https://doi.org/10.1016/j.resconrec.2019.104606

Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning

Yangyang Shu
School of Computer Science, University of Technology Sydney
Sydney, NSW, Australia
Yangyang.Shu@student.uts.edu.au

Yulei Sui
School of Computer Science, University of Technology Sydney
Sydney, NSW, Australia
yulei.sui@uts.edu.au

Hongyu Zhang
School of Electrical Engineering and Computing,
The University of Newcastle, Australia
Newcastle, NSW, Australia
hongyu.zhang@newcastle.edu.au

Guandong Xu*
School of Computer Science, University of Technology Sydney
Sydney, NSW, Australia
Guandong.Xu@uts.edu.au

ABSTRACT

Context: Many software systems are highly configurable. Different configuration options could lead to varying performances of the system. It is difficult to measure system performance in the presence of an exponential number of possible combinations of these options.
Goal: Predicting software performance by using a small configuration sample.
Method: This paper proposes PERF-AL to address this problem via adversarial learning. Specifically, we use a generative network combined with several different regularization techniques (L1 regularization, L2 regularization and a dropout technique) to output predicted values as close to the ground truth labels as possible. With the use of adversarial learning, our network identifies and distinguishes the predicted values of the generator network from the ground truth value distribution. The generator and the discriminator compete with each other by refining the prediction model iteratively until its predicted values converge towards the ground truth distribution.
Results: We argue that (i) the proposed method can achieve the same level of prediction accuracy, but with a smaller number of training samples. (ii) Our proposed model using seven real-world datasets show that our approach outperforms the state-of-the-art methods. This help to further promote software configurable performance.
Conclusion: Experimental results on seven public real-world datasets demonstrate that PERF-AL outperforms state-of-the-art software performance prediction methods.

CCS CONCEPTS

• **Software and its engineering** → **Software performance**.

KEYWORDS

Software performance prediction, adversarial learning, regularization, configurable systems

ACM Reference Format:
Yangyang Shu, Yulei Sui, Hongyu Zhang, and Guandong Xu*. 2020. Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning. In *ESEM '20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3382494.3410677

1 INTRODUCTION

Many large and complex software systems, such as DBMS, compilers, and web servers, are highly configurable. They provide many configuration options for users to select/deselect. User-relevant configuration options are also called features [2], [3], [4], [18] in software product line context. Different combinations of configuration options could lead to different quality attributes. Among these quality attributes, performance (e.g., response time or throughput) is one of the most important attributes, because it directly affects user experience and cost [34].

It is important to find a good configuration to meet a specific performance requirement. However, highly configurable software systems usually have a large number of configuration options (e.g., a compiler normally has hundreds of optimization options for compiling a program), resulting in an exponential number of possible option combinations. For example, 10 binary options can produce 2^{10} configuration possibilities. For the numeric option inputs, the configuration settings become even more complicated. It follows naturally that there is a demand in developing automatic models for performance prediction.

***Existing Efforts and Limitations*.** Software performance prediction can be treated as a regression problem in machine learning. Over years, many software performance prediction methods have been proposed. For example, *CART* [10] proposes to use the Classification and Regression Trees (CART) algorithm to model the correlation between features and performance. Guo et al. [12] and Westermann et al. [34] use statistical sampling and machine learning approaches for configurable software performance prediction.

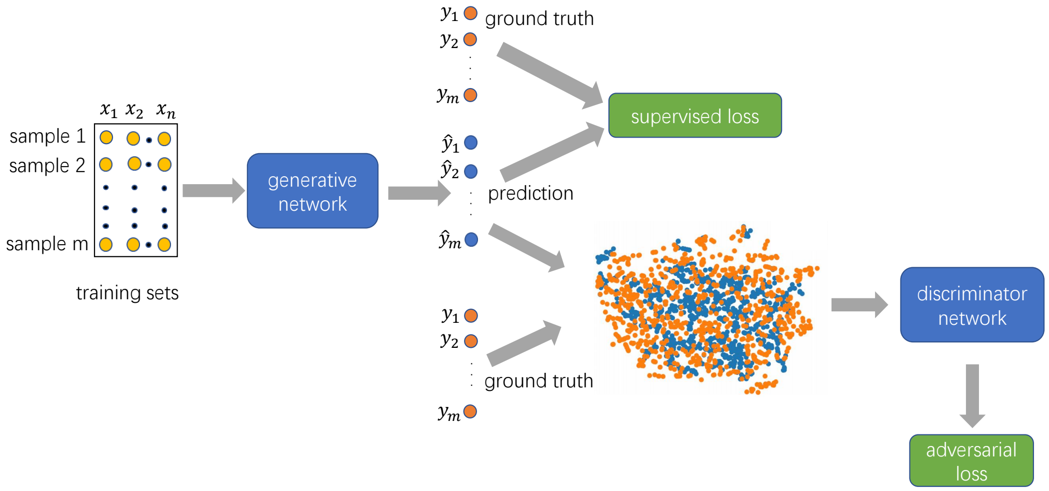


Figure 1: An overview of our proposed configurable software through adversarial learning.

Siegmund et al. [28] introduce a measurement-based prediction approach, called *SPLConqueror*, which aims to learn the influences of individual configuration options and their interactions from the differences among the measurements of the samples. However, the above mentioned approaches usually require a large amount of training samples [12, 24, 30], and the prediction accuracy relies heavily on high-quality samples, which are always limited and very costly to measure and collect.

Insights and Our Solution. The key challenge for achieving more accurate performance prediction for a large-scale system lies in the limited size of samples. This paper aims to address this challenge by using a small set of samples to achieve the comparable level of prediction accuracy via adversarial learning, a recently emerged machine learning approach.

Particularly, a new performance prediction approach, PERF-AL, is proposed for configurable software taking the adversarial learning idea used in GAN (generative adversarial network) algorithm. We train (1) a generator network to generate optimal inputs to a software system (which is actually a nonlinear dynamic system), and (2) a discriminator network as a critic that checks the optimality of the performance prediction. The generator and discriminator networks compete with each other by iteratively refining and adapting the prediction model, resulting in a better prediction model than those by other machine learning approaches.

Figure 1 illustrates the framework of our Perf-AL model. It consists of both generator and discriminator networks. The generator network is a deep neural network, with its first layer taking the input, the last layer generating the output, and the middle layers as hidden layers connecting the input and output layer. In discriminator network, the inputs are prediction labels obtained by the generator network as well as the ground truth, with the sign label 0 or 1 as a judgment. The discriminator tries to distinguish which inputs from the mixed data are closer to the ground truth. Through adversarial competing and adjusting, the distribution is updated to further regularize the predicted values and real values. The combination of supervised loss and adversarial loss is then jointly optimized for performance prediction. Therefore, our proposed approach is able to train a performance prediction model using the adversarial learning mechanism with a small sample set. In order to overcome overfitting, which is often encountered in training with a small size of samples, various regularization techniques, e.g. L1, L2 regularization, and dropout technique are integrated into the proposed PERF-AL network. We summarize our main contributions as follows:

- We propose a novel adversarial model for performance prediction with various regularization techniques.
- Our PERF-AL model can achieve the same level of prediction accuracy, but with a smaller number of training samples.
- We have validated our proposed model using seven real-world datasets. The experimental results show that our approach outperforms the state-of-the-art methods.

Organization. The rest of this paper is organized as follows: Section 2 briefly introduces the background on configurable software, regularization and generative adversarial network. Section 3 shows our proposed software performance prediction model with deep generative adversarial network, giving the details in our proposed generator and discriminator networks. Section 4 shows the evaluation results and analysis of our framework by comparing with the state-of-the-art approaches using seven datasets. Section 5 discusses related work and our paper is concluded in Section 6.

2 PRELIMINARIES

2.1 Problem Formulation

In this section, we first formalize the problem of performance prediction for configurable software as a regression problem. The mapping from any configuration options of a system with n features to its performance values can be formalized as:

$$f(x) = f(x_1, x_2, ..., x_n) : \mathbb{X} \to \mathbb{R}. \tag{1}$$

Table 1: Example of a configurable software system and its performance values

x_1	x_2	x_3	...	x_8	x_9	x_{10}	x_{11}	$f(x)$
0	0	0	...	1	10	5	12	y_1
0	1	0	...	1	20	1	3	y_2
1	0	1	...	0	12	3	11	y_3
1	1	0	...	1	9	7	12	y4
.	.	.	...	.	.	.	.	.
1	0	0	...	0	55	0	5	y_{m-1}
1	0	0	...	0	53	3	2	y_m

where $\mathbb{X}$ is the Cartesian product of the domains of all the configuration options. x_i ($i = 1, 2, \ldots, n$) is the variable that stores the value of the configuration option i^{th}. It can be either a Boolean value (indicating whether a configuration option is selected or not) or a real value in the value range of the configuration option. If $\mathbb{X}$ are all Boolean values, Eq. 1 can be simplified as a Boolean function:

$$f : \{0, 1\}^n \to \mathbb{R} \tag{2}$$

The objective is to predict the software performance value $f(\mathbf{x})$ of any new configuration vector $\mathbf{x}$ given a small sample m: $\{\mathbf{x}_i, f(\mathbf{x}_i)\}$, $i = 1, 2, \ldots, m$.

Table 1 gives an example of a software performance function $f(x_1, ..., x_n)$ and its configuration space. This software system has 11 configuration options, in which 8 options take binary values and 3 options take numeric values. Measuring the time performance of all possible configurations of the system is difficult and costly, demanding a lot of time and effort. To address this issue, researchers propose to measure only the performance values of a limited number of configurations (samples), then build a prediction model from these training (configurations) data to predict the performance values of all other possible configurations. The challenge here is to use a small yet effective sample to predict for all other configurations with high accuracy.

2.2 Dropout, L1 and L2 Regularization

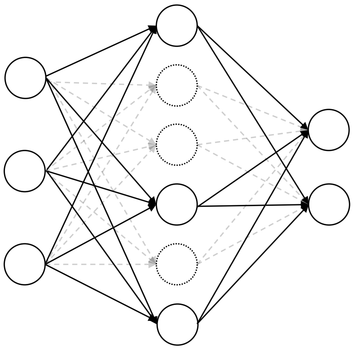


Figure 2: The method of Dropout.

When building a prediction model, overfitting may happen during training, especially when the parameters of the model are numerous and the training sample size is small. Specifically, Overfitting could lead to small training error but large test error and low prediction accuracy. In our work, because we expect to use a small quantity of training data to predict performance values of highly configurable software systems and at the same time deep neural networks contain a large number of parameters, overfitting is an important issue to be solved. To address this issue, we adopt dropout, which was first introduced by Hinton [14] for neural networks. The key idea is to randomly drop some units (along with their connections) from the neural network during training, as illustrated in Figure. 2.

L1 regularization [31] and L2 [23] regularization are other two widely used techniques for norm in order to create less complex (parsimonious) models when we have a high population of features in the datasets. A regression model that uses L1 regularization technique is labelled Least Absolute Shrinkage and Selection Operator (Lasso) Regression and model which uses L2 is known as Ridge Regression. Lasso Regression adds "absolute value of magnitude" of coefficient as a penalty term to the loss function and Ridge Regression adds "squared magnitude" of coefficient as the penalty term to the loss function. Notably, it has been frequently observed that L1 regularization can cause many parameters to be equal to zero, which makes the parameter vector sparse [23]. Nowadays, L1 regularization and L2 regularization are arguably the most popular technique in machine learning to combat overfitting not only in linear regression but also in other models, including neural network [7, 32].

2.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) was first proposed by Goodfellow [9] and is now widely applied in many areas such as computer vision and image processing. This is a class of machine learning systems in which two neural networks contest with each other in a zero-sum game framework. The generative network learns to map from a latent space to a data distribution of interest, while the discriminator network distinguishes candidates produced by the generator from the true data distribution. The generative network's training objective is to increase the error rate of the discriminator network, i.e., "fool" the discriminator network by producing novel candidates that the discriminator identifies as not synthesized (are part of the true data distribution) [9]. Specifically, this framework includes two models simultaneously trained: a generative model G that captures the data distribution, and a discriminator model D that estimates the probability that a sample came from the training data. Through an adversarial process, we iteratively evaluate the generative models to determine which one is the best for fitting real data distribution. The objective function for jointly training the prediction model from two networks is:

$$\min_{\theta_\mathrm{g}} \max_{\theta_\mathrm{d}} [\mathbb{E}_{x \sim Pdata} \mathrm{log} \mathrm{D}_{\theta_\mathrm{d}}(\mathrm{x}) + \mathbb{E}_{\mathrm{z} \sim \mathrm{p(z)}} \mathrm{log}(1 - \mathrm{D}_{\theta_\mathrm{d}}(\mathrm{G}_{\theta_\mathrm{g}}(\mathrm{z})))], \tag{3}$$

where $D_{\theta_d}(x)$ is Discriminator outputting for real data x, $D_{\theta_d}(G_{\theta_g}(z))$ is Discriminator outputting for generated predicted data G(z). In this joint model:

- *Generator network:* Try to fool the discriminator by generating real-looking labels. Specifically, Discriminator (θ_d) wants to maximize the objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (prediction).

- *Discriminator network:* Try to distinguish between the real and predicted label. Specifically, Generator (θ_g) wants to minimize objectives such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real).

3 A DEEP NEURAL NETWORK VIA ADVERSARIAL LEARNING FOR SOFTWARE PERFORMANCE PREDICTION

In this section, we describe in detail the proposed deep neural network and adversarial learning based framework for modeling the performance of configurable software systems. The framework is shown in Figure 1, which consists of two major components: the generative network and the adversarial network (discriminator network). The generative network outputs the predicted values of function $f(x)$ and tries to use its value to "fool" the adversarial network, which tries to distinguish between the predicted and the real labels. Through this neural network architecture, we can build a software performance predication model with a small sample but can still achieve a high prediction accuracy.

3.1 Objective

Denote $S = \{(\mathbf{x_i}, y_i)|i = 1, ..., m\}$, where $\mathbf{x_i} \in \mathbb{R}^n$ denotes the value of the configure option. Each of them can be binary $\{0, 1\}$ or numeric, $y_i \in \mathbb{R}$ denotes the real value with any configuration options. m is the number of training samples. The goal is to learn a deep neural network $f : \mathbb{R}^n - > \mathbb{R}^1$, which can predict the software performance value $f(x)$.

3.2 The Design of Generator Network

The generator network is a regression network that is a performance prediction model consisting of the input of neurons, activation functions, hidden layers and output. The main hyperparameters in generator network include the depth (the number of hidden layers), the width (the number of neurons per layer) and activation functions, which determine the model structure or decide how the network is trained.

The width and the depth are two key components in the design of a neural network architecture. Width and depth are both important and should be carefully tuned together for the best performance of neural networks [20]. Going deeper will make a network more expressive, which means it can capture variations of the data better. This is known to yield expressiveness more efficiently than increasing the width of hidden layers. Against this, the trade-off for more expressiveness is always the increased tendency to overfit the training data, inferring that more data or additional regularization will be needed. Hence, the neural network should be formed as deeply as the training data allows. The depth can be determined by experiments [5].

Activation functions are another really important element for a deep neural network to learn really complicated and non-linear complex functional mappings between the inputs and response variables. They introduce non-linear properties to our network. Their main purpose is to convert an input signal of a node in a deep neural network to an output signal. That output signal is now used as an input in the next layer in the network stack.

The architecture of the generator network in Figure 3 for software performance prediction is as follows:

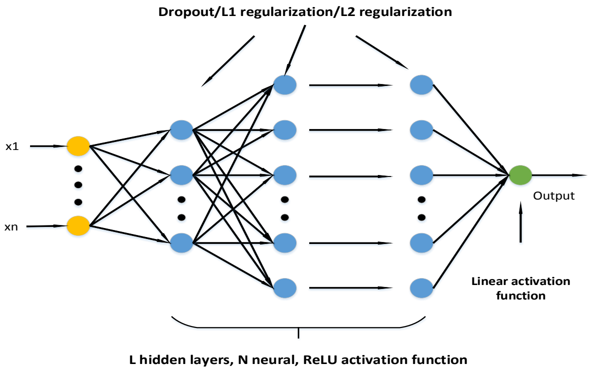

Dropout/L1 regularization/L2 regularization
x1
xn
Output
Linear activation function
L hidden layers, N neural, ReLU activation function

Figure 3: The proposed generator network with dropout or L1/L2 regularization for configurable software performance prediction. The inputs of the network are the n configuration options of the software system and the output of the network is the performance value.

- The input layer has n neurons, where n is the number of configuration options of the software system to be predicted. The output layer has 1 neuron, which outputs the performance value of the software system.
- There are L hidden layers ($L \geq 2$) and i^{th} hidden layer has N_i number of neurons.
- All the hidden layers use a ReLU activation function while the output layer uses a linear activation function. Using a linear output layer is required since the performance prediction is a regression problem. Meanwhile, the ReLU is chosen as the activation function of the hidden layers due to its ability to learn much faster in networks with many layers compared to other non-linear activation functions[7, 21].

Formally, we build a generative network $\hat{y} = G(\mathbf{x}; \theta_G)$. The objective function of the generative network is defined as follows:

$$J(\theta_G) = L(\theta_G) + \frac{\lambda}{M}\sum_{i=1}^{M} L(\hat{y}_i, y_i), \tag{4}$$

where

$$L(\theta_G) = \begin{cases} 0 & dropout \\ ||\theta_G||_1 & L_1\ regularization \\ ||\theta_G||_2 & L_2\ regularization, \end{cases} \tag{5}$$

$$L(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2 \tag{6}$$

λ denotes the weight hyperparameters.

3.3 The Design of Discriminator Network

Inspired by the framework of GANs [9], we explore label dependencies from ground truth labels by adversarial learning. We introduce a discriminator $y_D = D(\mathrm{y}; \theta_D)$ to distinguish the predicted values from real labels. For the generator network, we keep the supervised learning objective. In addition, we want the generative network to make predictions which "fool" the discriminator, leading to a new adversarial learning objective. Under these two learning objectives, the generator network will be expected to output predictions, which minimize the supervised loss.

Algorithm 1 Training algorithm of the proposed model

```
Input:
  training sample $D = (\mathbf{x_i}, y_i)$,
  coefficient $\alpha$, learning rate $\eta_1$ and $\eta_2$
  the number of steps of updating adversarial network $K_1$, the number of steps of updating generative networks $K2$, bitch size l,
Output: the generative network $G$
  Randomly initialize parameters of Model parameters $\theta_D$ and $\theta_G$;
  for number of training iterations do
    for $K1$ steps do
      Sample a mini-batch of l training data $\mathbf{x_i}_{i=1}^{l}$ from $\{1, ..., m\}$
      Sample a mini-batch of l real labels $y_i{}_{i=1}^{l}$ from $\{1, ..., m\}$
      Update the adversarial network $D$ by gradient descent $\theta_D := \theta_D - \eta_1 \frac{\partial J_D(\theta_D)}{\partial \theta_D}$
    end for
    for $K2$ steps do
      update the adversarial network $G$ by gradient descent $\theta_G := \theta_G - \eta_2 \frac{\partial J_G(\theta_G)}{\partial \theta_G}$
    end for
  end for
```

Specifically, the learning objective of the framework can be written as follows:

$$\begin{aligned} \min_{\theta_{\mathrm{G}}} \max_{\theta_{\mathrm{D}}} \frac{1}{M} \sum_{i=1}^{M} [\mathrm{logD(y_i)} + \mathrm{log(1 - D(G(}\mathbf{x}_{\mathrm{i}}\mathrm{))))}] \\ + \alpha(\frac{\lambda}{M} \sum_{i=1}^{M} L(G(\mathbf{x}_i), y_i) + L(\theta_G)) \end{aligned} \tag{7}$$

where α is a weight coefficient between generator and discriminator networks controlling the proportion of the supervised objective and its adversarial counterpart. Theoretically, if $\alpha = 0$, the learning objective is the same as that of the original GAN. On the contrary, if $\alpha \longrightarrow \infty$, the learning objective is equivalent to that of the single-task network without the assist of a discriminator. λ is the weight coefficient between loss function and regularization in the generator network. As part of training the module, we first initialize these parameters as small random numbers, then conduct model selection with grid search, by choosing these parameters ranging from {0.01, 0.1, 1, 5, 10} for simplicity.

Similar to the optimization procedure of GAN, the learning objective in Eq.7 cannot be optimized directly. The discriminator and the generator network are optimized alternately by fixing their opponents. According to the suggestion in [8], it is better for generative network to minimize -log D(G(x)) instead of log(1-D(G(x))) in order to avoid the flat gradient. Finally, after rewriting the formula, the learning objectives of the discriminator and the generator networks are given as follows:

$$\max_{\theta_{\mathrm{D}}} \frac{1}{\mathrm{M}} \sum_{\mathrm{i=1}}^{\mathrm{M}} [\mathrm{logD(y_i)} + \mathrm{log(1 - D(G(}\mathbf{x}_{\mathrm{i}}\mathrm{))))}] \tag{8}$$

$$\min_{\theta_{\mathrm{G}}} - \frac{1}{M} \sum_{i=1}^{M} [\mathrm{log(D(G(}\mathbf{x}_{\mathrm{i}}\mathrm{))))}] + \alpha(\frac{1}{\mathrm{M}} \sum_{\mathrm{i=1}}^{\mathrm{M}} \mathrm{L(G(}\mathbf{x}_{\mathrm{i}}\mathrm{), y_i) + L(}\theta_{\mathrm{G}}\mathrm{))} \tag{9}$$

For the generator network, since the predicted value and real value are both numeric, we use the mean square errors as the loss function since this is the most frequently used regression loss function in machine learning. However, as for adversarial network, the predicted label and ground truth labels are still numeric while the sign labels we set are binary {0,1}. This allow us to give the additional formula:

$$L_D(\hat{y}, y) = -y\mathrm{log}\hat{\mathrm{y}} - (1 - \mathrm{y})\mathrm{log}(1 - \hat{\mathrm{y}}) \tag{10}$$

where $\hat{y}$ denotes the numeric of the predicted and ground truth labels, y denotes binary{0,1} of sign labels. Since y is binary, this formula just calculates $-y\mathrm{log}\hat{\mathrm{y}}$ or $-(1 - y)log(1 - \hat{y})$ with y being 1 or 0 respectively. When y equals 1, the predicted value $\hat{y}$ is expected to close to 1 in order to minimize L_D loss function. Similarly, when y equals 0, the predicted value $\hat{y}$ is expected to close to 0 in order to minimize L_D loss function. Through this formula we defined, the discriminator network devotes to distinguishing the label from the predicted and ground truth values.

From this it follows that the adversarial network and generative network can be rewritten as:

$$\begin{aligned} J_D(\theta_D) &= \frac{1}{M} \sum_{i=1}^{M} [L_D(D(y_i), 1) + L_D(D(G(\mathbf{x_i})), 0)] \\ &= \frac{1}{M} \sum_{i=1}^{M} -\mathrm{logD(y_i)} - \mathrm{log(1 - D(G(}\mathbf{x}_{\mathrm{i}}\mathrm{)))} \end{aligned} \tag{11}$$

$$\begin{aligned} J_G(\theta_G) &= \alpha(\frac{1}{M} \sum_{i=1}^{M} L(G(\mathbf{x}_i), y_i) + L(\theta_G)) + \frac{1}{M} \sum_{i=1}^{M} L_D(D(G(\mathbf{x_i})), 1) \\ &= \alpha(\frac{1}{M} \sum_{i=1}^{M} (G(\mathbf{x_i} - y_i))^2 + L(\theta_G)) - \frac{1}{M} \sum_{i=1}^{M} \mathrm{logD(G(}\mathbf{x}_{\mathrm{i}}\mathrm{))} \end{aligned} \tag{12}$$

We apply the alternate optimization steps to train model parameters. The learning procedure of our method is shown in Algorithm 1.

3.4 Trade-off through Regularization

In a configurable software system, the combination of configurations could be exponential, which means that there is an infinite number of parameters and samples. It is easy to overfit the network if exhaustively deploying and measuring system performance under all possible configurations. In order to solve the overfitting problem, we introduce additional information to the network by using three suitable regularization techniques, i.e. L1 regularization, L2 regularization and dropout technique.

1) *L1 regularization of the network.* It is known that even though the possible number of interactions among configuration options is exponential, a very large portion of potential interactions has no influence on the performance of software systems [16]. This means that only a small number of parameters have significant impact on the model. In other words, the parameters of the neural network could be sparse. L1 regularization implements feature selection by assigning insignificant input features with zero weight and useful features with a non zero weight. Hence, we can use L1 regularization to satisfy this condition. As show in Eq. 6, the idea of L1 regularization is to add every hidden layer on the parameters.

2) *L2 regularization of the network.* Although L2 regularization cannot produce sparsely, it forces the weights to be small. L2 regularization can judge whether the different features have the different impact on output through allocating different weights to every feature. It is arguably the most popular technique in machine learning used to combat overfitting. In our model, we use L2 regularization in every hidden layer of the parameters. The formula is given in Eq. 6.

3) *Dropout technique of the network.* The deep neural network of software systems with a large number of parameters is indeed a high computational program, resulting in serious overfitting problems. Dropout is also a technique for addressing this problem. It randomly drops units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks simply by using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods [29]. With this benefit, we apply dropout technique in every hidden layer.

After we apply these three regularization techniques and conduct experiments, we select the best regularization technique in our network. According to our experiments (described in Section 4.6), the L2 regularization performs best among these three regularization techniques. Therefore we choose L2 regularization in our PERF-AL network.

4 EXPERIMENTS AND ANALYSIS

To evaluate our proposed approach, we conduct experiments to answer the following research questions:

- **RQ1**: Does our proposed approach improve performance in the prediction of configurable software systems with binary options when compared with state-of-the-art approaches?
- **RQ2**: Does our proposed approach improve the performance prediction of configurable software systems with binary and numeric options when compared with state-of-the-art approaches?
- **RQ3**: What contribution does the adversarial component hold for our proposed model? Specifically, what about the prediction accuracy of deep networks with and without adversarial learning?
- **RQ4**: What kind of regularization should be used to achieve the best performance?

We ask RQ1 and RQ2 to evaluate the effectiveness of our PERF-AL model on binary and binary-numeric configurable software systems and compare it with some state-of-the-art baselines. RQ3 aims to evaluate the contribution of the adversarial component in our model. We ask RQ4 to compare the regularization techniques in deep adversarial network.

4.1 Data Preparation

In evaluating the prediction accuracy of our proposed method, we use four datasets with binary configuration options and three with both binary and numeric configuration options. These seven datasets are different real-world configurable software systems. They have different sizes (between 45000 and 300000+ lines of code) and are written in various languages, e.g. Java, C and C++. These systems also have different characteristics and are from diverse application domains, e.g. multi-grid solver, web sever, video encoder, database library, database management system, compiler, etc. The number of configuration options ranges from 8 to 39 while the number of valid configurations ranges from 180 to 13485. These software systems were measured and published online [1]. More details about them can be found in [27, 28]. Table 2 gives the overview of these seven subject systems.

Table 2: The subject software systems. #Binary is the number of binary configuration options; #Numeric is the number of numeric configuration options; #Configs is the number of valid configurations

System	Domain	#Binary	#Numeric	#Configs
Apache	Web server	9	0	192
LLVM	Compiler	11	0	1024
BDB-C	Database System	18	0	2560
BDB-J	Database System	26	0	180
DUNE MGS	Multi-Grid solver	8	3	2304
HIPA	Image Processing	31	2	13485
HSMGP	Stencil-Grid Solver	11	3	3456

4.2 Evaluation Metrics

Mean Relative Error (MRE), Mean Square Error (MSE) and Spearman correlation are used to evaluate the prediction accuracy of the proposed method, as they are widely used evaluation metrics in the area of prediction models [6, 12, 17, 27]. MRE is computed as,

$$MRE = \frac{1}{|C|} \sum_{c \in V} \frac{|predicted_c - actual_c|}{actual_c}, \tag{13}$$

where V is the testing dataset, $predicted_c$ is the predicted performance value of configuration c. $actual_c$ is the actual performance value of configuration c. MSE is computed as,

$$MSE = \frac{1}{|C|} \sum_{c \in V} (predicted_c - actual_c)^2, \tag{14}$$

Spearman correlation is computed as,

$$R = \frac{\sum(predicted_c - \overline{predicted_c})(actual_c - \overline{actual_c})}{\sqrt{\sum(predicted_c - \overline{predicted_c})^2 \sum(actual_c - \overline{actual_c})^2}}, \tag{15}$$

where $\overline{predicted_c}$ and $\overline{actual_c}$ are the mean values of $predicted_c$ and $actual_c$ respectively. These three evaluation metrics analyze the correlation between the predicted values and ground truth from different aspects. MRE and MSE measure the precision of a set of predicted values and real values which measure difference between the predicted values and ground truth. Spearman correlation measures the strength and direction of association between predicted values and real values.

4.3 Training Details

In the generator network, the hidden size is set to 3 and the number of neurons per layer is 512. The mini-batch size is set to 64, while the learning rate starts from 0.001 and is divided by 10 when the performance is predicted on the validation sets. The input is normalized between 0 and 1. We set the last output layer with the linear activation. Beforehand, we mapped the software performance values into [0, 1]. In discriminator network, the number of hidden layers is 3 and the number of neurons per layer is 128. The learning rate in the discriminator network starts from 0.0001 and is divided by 10 when the performance is predicted on the validation sets. We train the model using the Adam algorithm [19] as it is a very computationally efficient method. All the experiments in this paper are implemented with Python 3.6, and run on a computer with a 2.2GHz Intel Core i7 CPU, 64GB 1600MHz DDR3 RAM, and a Titan X GPU with 12GB memory, running Red Hat Linux 7.5.

4.4 RQ1: Comparisons on Binary Options

As mentioned in Section I, some existing methods, e.g. SPLC Conqueror [27, 28], Fourier Learning [38], DECART [12] (the improved version of CART [10]), and DeepPerf [15] can predict performance values of software systems with binary options. Among these methods, DECART and DeepPerf are superior to others and DeepPerf is the first to be proposed to use deep network to model highly configurable software systems. It is natural, therefore, in this experiments, we will compare our proposed method with DECART and DeepPerf. We adopt the experiment setup in [12]. Initially, we randomly select five different sizes from these four subject systems respectively and their corresponding performance values for the training datasets: n, $2n$, $3n$, $4n$ and $5n$, where 67% of samples for training and 33% for validation, n is the number of options of each system, shown in the column Binary of Table 2. all the remaining measurements are then used as the testing datasets. To evaluate the consistency and stability of the approaches, for each sample size of each subject system, we repeat this random sampling, training and testing process 30 times. We then report the mean of the MREs obtained after 30 experiments with Perf-AL, DeepPerf and DECART.

Table 3 shows the experiments results of software performance prediction on the Apache, LLVM, BDB-C and BDB-J systems. These four subject systems were also used in [12]. As can be seen, with the increase of sample size, the accuracy of performance prediction is improved, since under-fitting is resolved with enough training data. Furthermore, our proposed Perf-AL method outperforms the DECART and DeepPerf methods. Specifically, on the Apache dataset, the mean relative error achieved by our method is 0.0744, 0.0657, 0.0583 when the sample size is $3n$, $4n$, $5n$, respectively, which leads to 0.81%, 0.4%, and 0.46% improvement over DeepPerf and 3.59%, 2.92%, 2.01% improvement over DECART. On the LLVM and BDB-J datasets, the improvements in the sample size of 3n, 4n, 5n are 0.17%, 0.22%, 0.35% (DeepPerf), 1.59%, 1.49%, 1.20% (DECART) and 0.09%, 0.06%, 0.02% (DeepPerf), 0.39%, 0.11%, 0.08% (DECART), respectively. On the BDB-C dataset, DeepPerf achieves the best performance in $3n$ size and DECART achieves the best performance in $5n$ size. Overall, our Perf-AL model achieves the best performance on all the datasets.

Table 3: Comparisons with baselines on binary configuration options systems.

Subject System	Sample Size	DECART MRE	DeepPerf MRE	Perf-AL MRE
Apache	n	-	**0.1787**	0.2416
	2n	0.1583	**0.1024**	0.1168
	3n	0.1103	0.0825	**0.0744**
	4n	0.0949	0.0697	**0.0657**
	5n	0.0784	0.0629	**0.0583**
LLVM	n	0.0600	0.0509	**0.0443**
	2n	0.0466	0.0387	**0.0313**
	3n	0.0396	0.0254	**0.0237**
	4n	0.0354	0.0227	**0.0205**
	5n	0.0284	0.0199	**0.0164**
BDB-C	n	1.5100	1.3360	**1.3323**
	2n	0.438	0.1677	**1.3756**
	3n	0.3190	**0.1310**	0.1409
	4n	0.0693	0.0695	**0.0691**
	5n	**0.0502**	0.0582	0.0571
BDB-J	n	0.1004	0.0725	**0.0710**
	2n	0.0230	**0.0207**	0.0514
	3n	0.0203	0.0173	**0.0164**
	4n	0.0172	0.0167	**0.0161**
	5n	0.0167	0.0161	**0.0159**

Compared with DECART, our Perf-AL method needs far less training data than the former for the same level of accuracy. For example, on the LLVM dataset, Perf-AL only needs $3n$ samples to achieve a prediction MRE of 0.0237 while DECART needs at least $5n$ samples in order to achieve the same prediction MRE. For DeepPerf, the distribution of the predictions may be far away from that of ground truth due to the lack of distinction between the prediction and the ground truth. Since our proposed method introduces a discriminator to distinguish the predicted values from the ground truth, the generator network tries to "fool" the discriminator network, which forces the distributions of the prediction and the ground truth to be closer. Hence, our Perf-AL model is superior to DECART and DeepPerf.

Table 4: Comparisons with baselines on binary and numeric configuration options systems.

Subject System	Sample Size	SPLConqueror MRE	DeepPerf MRE	Perf-AL MRE
DUNE MGS	49	0.201	0.1573	**0.1451**
	78	0.221	0.1367	**0.1297**
	240	0.106	0.0819	**0.0763**
	375	0.088	0.0720	**0.0649**
HIPA	261	0.142	0.0939	**0.0922**
	528	0.138	0.0638	**0.0621**
	736	0.139	0.0506	**0.0488**
	1281	0.139	0.0375	**0.0371**
HSMGP	77	**0.045**	0.0676	0.0470
	173	0.028	0.0360	**0.0262**
	384	0.022	0.0253	**0.0215**
	480	0.017	0.0224	**0.0168**

4.5 RQ2: Compared to Baseline on Binary-numeric Options

Our next step is to compare our PERF-AL model with SPCLConqueror and DeepPerf, which can all predict performance values of software systems in both binary and numeric options. Three datasets, DUNE MGS, HIPA and HSMGP, as shown in Table 2 will be used for comparison. These subjects systems are also used in [28] and [15] for evaluating SPLConqueror and DeepPerf. We use the same sample sizes as their values for evaluation. To reduce the fluctuations caused by randomness, we repeat the random sample, training, validation and testing process 30 times with each sample size. Then, we report the mean of MREs. The testing dataset consists of all the remaining configurations after selecting the training sample.

The experimental results are shown in Table 4. As can be seen, our proposed PERF-AL method outperforms the SPLConqueror and DeepPerf methods. For example, for the sample size of 240 and 375, our PERF-AL model achieves a MSE of 0.0763 and 0.0649 on DUNE MGS dataset, Both better than the results achieved by the DeepPerf method. This demonstrates that the discriminator plays an important role in distinguishing the predictions from the ground truth and enforces the generator network to generate predictions which are closer to the distribution of the ground truth. Compared with SPLConqueror, our PERF-AL method outperforms in almost all cases, which indicates the superiority of using neural network to predict software performance.

4.6 RQ3: Comparisons with Different Regularization

In this section, we aim to evaluate how different regularization methods have different impacts on prediction values in deep neural networks. After that, we can decide which regularization techniques (L1/L2/Dropout) should be used in the PERF-AL method to achieve the best performance. For this, we design the following experiments in Apache, LLVM, BDB-C and BDB-J four datasets: deep generator network with L1 regularization (**L1-NN**), deep generator network with L2 regularization (**L2-NN**), deep generator network with dropout regularization (**dropout-NN**), generative adversarial networks with the best regularization technique (**PERF-AL**). In fact, we use L2 regularization in our generative adversarial network because it can achieve the highest accuracy among the three regularization techniques.

Table 5 shows the prediction MRE, MSE and R in four binary subject systems with three sample sizes per system. From Table 5, we observe the following:

First, with a smaller sample size, the prediction results are worse due to the fact that the network or classifier is prone to overfit on small training data.

Second, the proposed method using adversarial learning has achieved the best performance among all methods with the lowest MSE, lowest MRE, and highest Spearman correlation. Specifically, compared with deep network without discriminator, the proposed method achieves the best in LLVM, BDB-C and BDB-J systems with different sample sizes. In the Apache subject system, the proposed method also has the best performance with the increase of sample size. The methods ignoring discriminator network only with different regularization totally rely on a loss function between Ground truth and prediction values. The distributions of the predictions may be far from that of the ground truth due to the lack of modeling discriminator network. For our proposed method, since we proposed a discriminator that can judge the predictions and ground truth, the process forces the generator network generating prediction values closer to ground truth.

Third, the deep neural network with L2 regularization has better performance than deep network with L1 regularization and dropout regularization. Specifically, in the four subject systems, L2 regularization has the lowest MSE, lowest MRE and highest Spearman correlation for almost all sample sizes. This demonstrates that L2 regularization is more effective than L1 regularization and dropout in software performance prediction.

4.7 RQ4: The Effectiveness of PERF-AL

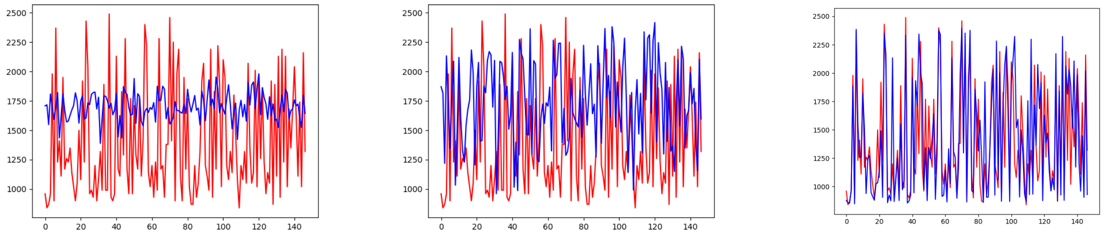
We can compare models with adversarial learning and general neural network on binary configuration options systems in Table 5. Table 6 gives the experimental results in adversarial learning model and general neural network with L2 regularization on both binary and numeric configuration options systems. Since L1 regularization and dropout techniques are not as effective as the L2 regularization discussed in Section 4.6, we only list the L2 regularization in Table 6. As we can see, the model with adversarial learning has the lowest MRE, MSE and highest R compared with model with general neural network. This demonstrates PERF-AL model has better predictions than only using neural network with L2 regularization on these three subject systems, which significantly verifies the superiority of our PERF-AL model. Actually, in our architecture, we utilize L2-NN model as our generator network due to its better performance than L1-NN and dropout-NN and we set parameter θ as the weight of Generator model and discriminator model shown in Eq. 7. Hence, our model is impossibly worse than L2-NN.

For our proposed method, since we introduce a discriminator to distinguish the prediction values from ground truth, the generator network tries to fool the discriminator, which forces the distributions of the prediction values and ground truth to be closer. Therefore, our PERF-AL method achieves better performance.

To further evaluate the effect of the PERF-AL introduced in our model, we visualize the distributions of the ground truth and prediction values in Figure 4. The distributions of ground truth and prediction values are considered. We draw the red line as ground truth values and the blue line as prediction values generated by the network. The first one is the starting stage of training. The second one is general deep network without adversarial learning. The last one is our proposed PERF-AL model. From Figure 4(a), we can observe that the training does not converge at the early stage, the prediction values are far away from ground truth. Then, we can observe that the overlapping between prediction values and ground truth in Figure 4(c) are smaller than Figure 4(b). Furthermore, compared with Figure 4(b), prediction values in Figure 4(c) display a more related fluctuation with ground truth. The visualization demonstrates that our proposed model is more effective.

Table 5: Comparison among PERF-AL, L1, L2 and Dropout regularization.

Subject System	Sample Size	PERF-AL			L1-NN			L2-NN			Dropout-NN		
		MRE	MSE	R	MRE	MSE	R	MRE	MSE	R	MRE	MSE	R
Apache	n	0.2416	0.2667	0.0730	**0.1516**	0.1454	0.0200	0.2101	0.2102	**0.1989**	0.1636	**0.1406**	0.0579
	3n	**0.0744**	0.0598	0.6483	0.1421	0.1391	0.0235	0.0839	**0.0513**	**0.6760**	0.1135	0.0914	0.2590
	5n	**0.0583**	**0.0454**	**0.7237**	0.1098	0.1349	0.0393	0.0668	0.0457	0.6815	0.0763	0.0785	0.5505
LLVM	n	**0.0443**	**0.0475**	**0.4545**	0.0559	0.0745	0.1102	0.0464	0.0675	0.4332	0.0489	0.2071	0.2920
	3n	**0.0237**	**0.0347**	**0.6782**	0.0331	0.0726	0.1177	0.0275	0.0610	0.4640	0.0388	0.1353	0.4033
	5n	**0.0164**	**0.0292**	**0.7062**	0.0259	0.0706	0.1668	0.0198	0.0419	0.5821	0.0223	0.0535	0.4958
BDB-C	n	**1.3323**	0.4083	0.3707	2.2341	**0.3512**	0.2272	1.6516	0.4595	**0.3708**	2.0808	0.3735	0.3010
	3n	**0.1409**	0.2790	0.4984	1.0367	0.3212	0.3293	0.2367	0.2170	**0.5863**	0.4520	**0.1789**	0.4865
	5n	**0.0570**	**0.0305**	**0.8117**	0.2095	0.2834	0.4318	0.0765	0.1476	0.7883	0.1073	0.0753	0.6941
BDB-J	n	**0.0710**	**0.0305**	**0.3526**	0.1295	0.4216	0.3427	0.0801	0.1603	0.3478	0.1024	0.2486	0.3098
	3n	**0.0164**	**0.0881**	**0.5819**	0.1050	0.3311	0.4329	0.0227	0.1107	0.4911	0.0409	0.1278	0.5374
	5n	**0.0159**	**0.0132**	**0.8211**	0.0740	0.3234	0.4620	0.0164	0.0246	0.8055	0.0240	0.0312	0.7686



(a) Performance in early iterations (b) Performance with ordinary deep network (c) Performance with Perf-AL model

Figure 4: An example of software prediction conducted by PERF-AL and ordinary deep network on Apache system. The red line denotes the ground truth and the blue line denotes the predicted values

Table 6: Comparisons between PERF-AL and L2-NN on both binary and numeric configuration options systems.

Subject System	Sample Size	PERF-AL			L2-NN		
		MRE	MSE	R	MRE	MSE	R
DUNE MGS	49	**0.1451**	**0.0090**	**0.5006**	0.1573	0.0106	0.4778
	78	**0.1297**	**0.0071**	**0.6410**	0.1540	0.0094	0.6025
	240	**0.0763**	**0.0040**	**0.7704**	0.0957	0.0084	0.7024
	375	**0.0649**	**0.0028**	**0.8002**	0.0802	0.0114	0.7877
HIPA	261	**0.0922**	**0.0641**	**0.5766**	0.1028	0.0877	0.5129
	528	**0.0621**	**0.0449**	**0.6215**	0.0872	0.0610	0.5822
	736	**0.0488**	**0.0394**	**0.6885**	0.0678	0.0467	0.6215
	1281	**0.0371**	**0.0128**	**0.7352**	0.0466	0.0380	0.7268
HSMGP	77	**0.0470**	**0.0206**	**0.6637**	0.0662	0.0305	0.6124
	173	**0.0262**	**0.0115**	**0.7403**	0.0288	0.0130	0.7363
	384	**0.0215**	**0.0028**	**0.8862**	0.0249	0.0055	0.8442
	480	**0.0168**	**0.0006**	**0.9300**	0.0180	0.0008	0.9262

4.8 Limitations and Threats to Validity

One limitation of our approach is that it is more time consuming than DECART and SPLConqueror methods. In our machine configuration, for binary subject systems, the time of model training increases from 40 seconds to 300 seconds when the sample size increases from n to $5n$ on Apache, LLVM, BDB-c and BDB-J datasets. For binary-numeric subject systems, the time of model training increases from 2 minutes to 8 minutes from the small sample size to the large sample size on DUNE MGS, HIPA and HSMGP datasets. Meanwhile, the time consumption of DECART and SPLConqueror method is from a few seconds to 2 minutes in these subject systems. Although our method takes longer to build the training model than DECART and SPLConqueror, the time cost of our model is still acceptable since our model can achieve higher accuracy.

One threat to validity is that there are many model parameters in our network, including the depth and width of this network, neurons and weight of trainable parameters, learning rate, weight decay etc. Training a huge number of parameters may cost a lot of computation time and memory. Besides, it is difficult to find the optimal solution with all the best model parameters. In our future work, we plan to propose an advanced parameter search strategy for improving convergence rate.

5 RELATED WORK

5.1 Software Performance Prediction

There has been much work focusing on selecting a rational size of sample configurations for performance prediction. For example, Sayyad et al. [25] employed a combination of static and evolutionary learning of model structure. They also utilized a pre-computed solution used as a "seed" in the midst of a randomly-generated initial population to help the Indicator-Based Evolutionary Algorithm

(IBEA) in finding sound and optimum configurations of very large variability models in the presence of competing objectives. Sarkar et al. [24] introduced a new heuristic based on feature frequencies and adapt two widely-used sampling strategies for performance prediction. They also evaluated them in terms of sampling cost with the consideration of prediction accuracy and measurement effort simultaneously. Nair et al. [22] proposed a fast-spectral learner, called WHAT, along with three new sampling techniques. The key idea of WHAT is to explore the configuration space with eigenvalues of the features used in a configuration to determine exactly those configurations for measurement that reveal key performance characteristics. Compared with their methods, ours focuses on developing a new learning method to construct software performance models with higher prediction accuracy and less sample data. In our future work, we will explore if the above related work could be integrated into our framework.

Apart from performance prediction, there is also much work on predicting other quality attributes of configurable software. For example, Zhang et al. [36] proposed a Bayesian Belief Network (BBN) based approach to quality prediction and assessment for a software product line. For developing a specific system, a member of the product line, they reused the expertise captured by a BBN. This helps to capture the impact of configuration options on quality attributes and assess the quality of a product line member by performing quantitative analysis over it. Their methods are good for quality attributes that are hard to measure and quantify. Guo et al. [11] presented a GA-based optimized feature selection approach, GAFES, for automated product derivation in a software product line. A key component of GAFES is an algorithm that can transform an arbitrary feature selection into a valid feature combination, which can minimize or maximize an objective function, such as total cost, subject to resource constraints. Our work focuses on performance prediction and can be extended to other quality attributes in the future.

5.2 Generative Adversarial Network and its Applications

The applications of generative adversarial network have been widely employed in many areas such as computer vision and industrial design [26][35][33]. There are also some applications of GAN in software engineering research. For example, Harer et al. [13] applied an adversarial learning approach to repair software vulnerabilities. Zhang et al [37] proposed DeepRoad, which applies Generative Adversarial Networks along with the corresponding real-world weather scenes to produce driving scenarios with various weather conditions (including those with extreme conditions). As an unsupervised DNN-based framework, their DeepRoad also utilizes metamorphic testing techniques to check the consistency of systems using synthetic images. In our work, we apply the adversarial learning idea in GAN to performance prediction for configurable software. We train a network to generate optimal control inputs to nonlinear dynamic systems. Where the discriminator network is known as a critic that checks the optimality of the software prediction and the generative network is known as an adaptive network that generates the optimal prediction accuracy. The critic and adaptive network train each other to approximate a nonlinear optimal control, which can achieve the best performance prediction in software. As far as we know, this paper is the first to introduce a GAN model into software performance prediction. Our approach achieves higher accuracy with smaller samples and supports both binary and numeric configuration options.

6 CONCLUSION

In this paper, we have proposed PERF-AL, an adversarial learning framework for software performance prediction. A deep neural network is adopted to learn the influence of configuration options on software performance during training phase. Adversarial learning is further introduced to capture the correlation between the prediction and ground truth, forcing the distributions of the prediction and ground truth to be close to each other. PERF-AL is able to work with both binary and numeric configuration options. Experimental results on the seven datasets demonstrate that our PERF-AL model outperforms the state-of-the-art approaches with smaller samples.

For replication purpose, our experimental data and source code are publicly available at: ***https://github.com/GANPerf/GANPerf***.

In the future, we will design a search mechanism for choosing the best hyper-parameters of the neural network model to further optimize the performance of our approach. We will also explore the synergy between our approach and the related work on sample selection.

REFERENCES

[1] 2019. SPLConqueror project page. (2019). Accessed 2019-02-01. http://www.fosd.de/SPLConqueror.

[2] Sven Apel and Christian Kästner. 2009. An overview of feature-oriented software development. *Journal of Object Technology* 8, 5 (2009), 49–84.

[3] Don Batory, David Felipe Benavides Cuevas, and Antonio Ruiz Cortés. 2006. Automated analysis of feature models: challenges ahead. *Communications of the ACM-Software product line, 49 (12), 45-47.* (2006).

[4] Krzysztof Czarnecki, Kasper Østerbye, and Markus Völter. 2002. Generative programming. In *European Conference on Object-Oriented Programming*. Springer, 15–29.

[5] Ronen Eldan and Ohad Shamir. 2016. The power of depth for feedforward neural networks. In *Conference on learning theory*. 907–940.

[6] Tron Foss, Erik Stensrud, Barbara Kitchenham, and Ingunn Myrtveit. 2003. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering* 29, 11 (2003), 985–995.

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 315–323.

[8] Ian Goodfellow. 2016. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

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

[11] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software* 84, 12 (2011), 2208–2221.

[12] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2018. Data-efficient performance learning for configurable systems. *Empirical Software Engineering* 23, 3 (2018), 1826–1867.

[13] Jacob Harer, Onur Ozdemir, Tomo Lazovich, Christopher Reale, Rebecca Russell, Louis Kim, et al. 2018. Learning to repair software vulnerabilities with generative adversarial networks. In *Advances in Neural Information Processing Systems*. 7933–7943.

[14] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[15] Hongyu Zhang Huong Ha. 2019. DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network. (2019). Accessed 2019-31-05. https://2019.icse-conferences.org/event/icse-2019-technical-papers-deepperf-performance-prediction-for-configurable-software-with-deep-sparse-neural-network.

[16] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 497–508.

[17] Magne Jorgensen and Martin Shepperd. 2007. A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering* 33, 1 (2007), 33–53.

[18] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[19] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*. 6231–6239.

[21] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[22] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Faster discovery of faster system configurations with spectral learning. *Automated Software Engineering* (2018), 1–31.

[23] Andrew Y Ng. 2004. Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 78.

[24] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-efficient sampling for performance prediction of configurable systems (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 342–352.

[25] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 465–474.

[26] Kevin Schawinski, Ce Zhang, Hantian Zhang, Lucas Fowler, and Gokula Krishnan Santhanam. 2017. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *Monthly Notices of the Royal Astronomical Society: Letters* 467, 1 (2017), L110–L114.

[27] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 284–294.

[28] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 167–177.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[30] Eno Thereska, Bjoern Doebel, Alice X Zheng, and Peter Nobel. 2010. Practical performance models for complex, popular applications. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38. ACM, 1–12.

[31] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[32] László Tóth. 2013. Phone recognition with deep sparse rectifier neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 6985–6989.

[33] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. 2016. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*. 613–621.

[34] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. 2012. Automated inference of goal-oriented performance prediction functions. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 190–199.

[35] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*. 82–90.

[36] Hongyu Zhang, Stan Jarzabek, and Bo Yang. 2003. Quality prediction and assessment for product lines. In *International Conference on Advanced Information Systems Engineering*. Springer, 681–695.

[37] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 132–142.

[38] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. 2015. Performance prediction of configurable software systems by fourier learning (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 365–373.