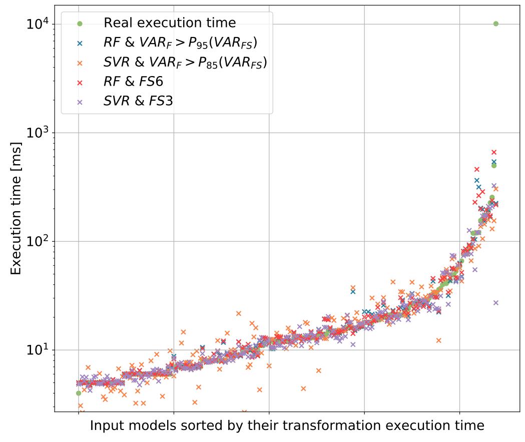

Figure 7: Real vs. predicted execution time of *ATL2Problem*.

use data consisting of up to 1,000 GB, 20 million rows and 2,500 images. They defined two different types of feature sets, namely black box models and gray box models. The black box models contain only information that is already known before the execution and the gray box models also contain information about the execution, like the average execution time of a task. Maros et al. optimized the hyperparameters and trained for each application a machine learning model to predict the execution time. They obtained a MAPE that lies between 2.4% and 41.9% using the black box models and a MAPE between 4.5% and 91.3% using the gray box models.

Based on these other works, it is also shown that the type of feature sets used can influence whether a trained machine learning model can be reused for predicting the performance of other applications/programs or not. Since the feature set used by Prathanrat and Polpraser [56] contains information about the execution it is sufficient to train one machine learning model and use it for different programs. The black box models used by Maros et al. [48] as well as the feature sets used in our presented work only use information available before the execution. Therefore the machine learning approach lacks execution details and is specialized for one application resp. one transformation module. In case of modifications or other applications or transformation modules, the data collection and training must be repeated.

6 RELATED WORK

To the best of our knowledge, no other approach exists for predicting the performance of model transformations. There are however many works that focus on machine learning for performance prediction in other engineering areas as well as several publications that apply machine learning in other areas of transformations. Other publications that can be related to our work are papers in the area of performance engineering on transformations and what-if analyses.

While there exist no other works that implement performance prediction for transformations, there are two short papers that envision this idea. In [33], the authors roughly outline the idea of using raw data from monitoring transformations to build a framework for performance prediction. Similarly, the authors of [69] propose an approach for predicting the performance of transformations within a three phase process for performance engineering. Their approach envisions the use of automatically generated models as input for a transformation. The generated models are automatically transformed and the performance for the transformation is measured and saved. All collected data then serves as a basis for predicting the performance of the transformation for other input models. Although not specifically stated, both proposed approaches envision performance predictions based on pre-collected execution data, which is similar to the machine learning approach presented in this paper. However, those papers do not present any evaluation.

What-if analyses have been applied to many different fields. The authors of [12] applied it in e-commerce, while Baybutt [6] describes the usage of a what-if analysis for hazard analysis and, Chaudhuri and Narasayya [19] use it for index selection in relational databases. More closely related to our work is the use of a what-if analysis by Herodotos and Babu [34] for a cost optimization approach of MapReduce programs. Their setup allows them to use an in-house developed what-if engine to investigate how the execution time of an application changes when the reduction tasks are increased or the number of nodes involved is changed. While their approach does not focus on model transformations, it demonstrates that a what-if analysis can be used to predict the impact of different factors on the performance of a system.

As previously stated, machine learning has not yet been used for performance prediction of model transformations, it has however been used in other aspects of model transformations. Kappel at al. [42] suggested to apply machine learning techniques, in order to improve the quality of model transformations produced through the concept of model transformation by example [65]. Moreover, Chioaşcă [20] suggests to use machine learning for natural language processing of textual specification documents to transform the textual representation into Object System Models.

There exist several works that apply machine learning techniques to try and predict the performance of applications. Venkataraman et al. [68] introduce Ernest, which is a framework for predicting the performance of cloud applications running on a shared infrastructure. Their goal is to predict the performance of a cloud-based application under different resource configurations to help choosing an optimal configuration. They argue that to predict the performance on large data sets and clusters, models can be trained on the same jobs for small sample data sets. Their evaluation of the framework based on a number of tests shows that the performance prediction is within 12% of the actual runtime and only requires a training overhead of 5% even for long-running applications.

Comparably, the authors of [48] present machine learning based performance prediction approaches for cloud applications on Apache Spark. Their approaches are based on gray-box and black-box supervised machine learning models. To evaluate the cost benefit of these approaches they compare it to Ernest on different scenarios, configurations and application workloads. Based on their evaluations the authors conclude that Ernest was able to achieve good results on regular test sets. But it was outperformed by their black-box based models when applications exhibited irregular patterns. Moreover their tests also indicate that no single approach was able to outperform all other approaches in all scenarios.

Another work that focuses on predicting the performance of parallel applications is presented in [38]. Their approach employs multilayer neural networks to predict the performance of SMG2000 applications. They evaluated their neural networks using two large-scale parallel platforms and were able to predict their performance within 5-7% accuracy. Similar to [48] their approach is independent from application internal details which they argue makes it easy to use. In [36], the authors show that machine learning-based empirical hardness models can not only be used to predict the performance of deterministic search algorithms but can also quite accurately predict the performance of non-deterministic, randomized methods based on stochastic local search. Furthermore they demonstrate that the machine learning models can be used to adjust and optimize the analyzed algorithms through adjusting their parameter inputs. These optimizations are evaluated empirically to provide evidence that the adjustments never reduced the performance and sometimes even improved it over standard parameter settings.

A different approach than machine learning for predicting performance of a system is simulation. The authors of [7] introduce a metamodel, called the Palladio Component Model, designed to support the prediction of extra-functional properties of a system such as performance. They provide a proof-of-concept case study evaluation to demonstrate that simulations based on the Palladio Component Model can be used to predict system performance. While the case study reveals shortcomings of the approach, it also shows that performance predictions are possible and can be sufficient for evaluating architectural design decisions.

The performance of model transformations is influenced by many different factors. One factor that has seen a lot of focus is the transformation engine. There exist several works [5, 14, 24, 67, 70] that try to improve engine performance. These works mainly focus on optimizing the search plans for graph-based model transformation languages which are generated from the defined model transformations in an effort to reduce the amount of time necessary to find matchings within the input model. For rule based transformation languages there exist works which try to improve the performance by optimizing the order of application of transformation rules within a transformation [22, 23]. Moreover, there are several publications that try to improve performance by optimizing the transformation code. In [63] the authors use experiments on ATL and QVT to demonstrate that the transformation code definitions can heavily influence the performance during runtime. Their results are mirrored in the work presented in [49] where the authors suggest that performance increases up to 70% are possible with manual optimizations of the transformation code. The authors of [62] present a selection of bad smells in Henshin transformation code which negatively influence the performance of the executed transformations. A similar approach is detailed in [18] where the authors present guidelines, developed through experimentation for writing model transformations that help improve execution performance. Lastly, when the performance of a system is not satisfactory, profilers are often used to detect bottlenecks during execution. Unfortunately there is a lack of profilers for many model transformation languages. Currently only ATL [55] and Henshin [31] have fully developed and usable profilers.

7 CONCLUSION AND FUTURE WORK

We conducted several experiments to investigate which combination of approach and feature set is suitable to predict the execution time of an ATL transformation based on input model characteristics. We have examined linear regression (LR), random forest (RF), and support vector regression using a radial basis function kernel (SVR). We have defined and compared different feature sets, which contain different detailed information about the input models.

We can conclude that the SVR approach is the most suitable of the approaches examined, as it provides accurate predictions (cf. Tables 3 and 4). The SVR approach provides the best predictions based on a feature set resulting from feature selection, but using a feature set which contains the number of model elements, number of references and number of attributes also provides decent results with much less effort to describe a model. Additionally, we showed one weakness of our approach, which is not able to predict the execution time for operations which are not connected to the structural and type information of the input models in our feature sets (cf. *ATL2Tracer*).

In order to overcome the presented weakness of our approach we aim to focus on how we can handle attributes, whose values can have an arbitrary unlimited size and occur randomly without a connection to the features in our feature sets. Additionally, we want to examine whether we can use artificially generated models to train the SVR approach. Generated models could improve the quality of the prediction approach and its usability, since it is then very easy to generate extremely large amounts of training data.

ACKNOWLEDGMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Ti 803/4-1. The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

REFERENCES

[1] 2008. EMF: Eclipse Modeling Framework. xxix, 704 p.
[2] 2022. Java Microbenchmark Harness (JMH). https://github.com/openjdk/jmh Accessed: 19.10.2022.
[3] Chris Albon. 2018. Machine learning with Python cookbook: practical solutions from preprocessing to deep learning. https://learning.oreilly.com/library/view/-/9781491989371/?ar 1 online resource (1 volume).
[4] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. 2007. UML2Alloy: A Challenging Model Transformation. In *Model Driven Engineering Languages and Systems*, Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 436–450. https://doi.org/10.1007/978-3-540-75209-7_30
[5] Gernot Veit Batz, Moritz Kroll, and Rubino Geiß. 2007. A first experimental evaluation of search plan driven graph pattern matching. In *International Symposium on Applications of Graph Transformations with Industrial Relevance*. Springer, 471–486. https://doi.org/10.1007/978-3-540-89020-1_32
[6] Paul Baybutt. 2003. Major hazards analysis: An improved method for process hazard analysis. *Process Safety Progress* 22, 1 (2003), 21–26. https://doi.org/10.1002/prs.680220103
[7] Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2007. Model-based performance prediction with the palladio component model. In *Proceedings of the 6th international workshop on Software and performance*. 54–65. https://doi.org/10.1145/1216993.1217006
[8] Peter Bellmann and Friedhelm Schwenker. 2020. Automated Pain Assessment: Is it Useful to Combine Person-Specific Data Samples?. In *SSCI*. IEEE, 1588–1593. https://doi.org/10.1109/SSCI47803.2020.9308279
[9] Peter Bellmann, Patrick Thiam, and Friedhelm Schwenker. 2018. *Multi-classifier-Systems: Architectures, Algorithms and Applications*. Springer International Publishing, Cham, 83–113. https://doi.org/10.1007/978-3-319-89629-8_4
[10] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (2013), 1798–1828. https://doi.org/10.1109/TPAMI.2013.50
[11] Daniel Berrar. 2019. Cross-Validation. In *Encyclopedia of Bioinformatics and Computational Biology*, Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian SchÃűnbach (Eds.). Academic Press, Oxford, 542–545. https://doi.org/10.1016/B978-0-12-809633-8.20349-X
[12] Hemant K Bhargava, Ramayya Krishnan, and Rudolf Müller. 1997. Electronic commerce in decision technologies: a business cycle analysis. *International Journal of Electronic Commerce* 1, 4 (1997), 109–127. https://doi.org/10.1080/10864415.1997.11518297
[13] André B. Bondi. 2014. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice.* Pearson Education.
[14] Artur Boronat. 2018. Expressive and efficient model transformation with an internal DSL of Xtend. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.* 78–88. https://doi.org/10.1145/3239372.3239386
[15] Alberto Boschetti and Luca Massaron. 2016. *Python Data Science Essentials.* Packt Publishing.
[16] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324
[17] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth.
[18] Roberto Bruni and Alberto Lluch Lafuente. 2011. Evaluating the performance of model transformation styles in maude. In *International Workshop on Formal Aspects of Component Software*. Springer, 79–96. https://doi.org/10.1007/978-3-642-35743-5_6
[19] Surajit Chaudhuri and Vivek Narasayya. 1998. AutoAdmin "'what-if"' index analysis utility. *ACM SIGMOD Record* 27, 2 (1998), 367–378. https://doi.org/10.1145/276305.276337
[20] Erol-Valeriu Chioaşcă. 2012. Using machine learning to enhance automated requirements model transformation. In *2012 34th International Conference on Software Engineering (ICSE)*. 1487–1490. https://doi.org/10.1109/ICSE.2012.6227055
[21] Eclipse. 2020. ATL Transformations. https://www.eclipse.org/atl/atlTransformations/ Accessed: 19.10.2022.
[22] Martin Fleck, Javier Troya, and Manuel Wimmer. 2015. Marrying search-based optimization and model transformation technology. *Proc. of NasBASE* (2015), 1–16.
[23] Lars Fritsche, Erhan Leblebici, Anthony Anjorin, and Andy Schürr. 2017. A Look-Ahead Strategy for Rule-Based Model Transformations. In *MODELS (Satellite Events)*. 45–53.
[24] Holger Giese, Stephan Hildebrandt, and Andreas Seibel. 2009. Improved flexibility and scalability by interpreting story diagrams. *Electronic Communications of the EASST* 18 (2009). https://doi.org/10.14279/tuj.eceasst.18.268
[25] Inc. GitHub. 2021. Search. https://docs.github.com/en/rest/reference/search Accessed: 19.10.2022.
[26] Matteo Golfarelli, Stefano Rizzi, and Andrea Proli. 2006. Designing What-If Analysis: Towards a Methodology. In *Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP (DOLAP '06)*. Association for Computing Machinery, New York, NY, USA, 51–58. https://doi.org/10.1145/1183512.1183523
[27] Stefan Götz, Raffaela Groner, and Matthias Tichy. 2020. Claimed Advantages and Disadvantages of (dedicated) Model Transformation Languages: A Systematic Literature Review. *Software and Systems Modeling* (2020), 1–35. https://doi.org/10.1007/s10270-020-00815-4
[28] Raffaela Groner, Luis Beaucamp, Matthias Tichy, and Steffen Becker. 2020. An Exploratory Study on Performance Engineering in Model Transformations. In *MODELS'20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20)*. Association for Computing Machinery, New York, NY, USA, 308–319. https://doi.org/10.1145/3365438.3410950
[29] Raffaela Groner, Luis Beaucamp, Matthias Tichy, and Steffen Becker. 2020. An Exploratory Study on Performance Engineering in Model Transformations: Data of the mixed method study. https://doi.org/10.18725/OPARU-32365
[30] Raffaela Groner, Peter Bellmann, Stefan Höppner, Patrick Thiam, Friedhelm Schwenker, and Matthias Tichy. 2023. Data Set for Predicting the Performance of ATL Model Transformations. https://doi.org/10.5281/zenodo.7597582
[31] Raffaela Groner, Sophie Gylstorff, and Matthias Tichy. 2020. A profiler for the matching process of henshin. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–5. https://doi.org/10.1145/3417990.3422000
[32] Raffaela Groner, Katharina Juhnke, Stefan Götz, Matthias Tichy, Steffen Becker, Vijayshree Vijayshree, and Sebastian Frank. 2021. A Survey on the Relevance of the Performance of Model Transformations: Data of the Participant Search and the Questionnaire. https://doi.org/10.18725/OPARU-38188
[33] Raffaela Groner, Matthias Tichy, and Steffen Becker. 2018. Towards Performance Engineering of Model Transformation. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 33–36. https://doi.org/10.1145/3185768.3186305
[34] Herodotos Herodotou and Shivnath Babu. 2011. Profiling, What-If Analysis, and Cost-Based Optimization of MapReduce Programs. *Proceedings of the VLDB Endowment* 4, 11 (Aug. 2011), 1111–1122. https://doi.org/10.14778/3402707.3402746
[35] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. 2003. A practical guide to support vector classification.
[36] Frank Hutter, Youssef Hamadi, Holger H Hoos, and Kevin Leyton-Brown. 2006. Performance prediction and automated tuning of randomized and parametric algorithms. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 213–228. https://doi.org/10.1007/11889205_17
[37] Rob J Hyndman and Yanan Fan. 1996. Sample quantiles in statistical packages. *The American Statistician* 50, 4 (1996), 361–365.
[38] Engin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A McKee. 2005. An approach to performance prediction for parallel applications. In *European Conference on Parallel Processing*. Springer, 196–205. https://doi.org/10.1007/11549468_24
[39] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. 2008. ATL: A Model Transformation Tool. *Science of Computer Programming* 72, 1-2 (2008), 31–39. https://doi.org/10.1016/j.scico.2007.08.002
[40] Frédéric Jouault and Jean Bézivin. 2006. KM3: a DSL for Metamodel Specification. In *International Conference on Formal Methods for Open Object-Based Distributed Systems*. Springer, 171–185. https://doi.org/10.1007/11768869_14
[41] Alexander Jung. 2022. Machine Learning: The Basics. http://dx.doi.org/10.1007/978-981-16-8193-6 1 Online-Ressource(XVII, 212 p. 77 illus., 42 illus. in color.).
[42] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 2012. Model transformation by-example: a survey of the first wave. In *Conceptual modelling and its theoretical foundations*. Springer, 197–215. https://doi.org/10.1007/978-3-642-28279-9_15
[43] Stefan Kögel and Matthias Tichy. 2018. Dataset of EMF Models from Eclipse Projects. https://doi.org/10.18725/OPARU-9850
[44] Miroslav Kubat. 2021. An Introduction to Machine Learning. http://dx.doi.org/10.1007/978-3-030-81935-4 1 Online-Ressource(XVIII, 458 p. 114 illus., 5 illus. in color.).
[45] Ludmila I. Kuncheva. 2014. *Combining Pattern Classifiers: Methods and Algorithms.* John Wiley & Sons.
[46] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nat.* 521, 7553 (2015), 436–444. https://doi.org/10.1038/nature14539
[47] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2018. Model-based engineering in the embedded systems domain: an industrial

survey on the state-of-practice. *Software and Systems Modeling* 17, 1 (2018), 91–113. https://doi.org/10.1007/s10270-016-0523-3

[48] Alexandre Maros, Fabricio Murai, Ana Paula Couto da Silva, Jussara M Almeida, Marco Lattuada, Eugenio Gianniti, Marjan Hosseini, and Danilo Ardagna. 2019. Machine learning for performance prediction of spark cloud applications. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 99–106. https://doi.org/10.1109/CLOUD.2019.00028

[49] Tamás Mészáros, Gergely Mezei, Tihamér Levendovszky, and Márk Asztalos. 2010. Manual and automated performance optimization of model transformation systems. *International Journal on Software Tools for Technology Transfer* 12, 3 (2010), 231–243. https://doi.org/10.1007/s10009-010-0151-0

[50] Arnaud De Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. 2016. Mean Absolute Percentage Error for regression models. *Neurocomputing* 192 (2016), 38–48. https://doi.org/10.1016/j.neucom.2015.12.114

[51] (OMG) Object Management Group. 2016. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.3. https://www.omg.org/spec/QVT/1.3/PDF Accessed: 19.10.2022.

[52] David Olive. 2017. *Linear Regression.* Springer International Publishing. https://doi.org/10.1007/978-3-319-55252-1

[53] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[54] Andrew S. Philippakis. 1988. Structured what if analysis in DSS models. In *[1988] Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences. Volume III: Decision Support and Knowledge Based Systems Track*, Vol. 3. 366–370. https://doi.org/10.1109/HICSS.1988.11929

[55] William Piers. 2010. ATL 3.1–Industrialization improvements. In *Proceedings of the 2nd International Workshop on Model Transformation with ATL. Citeseer*. Citeseer.

[56] Pariwat Prathanrat and Chantri Polprasert. 2018. Performance Prediction of Jupyter Notebook in JupyterHub using Machine Learning. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Vol. 3. 157–162. https://doi.org/10.1109/ICIIBMS.2018.8550030

[57] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. 2019. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: data mining and knowledge discovery* 9, 3 (2019), e1301. https://doi.org/10.1002/widm.1301

[58] Hyatt Saleh. 2020. *The Deep Learning with PyTorch Workshop: Build deep neural networks and artificial intelligence applications with PyTorch.* Packt Publishing.

[59] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. 2017. Henshin: A Usability-Focused Framework for EMF Model Transformation Development. In *Proceedings of the 10th International Conference on Graph Transformation (ICGT'17)*. Springer, 196–208. https://doi.org/10.1007/978-3-319-61470-0_12 Lecture Notes in Computer Science, vol 10373.

[60] Hariom Tatsat, Sahil Puri, and Brad Lookabaugh. 2020. *Machine Learning and Data Science Blueprints for Finance.* O'Reilly Media.

[61] Patrick Thiam, Viktor Kessler, Mohammadreza Amirian, Peter Bellmann, Georg Layher, Yan Zhang, Maria Velana, Sascha Gruss, Steffen Walter, Harald C. Traue, Daniel Schork, Jonghwa Kim, Elisabeth André, Heiko Neumann, and Friedhelm Schwenker. 2019. Multi-modal Pain Intensity Recognition based on the SenseEmotion Database. *IEEE Transactions on Affective Computing* 12, 3 (2019), 743–760. https://doi.org/10.1109/TAFFC.2019.2892090

[62] Matthias Tichy, Christian Krause, and Grischa Liebel. 2013. Detecting Performance Bad Smells for Henshin Model Transformations. *Amt@ models* 1077 (2013).

[63] Marcel Van Amstel, Steven Bosems, Ivan Kurtev, and Luís Ferreira Pires. 2011. Performance in model transformations: experiments with ATL and QVT. In *International Conference on Theory and Practice of Model Transformations*. Springer, 198–212. https://doi.org/10.1007/978-3-642-21732-6_14

[64] Vladimir Vapnik. 2013. *The nature of statistical learning theory.* Springer science & business media.

[65] Dániel Varró. 2006. Model transformation by example. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 410–424. https://doi.org/10.1007/11880240_29

[66] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. 2016. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Software & Systems Modeling* 15, 3 (2016), 609–629. https://doi.org/10.1007/s10270-016-0530-4

[67] Gergely Varró, Frederik Deckwerth, Martin Wieber, and Andy Schürr. 2012. An algorithm for generating model-sensitive search plans for EMF models. In *International Conference on Theory and Practice of Model Transformations*. Springer, 224–239. https://doi.org/10.1007/978-3-642-30476-7_15

[68] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 363–378.

[69] Vijayshree Vijayshree, Markus Frank, and Steffen Becker. 2020. Extended Abstract of Performance Analysis and Prediction of Model Transformation. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 8–9. https://doi.org/10.1145/3375555.3384935

[70] Attila Vizhanyo, Aditya Agrawal, and Feng Shi. 2004. Towards generation of efficient transformations. In *International Conference on Generative Programming and Component Engineering*. Springer, 298–316. https://doi.org/10.1007/978-3-540-30175-2_16

[71] Kris Welsh, Nelly Bencomo, Pete Sawyer, and Jon Whittle. 2014. *Self-Explanation in Adaptive Systems Based on Runtime Goal-Based Models.* Springer Berlin Heidelberg, Berlin, Heidelberg, 122–145. https://doi.org/10.1007/978-3-662-44871-7_5

[72] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering.* Springer, Berlin. https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6314538 1 Online-Ressource (249 pages).