Predicting Configuration Performance in Multiple Environments with Sequential Meta-Learning

JINGZHI GONG*, University of Electronic Science and Technology of China, China and Loughborough University, United Kingdom
TAO CHEN†, University of Birmingham, United Kingdom

Learning and predicting the performance of given software configurations are of high importance to many software engineering activities. While configurable software systems will almost certainly face diverse running environments (e.g., version, hardware, and workload), current work often either builds performance models under a single environment or fails to properly handle data from diverse settings, hence restricting their accuracy for new environments. In this paper, we target configuration performance learning under multiple environments. We do so by designing `SeMPL`—a meta-learning framework that learns the common understanding from configurations measured in distinct (meta) environments and generalizes them to the unforeseen, target environment. What makes it unique is that unlike common meta-learning frameworks (e.g., `MAML` and `MetaSGD`) that train the meta environments in parallel, we train them sequentially, one at a time. The order of training naturally allows discriminating the contributions among meta environments in the meta-model built, which fits better with the characteristic of configuration data that is known to dramatically differ between different environments. Through comparing with 15 state-of-the-art models under nine systems, our extensive experimental results demonstrate that `SeMPL` performs considerably better on 89% of the systems with up to 99% accuracy improvement, while being data-efficient, leading to a maximum of 3.86× speedup. All code and data can be found at our repository: https://github.com/ideas-labo/SeMPL.

CCS Concepts: • **Software and its engineering** → **Software performance**.

Additional Key Words and Phrases: Configurable System, Machine Learning, Meta Learning, Performance Prediction, Performance Learning, Configuration Learning

ACM Reference Format:
Jingzhi Gong and Tao Chen. 2024. Predicting Configuration Performance in Multiple Environments with Sequential Meta-Learning. *Proc. ACM Softw. Eng.* 1, FSE, Article 17 (July 2024), 24 pages. https://doi.org/10.1145/3643743

1 INTRODUCTION

Most software systems can be flexibly configured to meet the needs of a certain scenario. This is achieved by jointly adjusting various configuration options, which can determine, e.g., the size of the threads pool; the capacity of the cache; or the underlying algorithm to use [Chen and Li 2023a,b; Sayagh et al. 2020]. As such, the configuration will inevitably lead to considerable impacts on the performance, e.g., runtime and throughput. Even carefully managed configuration could

*This research was conducted when Jingzhi Going visited the University of Electronic Science and Technology of China.
†Tao Chen is the corresponding author (t.chen@bham.ac.uk).

Authors' addresses: Jingzhi Gong, University of Electronic Science and Technology of China, Chengdu, China and Loughborough University, Loughborough, United Kingdom, j.gong@lboro.ac.uk; Tao Chen, University of Birmingham, Birmingham, United Kingdom, t.chen@bham.ac.uk.

ACM 2994-970X/2024/7-ART17
https://doi.org/10.1145/3643743

still be error-prone and result in devastating consequences—a previous study finds that 59% of the configuration-related issues have caused severe performance concerns in modern configurable software systems [Han and Yu 2016].

The key to configuration management is how to accurately infer the performance of the given configurations via a performance model, serving as the foundation for, e.g., performance debugging [Iqbal et al. 2022], configuration tuning [Nair et al. 2020], and even self-adaptation [Chen 2022; Chen and Bahsoon 2017b; Chen et al. 2018]. This, however, can neither be realistically tackled by analytical models (due to diverse types of configuration options [Chen and Li 2021]) nor profiling (due to expensive measurement [Nair et al. 2020]). Through learning on the available data, current work has successfully leveraged machine learning to build various performance models [Gong and Chen 2022]. Yet, those approaches mostly focus on configuration performance learning under one environment [Chen et al. 2021; Gong and Chen 2023; Guo et al. 2018; Ha and Zhang 2019; Queiroz et al. 2016; Siegmund et al. 2012a,b; Valov et al. 2015], e.g., a pre-defined workload, a fixed hardware, and a specific version.

Working on a single environment is an over-optimistic assumption as it is not uncommon to see configurable software systems run under diverse conditions. For example, a database system may experience both read-heavy and write-heavy workload [Jamshidi et al. 2017a]. Similarly, the hardware between the testing and production infrastructure might be drastically different [Brunnert et al. 2015; Leitner and Cito 2016], especially during the modern DevOps era. The ignorance of multiple environments would inevitably harm the effectiveness of a performance model. Jamshidi et al. [2017a] reveal that the accuracy of a single environment model can be severely degraded when used in a different environment. Furthermore, due to the expensive measurement of configuration, e.g., it can take hours or days to measure only a few configurations [Chen and Li 2021; Valov et al. 2017], building a new model for every distinct environment is unrealistic. Recently, at ICSE'23, Mühlbauer et al. [2023] have demonstrated that predicting under multiple environments can pose significant threats to the robustness and generalizability of performance models learned using a single environment. Through a large-scale study, they concluded that:

> *"Performance models based on a single workload are useless, unless the configuration options' sensitivity to workloads is accounted for."*

As a result, the failure to take multiple environments into account when learning the performance model for configurable software not only degrades accuracy but also incurs extra overhead of model re-building, as the valuable data samples measured under different environments are wasted [Iorio et al. 2019; Krishna et al. 2021; Nair et al. 2018]. This leads to a previously unaddressed problem: *How to effectively leverage configurations measured in different environments for modeling configuration performance?* Yet, learning a performance model under multiple environments for configurable software is challenging due to the large variations between data measured in distinct environments. For example, several studies have revealed that varying environments can cause substantial changes in the performance distributions with non-monotonic correlations, including workloads [Chen 2022; Mühlbauer et al. 2023; Pereira et al. 2020], versions [Martin et al. 2022] and hardware [Jamshidi et al. 2017a].

Despite being uncommon, Mühlbauer et al. [2023] summarized two major categories from existing work on how multiple environments have been handled, each with their own limitations:

(1) **Environment as additional features [Chen 2019; Chen and Bahsoon 2017a; Koc et al. 2021; Lengauer et al. 2014]:** Here, the specific properties of an environment, e.g., size and job counts, are considered as model features alongside the configuration options in performance model learning. However, in this category, not only the additional measurements can be

costly, but the extra dimension(s) also make the true concept more difficult to learn and generalize.

(2) **Transfer learning [Jamshidi et al. 2017b; Krishna et al. 2021; Martin et al. 2022; Valov et al. 2017]:** Given an existing performance model trained in the source environment(s), its differences to the new environment can be learned via transfer learning models. Yet, the key shortcoming thereof is that the loss function in transfer learning is tailored to a specific target environment, hence lacking generalizability to arbitrary environments.

In this paper, we propose the *third category* to address the aforementioned gap and challenges using the concept of meta-learning—a form of machine learning that is capable of "learning to learn" by learning data across multiple environments (or tasks[1]), and generalize the learning to an unforeseen one. From this, we present **Se**quential **M**eta **P**erformance **L**earning (SeMPL), a framework that produces a general meta-model learned from data of all existing meta environments during pre-training. When needed, such a meta-model can then be quickly adapted for accurately predicting performance under a target environment via fine-tuning. What makes SeMPL unique is that, unlike popular general-purpose meta-learning frameworks such as MAML [Finn et al. 2017] and MetaSGD [Li et al. 2017] which learn the data of meta environments in parallel, SeMPL follows sequential meta-learning that learns the datasets of meta environments one at a time in a specific order, aiming to discriminate their contributions in the meta-model built. This is a tailored design to deal with the potentially large variations between measurements in different environments from the configuration data.

Specifically, our contributions are:

- We justify, both analytically and empirically, why the concept of sequential meta-learning in SeMPL is more suitable for learning multi-environment configuration data than general frameworks such as MAML and MetaSGD, according to three unique properties in SeMPL:
  (1) the sequence matters;
  (2) train later contributes more;
  (3) and using more meta environments are beneficial.
- Drawing on those properties provided by SeMPL, we design a novel method that selects the optimal sequence of learning the data of meta environments, which fits the characteristics of the configuration data under multiple environments.
- Based on nine systems with 3-10 meta environments and five training data sizes each, we experimentally compare SeMPL against 15 state-of-the-art models for single or multiple environments, taken from the software, system, and machine learning communities.

The results are encouraging: SeMPL significantly outperforms existing models in 89% of the systems with up to 99% accuracy improvement; it is also data-efficient with at most 3.86× speedup.

The paper is organized as follows: Section 2 overviews the preliminaries and related work. Section 3 elaborates the theory behind this work and Section 4 illustrates the algorithm and details of the SeMPL framework. Section 5 presents the experimental setup followed by the experiment results in Section 6. Discussion and conclusion are presented in Section 7 and 8, respectively.

2 PRELIMINARIES AND RELATED WORK

2.1 Single Environment Configuration Performance Learning

Learning performance for configurable software systems is commonly formulated as single-task learning that learns and generalizes data under a single environment. Formally, we aim to build a

[1]In this paper, we use task and environment interchangeably.

regression model f that predicts the performance p of an unforeseen configuration $\overline{\mathbf{x}}'$:

$$\begin{aligned} &\text{Train: } \boldsymbol{\mathcal{E}}_{target} \Longrightarrow f \\ &\text{Predict: } f(\overline{\mathbf{x}}') = p_{target} \mid \overline{\mathbf{x}}' \in \boldsymbol{\mathcal{E}}_{target} \end{aligned} \quad (1)$$

whereby $\boldsymbol{\mathcal{E}}_{target}$ denotes the training samples of configuration-performance pairs ($\{\mathcal{C} \rightarrow \mathcal{P}\}$) measured under the target environment, such that $\overline{\mathbf{x}} \in \boldsymbol{\mathcal{E}}_{target} = \{\mathcal{C} \rightarrow \mathcal{P}\}$. $\overline{\mathbf{x}}$ is a measured configuration and $\overline{\mathbf{x}} = (x_1, x_2, \cdots, x_n)$, where there are n configuration options and each option x_i can be either binary or categorical/numerical. f should be trained in such a way that its prediction (p_{target}) on $\overline{\mathbf{x}}'$ is as close to the actual performance as possible. Many models for learning configuration performance in a single environment have been proposed [Chen et al. 2021; Gong and Chen 2023; Guo et al. 2018; Ha and Zhang 2019; Queiroz et al. 2016; Siegmund et al. 2012a,b; Valov et al. 2015]. Here, we briefly explain some state-of-the-art models, which are also compared in Section 6:

- **DeepPerf** [Ha and Zhang 2019]: a Deep Neural Network (DNN) model with L_1 regularization and fast hyperparameter tuning for sparse performance learning.
- **DECART** [Guo et al. 2018]: an improved regression tree model [Breiman 2017] with a specialized sampling mechanism.
- **Random Forest (RF)** [Chen et al. 2021; Queiroz et al. 2016; Valov et al. 2015]: an ensemble of trees that tackle the feature sparsity issue.
- **SPLConqueror** [Siegmund et al. 2012b]: a baseline model that relies on linear regression.
- **XGBoost** [Chen and Guestrin 2016]: a gradient boosting algorithm that leverages the combination of multiple trees to create a robust ensemble.

2.2 Configuration Performance Learning with Multiple Environment Inputs

To handle multiple environments, a natural way is to combine the environmental features with configuration options in the model. In this way, Equation (1) remains unchanged but the configuration-performance pairs from all available environments are merged and the environment features serve as additional inputs, denoted as $\mathcal{E}$, i.e., $\boldsymbol{\mathcal{E}}_{target} = \{\mathcal{C} \times \mathcal{E} \rightarrow \mathcal{P}\}$. For example, Chen [2019] has followed this strategy to examine various types of underlying models. The environmental features therein are the workload and request rate, *etc.* More domain-specific environmental features also exist, such as those for program verification [Koc et al. 2021], high-performance computing [Lengauer et al. 2014], and cloud computing [Chen and Bahsoon 2017a].

However, a major limitation thereof is that the combined input space of options and environmental features makes the training even more difficult, as not all options are environment-sensitive [Lesoil et al. 2023].

2.3 Joint Learning for Configuration Performance

2.3.1 Transfer Configuration Performance Learning. Building performance models under multiple environments can be also handled by transfer learning, in which Equation (1) is changed to:

$$\begin{aligned} &\text{Train: } \boldsymbol{\mathcal{E}}_1 \cup \boldsymbol{\mathcal{E}}_2 ... \cup \boldsymbol{\mathcal{E}}_m \cup \boldsymbol{\mathcal{E}}_{target} \Longrightarrow f \\ &\text{Predict: } f(\overline{\mathbf{x}}') = p_{target} \mid \overline{\mathbf{x}}' \in \boldsymbol{\mathcal{E}}_{target} \end{aligned} \quad (2)$$

Here, by treating the learning of data under different environments as independent tasks, both the data from source environment(s) ($\boldsymbol{\mathcal{E}}_1, ..., \boldsymbol{\mathcal{E}}_m$) and target environment ($\boldsymbol{\mathcal{E}}_{target}$) are jointly learned by a base learner with information sharing on, e.g., model parameters/data samples [Jamshidi et al. 2017b; Krishna et al. 2021], features [Martin et al. 2022], or prediction outputs [Valov et al. 2017]. Among others, **BEETLE** [Krishna et al. 2021] is a transfer learning model for configuration performance learning. The key idea is to evaluate all the source-target pairs over the candidate environments and identify the best source as the "bellwether" to use. The data of the source and target environment are jointly learned by a regression tree model. **tEAMS** [Martin et al. 2022] is a recent model caters for the evolution of configurable software. With feature alignment, the model trained on the data of an older version is transferred to learn the data under the new version.

Yet, the key limitation of transfer learning is that, regardless of how many environments serve as the source data, the loss function therein needs to be specific to the target environment, making it difficult to generalize to the other environments [Torrey and Shavlik 2010].

2.3.2 Multi-Task Configuration Performance Learning. To cope with the above limitation, a relevant paradigm is multi-task learning, which can also learn data from multiple environments as the sources ($\mathcal{E}_1, ..., \mathcal{E}_m$) [Alabed and Yoneki 2021; Madhumathi and Suresh 2018; Yang and Hospedales 2017] and has the ability to generalize the predictions for all environments learned simultaneously. Here, Equation (2) will be changed to:

$$\begin{aligned} &\text{Train: } \mathcal{E}_1 \cup \mathcal{E}_2 ... \cup \mathcal{E}_m \cup \mathcal{E}_{target} \Longrightarrow f \\ &\text{Predict: } f(\overline{\mathbf{x}}') = \{p_1, p_2, ..., p_m, p_{target}\} \mid \overline{\mathbf{x}}' \in \mathcal{E}_{target} \end{aligned} \tag{3}$$

whereby p_m is the predicted performance of $\overline{\mathbf{x}}'$ under the mth environment. There is no clear distinction between source and target environments, as the key in multi-task learning is to share information on data from whatever available environments. For example, Madhumathi and Suresh [2018] propose **Multi-Output Random Forest (MORF)**, a multi-task learning version of Random Forest where there is one dedicated output for each environment of performance prediction.

However, the training requires foreseeing the target environment and suffers "negative transfer", as task gradients may interfere, overcomplicating the loss function landscape [Standley et al. 2020].

2.4 Meta-Learning for Configuration Performance

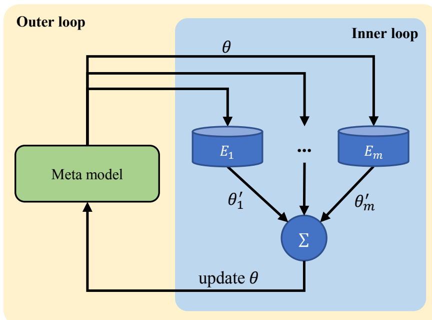
Unlike transfer and multi-task learning, meta-learning seeks to learn a meta-model based on some readily available data from known environments (a.k.a meta environments) without foreseeing the target environment [Vilalta and Drissi 2002]. Formally, in meta-learning, the process becomes:

$$\begin{aligned} &\text{Train: } \mathcal{E}_1 \cup \mathcal{E}_2 ... \cup \mathcal{E}_m \Longrightarrow f' ; \mathcal{E}_{target} + f' \Longrightarrow f \\ &\text{Predict: } f(\overline{\mathbf{x}}') = p_{target} \mid \overline{\mathbf{x}}' \in \mathcal{E}_{target} \end{aligned} \tag{4}$$

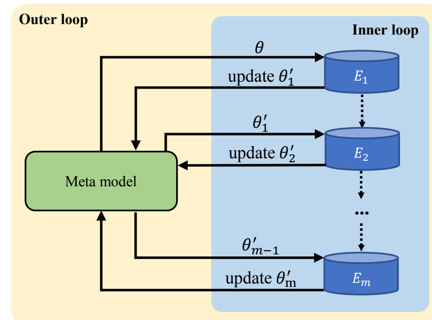
The key idea is that by learning the data of meta environments using a base learner at the meta-level, the meta-model f' will obtain the ability of "learning to learn", enabling easier fine-tuning on the new target environment ($\mathcal{E}_{target}$) when it becomes available. As such, meta-learning achieves: (1) fast adaptation; and (2) a better generalized and more accurate model [Jamshidi et al. 2017b].

Although rarely used for configuration performance learning, there exist different models for meta-learning in the machine learning community [Chua et al. 2021; Finn et al. 2017; Li et al. 2017; Yao et al. 2019], in which the meta tasks are learned simultaneously and the outcomes are combined. The most noticeable ones are MAML [Finn et al. 2017] and its variant MetaSGD [Li et al. 2017]: **MAML** [Finn et al. 2017] is a state-of-the-art meta-learning framework that has been widely applied in different domains [Chen and Chen 2022], including software engineering [Chai et al. 2022]. MAML, if used for configuration performance learning, learns a set of good model parameter values on data of the meta environments in parallel, building a meta-model (f'), which can then be adapted to build a fine-tuned model f for the target environment. In contrast, **MetaSGD** [Li et al. 2017] extends the MAML by additionally adapting the learning rate along the meta-training, expediting the learning over MAML.

Nevertheless, in Section 3, we will explain why general models like MAML are relatively ill-suited to configuration performance learning and elaborate on the theory behind the proposed SeMPL framework.



(a) MAML



(b) SeMPL

Fig. 1. Workflow of MAML and the proposed SeMPL. The meta-model can be produced by any base learner.

3 THE THEORY BEHIND SEMPL FOR CONFIGURATION PERFORMANCE LEARNING

In essence, the key to the success of MAML (and its variants) is that it can produce a set of good model parameter values for the target environment to start training, paired with any gradient descent-based learner, which enables fast adaptation of the created meta-model [Raghu et al. 2020]. As in Figure 1a, this is achieved via iteratively updating the parameters of the meta-model (θ for the outer loop[2]) using those trained on the data of each meta environment individually (i.e., $\theta'_1, ..., \theta'_m$ from the inner loop). In such a process, data of all meta environments are learned simultaneously followed by a linear aggregation of their model parameter values, hence they provide equal contributions to the meta-model. This, however, is ill-suited for the performance learning of configurable software systems. Because, unlike some other domains, prior studies have revealed that the data from distinct environments exhibit drastically different correlations between performance and configuration:

- A study on x264 [Pereira et al. 2020] showed that different video inputs can lead to varying and non-monotonic performance results.
- Chen [2022] discovered dramatic configuration landscape shifts due to workload changes.
- Mühlbauer et al. [2023] reported that varying workloads can change how configuration options affect performance, causing drastic variations among performance distributions.
- Jamshidi et al. [2017a] revealed the non-linear correlation on data from different workloads, software versions and hardware.

As a result, these prior observations derive the following insight:

> ***Insight of Learning Configuration Data:*** The ideal parameter values and distributions for models learned under different environments can be, unfortunately, rather different.

This means that treating all meta environments equally in MAML can force some less useful (or even misleading) meta environments to contribute, causing some model parameter values to largely deviate from the optima required for the target environment. Therefore, what we need is a tailored model for multi-environment configuration performance learning with the following requirements:

- **Requirement 1:** Configuration data of distinct meta environments should be discriminated.
- **Requirement 2:** Configuration data of distinct meta environments should have alterable contributions to the learning.

[2]The default iteration limit of the outer loop in MAML is 1.

- **Requirement 3:** The valuable information of data from different meta environments should be exploited fully since measuring configurations is highly expensive [Chen and Li 2021; Nair et al. 2020].

A naive (perhaps natural) solution to the above is to consider weights between meta environments when updating the θ in MAML. However, this has the following shortcomings:

- It is difficult to precisely quantify the relative contributions between meta environments, as the model parameter values of neurons are naturally multi-dimensional.
- Setting/updating the weights is challenging, especially with more meta environments—a typical cognitive issue when weighting on multiple criteria/entities [Li et al. 2022].
- It has been shown that the weights can obscure the optimization during the training [Chen and Li 2023b].

Instead of weights, we propose sequential meta-learning in SeMPL: as in Figure 1b, the datasets in distinct meta environments are learned one by one in a certain order, and so does the update of meta-model's parameters θ. The sequence of meta environments prioritizes their contributions to learning, thereby resolving the limitation of MAML for configuration performance learning without extra parameters. In what follows, we elaborate on the theory behind SeMPL and its properties.

3.1 The Sequence Matters

The very first property that we intentionally design for SeMPL, which meets **Requirement 1**, is:

PROPERTY 1. *The sequence of learning the data in distinct meta environments significantly influences the meta-model built, hence leading to different adaptation effects for the target environment.*

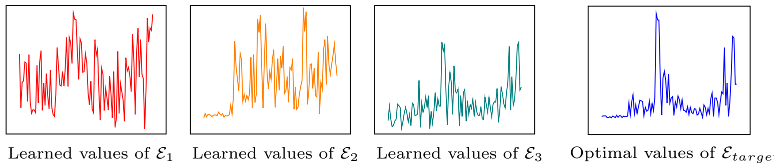
Given that the datasets in meta environments are learned in sequence with only one being considered each time, the model parameter values for a meta environment trained earlier would serve as the initial values of the ones that will be learned later. As such, different sequences would cause distinct orders of intermediate initialization on the model parameter values for learning some meta environments. Since the heuristic of gradient descent in training is often stochastic and the training sample size can be limited for configurable software systems, working on different sets of starting points would most likely have different results, leading to diverse meta-models in the end.

For example, Figure 2a illustrates the distributions of model parameter values when trained individually on data of three single environments, namely $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$. With SeMPL in this case (Figure 2b), the model parameter values trained from $\mathcal{E}_1$ becomes the initial values for learning the data of $\mathcal{E}_2$, after which the values become the initials for learning the $\mathcal{E}_3$ data. Here, the resulted model parameter values when learning the data of $\mathcal{E}_2$, whose initialization is set by learning the data of $\mathcal{E}_1$, would lead to a distribution mixed from those of learning the data of each meta environment individually (Figure 2a). As such, if we swap $\mathcal{E}_1$ with $\mathcal{E}_3$, the resulting distribution of model parameter values would be different. MAML, in contrast, is insensitive to the sequence of meta environments as all of them are learned simultaneously and the learned model parameter values are aggregated to update the meta-model, as in Figure 2c. Figure 3a shows the empirical results for a randomly selected system and we observed similar outcomes for the others. Clearly, the different sequences in the meta-learning of SeMPL can lead to distinct accuracy for the target environment.

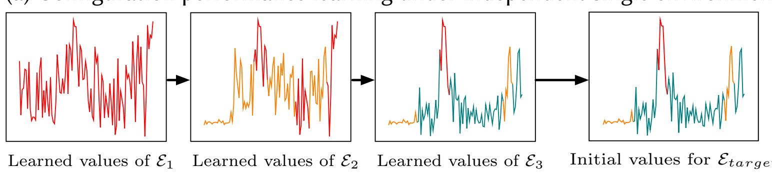
3.2 Train Later Contributes More

To satisfy **Requirement 2**, a related property that can be derived from **Property 1** in SeMPL is:

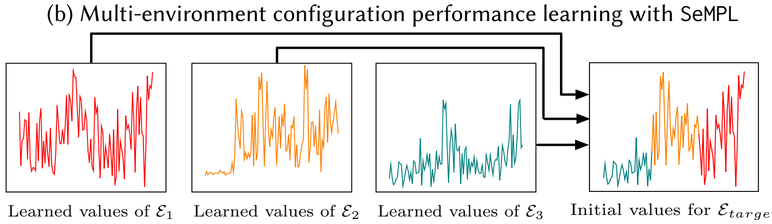
PROPERTY 2. *If the data of meta environment $\mathcal{E}_{i+1}$ is learned later than that of $\mathcal{E}_i$, then the model parameter values of meta-model are more similar to those of learning $\mathcal{E}_{i+1}$ data alone than those trained under $\mathcal{E}_i$. Therefore, data of the more useful meta environment should be learned later.*



(a) Configuration performance learning under independent single environment



(b) Multi-environment configuration performance learning with SeMPL

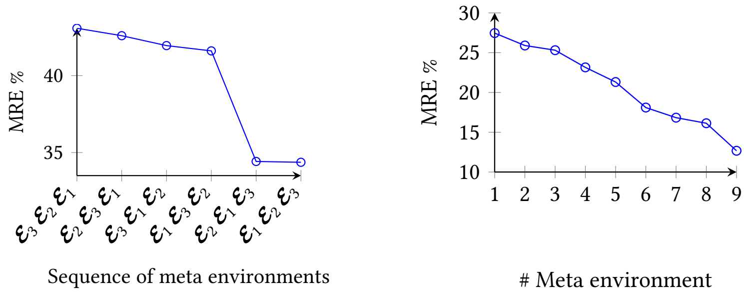


(c) Multi-environment configuration performance learning with MAML

Fig. 2. Illustrating the distributions of the model parameter values in different situations under a real-world software system; the base learner is a regularized Deep Neural Network (it is best viewed in color). The x- and y-axis are model parameters and their corresponding values, respectively.

With the sequential training of meta environments data in SeMPL, it is easy to see that, if there are n meta environments ($n > 0$), the model parameter values learned on $\mathcal{E}_i$ will experience $n - i$ ($i \in [1, n]$) updates later on. Since in each update, the training tends to tune the model parameter to fit the data in the corresponding meta environment, which can be rather different to $\mathcal{E}_i$ due to the characteristics of configuration landscape, the distribution of model parameter values learned from $\mathcal{E}_i$ will be gradually overridden with more subsequent updates. As a result, the bigger the i, i.e., a meta environment sits at a later position of the sequence, the fewer updates to the parameters trained under $\mathcal{E}_i$, allowing it to preserve more model parameter values (and distribution) in the meta-model and contribute more therein than the meta environments learned earlier. Importantly, this leads to an obvious rule: the more useful meta environments—those that can provide better initial model parameter values for the target environment—should be left to later positions (we will explain how we measure the usefulness in Section 4).

Using the example from Figure 2b again, clearly, the model parameter values learned on the first meta environment $\mathcal{E}_1$ will need to be updated via learning the $\mathcal{E}_2$ data and $\mathcal{E}_3$ data, hence it only contributes to the smallest proportion of its distribution in the meta-model. In contrast, the last meta environment $\mathcal{E}_3$ will contribute the most to the meta-model as most of the model parameter values learned under it will be preserved into those of the meta-model (i.e., there will be no further update). $\mathcal{E}_2$ contributes less than $\mathcal{E}_3$ but more than $\mathcal{E}_1$, as its resulted model parameter values are updated once. According to Figure 2a, in this case, $\mathcal{E}_3$ has most parts of its parameter distribution close to that of $\mathcal{E}_{target}$, hence it is beneficial to leave it as the last to be learned, which preserves more of its model parameter values. Similarly, $\mathcal{E}_2$ tends to have more proportions of its parameter distribution closer to that of $\mathcal{E}_{target}$ than that of $\mathcal{E}_1$, hence learning its data later than that of $\mathcal{E}_1$ would ensure it contributes relatively more. All these can then lead to good initialization



(a) DeepArch (b) x264

Fig. 3. Empirical results that verify the properties of real-world software systems. The y-axis is the testing Mean Relative Error (MRE) on $\mathcal{E}_{target}$. (a) confirms Property 1 and 2; $\mathcal{E}_3$ is the most useful environments for $\mathcal{E}_{target}$, following by $\mathcal{E}_2$ and then $\mathcal{E}_1$. (b) reveals Property 3.

of the parameter's value in the meta-model (the rightmost in Figure 2b) with respect to the optimal parameter distribution for $\mathcal{E}_{target}$ (the rightmost in Figure 2a). Empirically, for a real-world system in Figure 3a, placing the more useful meta environments to the latter positions in SeMPL can lead to better accuracy for the target environment. Similar results have been observed in other cases.

MAML (Figure 2c), in contrast, forces all meta environments to contribute equally in an aggregation to update the parameter values in the meta-model, hence for configuration data, this can easily result in an initial distribution that is highly deviated from the optimal setting of the target environment (the rightmost in Figure 2c). We will also experimentally justify this in Section 6.

3.3 More Meta Environments are Beneficial

It is natural to ask, given **Property 1** and **Property 2**, why not use only the most useful meta environment to initialize the model parameters for the target environment? The answer is that those less useful meta environments, albeit contributing to relatively fewer proportions in the meta-model, may still provide excellent starting points for certain parts of the parameter distribution. This leads to SeMPL's final property that fulfills **Requirement 3**:

Property 3. *Learning from more meta environments can help to cover the "corner cases" of initialing the model parameter values for the target environment.*

Considering the example from Figure 2b, we see that, although $\mathcal{E}_1$ only preserves a small part of its learned parameter distribution in the meta-model, this may still be close to the optimal setting for the target environment (e.g., the peak parameter values as in Figure 2b), hence complementary to what is missing in the contributions from $\mathcal{E}_2$ and $\mathcal{E}_3$. The same principle can be similarly applied for $\mathcal{E}_2$. As a random example, Figure 3b illustrates the sensitivity of SeMPL's accuracy over a target environment to the number of meta environments (with the appropriate sequence). Clearly, the more meta environments, the better the accuracy—a pattern that we observed for all systems.

Noteworthily, **Property 3** may not be true for MAML as exploiting all meta environments equally makes it severely suffer from the side-effect of less useful meta environments, which is not uncommon with the configuration data. SeMPL, in contrast, is able to mitigate such a side-effect by prioritizing the sequence of meta environments, thanks to **Property 1** and **Property 2**. Indeed, as we will show in Section 6.2, the sequence-insensitivity has caused MAML to perform significantly worse than SeMPL in learning configuration performance data.

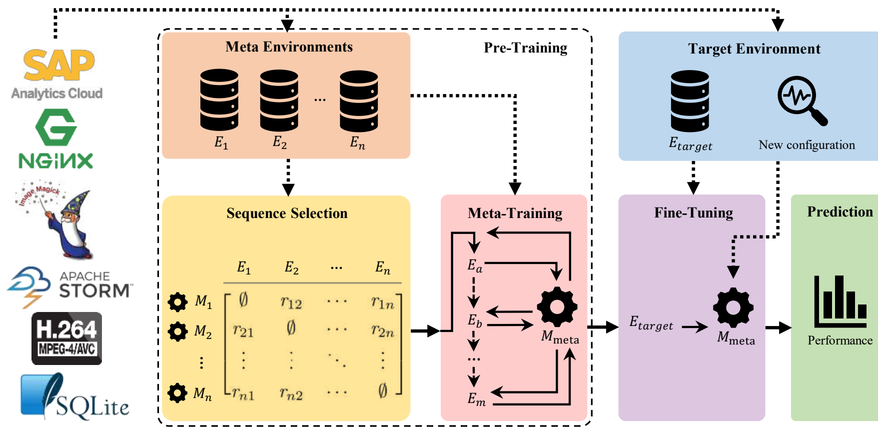


Fig. 4. The SeMPL architecture for learning configuration performance of a system with multiple environments.

4 IMPLEMENTING AND ENGINEERING SEMPL

To engineer the properties from the theory behind SeMPL that fulfill the requirements for configuration performance learning under multiple environments, our implementation has three core components, namely, *Sequence Selection*, *Meta-Training*, and *Fine-Tuning*. The former two resemble a pre-training process for the outer and inner loop (Figure 1b) whereas the last is triggered when the target environment becomes available, as shown in Figure 4. These are specified below:

- **Sequence Selection:** This component finds the optimal training sequence of meta environment data in SeMPLwith respect to **Property 1** and **Property 2**, considering all the available environments (**Property 3**). It deals with three low-level questions:
  (1) How to assess the usefulness of the data of a known meta environment with respect to an unforeseen target environment?
  (2) How to ensure such an assessment is reliable?
  (3) How to guarantee efficiency in the assessment?
- **Meta-Training:** Here, we update the parameters of meta-model by sequentially learning the data of meta environments following the order provided by the *Sequence Selection*.
- **Fine-Tuning:** Given the meta-model from the *Meta-Training*, here the aim is to update the model parameter values using measured configurations under the target environment.

Given the flexible nature, SeMPL is agnostic to the base learner that learns the meta-model, hence it can be paired with any regression learning algorithm from the literature. In the following, we will delineate the above components and the pre-training process of SeMPL in Algorithm 1.

4.1 Sequence Selection

4.1.1 How to assess the usefulness of meta environments to the target environment? Since the target environment is unforeseen by the time of building the meta-model, we assess the overall Mean Relative Error (MRE) on how the single-model $\mathcal{M}_i$, which is trained under an individual meta environment $\mathcal{E}_i$, performs when being validated across all other remaining meta environments.

Formally, MRE is a widely used scale-free metric for performance prediction [Guo et al. 2018; Ha and Zhang 2019; Shu et al. 2020]:

$$MRE = \frac{1}{k} \times \sum_{t=1}^{k} \frac{|A_t - P_t|}{A_t} \times 100\% \tag{5}$$

Algorithm 1: Sequence Selection

Input: Data from meta environments $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n\}$
Output: The optimal sequence $\mathcal{E}_{seq}$

1 **for** $\forall \mathcal{E}_i \in \mathcal{E}$ **do**
2 $\mathcal{M}_i \leftarrow$ TRAIN($\mathcal{E}_i$)
3 **for** $\forall \mathcal{E}_j \in \mathcal{E}$ **do**
4   **if** $\mathcal{E}_j$ *is not* $\mathcal{E}_i$ **then**
5    $\mathbf{A} \leftarrow \overline{\mathbf{a}}_{\mathbf{ij}}$ = TESTMREWITHREPEATS($\mathcal{M}_i, \mathcal{E}_j$)
6   **else**
7    $\mathbf{A} \leftarrow \emptyset$
8 **for** $\forall \mathcal{E}_j \in \mathcal{E}$ **do**
9 **for** $\forall \mathcal{E}_i \in \mathcal{E}$ **do**
10   $\overline{\mathbf{a}}'_{\mathbf{j}} \leftarrow \overline{\mathbf{a}}_{\mathbf{ij}} \in \mathbf{A}$
11 $\overline{\mathbf{r}}_{\mathbf{j}}$ = SCOTTKNOTTTEST($\overline{\mathbf{a}}'_{\mathbf{j}}$)
12 **for** $\forall r_{ij} \in \overline{\mathbf{r}}_{\mathbf{j}} : r_{ij} \neq 0$ **do**
13   $\mathbf{R} \leftarrow r_{ij}$
14 **for** $\forall$ *row* $\overline{\mathbf{r}}_{\mathbf{i}} \in \mathbf{R}$ **do**
15 r_{mean} = averaging the rank scores $r_{ij} \in \overline{\mathbf{r}}_{\mathbf{i}}$
16 $\mathcal{E}_{seq} \leftarrow \{\mathcal{E}_i / \mathcal{M}_i, r_{mean}\}$
17 **return** $\mathcal{E}_{seq} \leftarrow$ SORT($\mathcal{E}_{seq}$)

Algorithm 2: Meta-Training

Input: The optimal sequence $\mathcal{E}_{seq}$
Output: The meta-model $\boldsymbol{\mathcal{M}}_{meta}$

$\mathcal{M}_{meta} \leftarrow$ randomly initialized model
while *not done* **do**
 for $\forall \mathcal{E}_i \in \mathcal{E}_{seq}$ **do**
  $\mathcal{M}_{meta} \leftarrow$ TRAIN($\mathcal{M}_{meta}, \mathcal{E}_i$)
return $\mathcal{M}_{meta}$

Algorithm 3: Fine-Tuning

Input: The meta-model $\boldsymbol{\mathcal{M}}_{meta}$
Output: The fine-tuned model $\boldsymbol{\mathcal{M}}_{tuned}$

$\mathcal{E}_{target} \leftarrow$ measured data samples from the target environment
return $\mathcal{M}_{tuned} \leftarrow$ TRAIN($\mathcal{M}_{meta}, \mathcal{E}_{target}$)

whereby A_t and P_t denote the tth actual and predicted performance, respectively. A better overall MRE implies that the distribution of model parameter values learned from $\mathcal{E}_i$ is generally closer to all the optimal distributions required for fully learning the data of other meta environments, thereby serving as an indicator of the possible usefulness for the unforeseen target environment. As such, $\mathcal{E}_i$ will more likely be useful for contributing to the meta-model, enabling quicker adaptation for the unknown target environment.

Algorithm 1 (lines 2-7) demonstrates the assessment steps below:

(1) Train a single-model $\boldsymbol{\mathcal{M}}_i$ using all the available data for a meta environment $\mathcal{E}_i$ (line 2).
(2) Test single-model $\boldsymbol{\mathcal{M}}_i$ over all data for the remaining meta environments $\mathcal{E}_j$. This is repeated x times ($x = 30$ in this work as we found that using more repeats did not change the result), leading to x MRE values for each tested meta environment, denoted as a vector $\overline{\mathbf{a}}_{\mathbf{ij}}$ (line 3-7).
(3) All the $\overline{\mathbf{a}}_{\mathbf{ij}}$ are represented in a matrix $\mathbf{A}$ below:

$$\mathbf{A} = \begin{array}{c} \\ \boldsymbol{\mathcal{M}}_1 \\ \boldsymbol{\mathcal{M}}_2 \\ \vdots \\ \boldsymbol{\mathcal{M}}_n \end{array} \begin{array}{c} \begin{array}{cccc} \mathcal{E}_1 & \mathcal{E}_2 & \cdots & \mathcal{E}_n \end{array} \\ \begin{bmatrix} \emptyset & \overline{\mathbf{a}}_{12} & \cdots & \overline{\mathbf{a}}_{1n} \\ \overline{\mathbf{a}}_{21} & \emptyset & \cdots & \overline{\mathbf{a}}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathbf{a}}_{n1} & \overline{\mathbf{a}}_{n2} & \cdots & \emptyset \end{bmatrix} \end{array} \tag{6}$$

(4) Repeat from (1) till there is a single-model for every meta environment.

4.1.2 How to ensure the reliability of assessment? A naive way to sort the sequence would be to directly use the overall MRE as the metric in the comparison. This, however, entails two issues:

- Since the overall MRE covers the accuracy tested over all the remaining meta environments, the comparisons may fail to consider statistical significance even with repeated runs.

- Due to the residual nature of MRE, the single-model $\mathcal{M}_i$ has to be built by the base learner, which may not be realistic when the training is expensive (see Section 4.1.3).

Instead, in SeMPL we use Scott-Knott test [Mittas and Angelis 2013] to rank the MREs of all single-models tested on one meta environment, then average the ranks for a single-model $\mathcal{M}_i$ when testing it on all the remaining meta environments. Scott-Knott sorts the list of treatments (e.g., $\overline{\mathbf{a}}_{\mathbf{1j}}, \overline{\mathbf{a}}_{\mathbf{2j}}, ..., \overline{\mathbf{a}}_{\mathbf{nj}}$ from all single-models tested on the jth meta environment) by their average MRE. Next, it splits the list into two sub-lists with the largest difference Δ [Xia et al. 2018]:

$$\Delta = \frac{|l_1|}{|l|}(\overline{l_1} - \bar{l})^2 + \frac{|l_2|}{|l|}(\overline{l_2} - \bar{l})^2 \tag{7}$$

whereby $|l_1|$ and $|l_2|$ are the sizes of two sub-lists (l_1 and l_2) from list l with a size $|l|$. $\overline{l_1}$, $\overline{l_2}$, and $\bar{l}$ denote their mean MRE. During the splitting, bootstrapping and $\hat{A}_{12}$ [Vargha and Delaney 2000] are applied to check if l_1 and l_2 are significantly different. If that is the case, Scott-Knott recurses on the splits. In other words, the models are divided into different sub-lists if both bootstrap sampling suggests that a split is significant under a confidence level of 99% and with a good effect $\hat{A}_{12} \geq 0.6$. The sub-lists are then ranked based on their mean MRE. For example, when comparing A, B, and C, a possible split could be $\{A, B\}$, $\{C\}$, with the rank score of 1 and 2, respectively. Hence, statistically, we say that A and B perform similarly, but they are significantly better than C.

Specifically, we compute the final rank scores as in Algorithm 1 (lines 9-17):

(1) Get the repeated MREs for all single-models tested for a meta environment $\mathcal{E}_j$ (lines 9-10).
(2) Compute the vector of rank scores $\overline{\mathbf{r}}_{\mathbf{j}}$ for all single-models when testing on $\mathcal{E}_j$ (line 11).
(3) Replace the tested result for $\mathcal{E}_j$ in matrix $\mathbf{A}$ from a vector $\overline{\mathbf{a}}_{\mathbf{ij}}$ to a single value of rank score r_{ij}, forming a new matrix of ranks $\mathbf{R}$. (lines 12-13).
(4) Repeat from (1) until all tested meta environments are covered.
(5) Compute the average rank ($\overline{\mathbf{r}}_{\mathbf{i}}$) over all tested meta environments for each single-model $\mathcal{M}_i$, and put it with the corresponding meta environment learned by $\mathcal{M}_i$ in $\mathcal{E}_{seq}$ (lines 14-16).

The optimal sequence of training the meta environments can be found by sorting the average rank scores in $\mathcal{E}_{seq}$ descendingly (line 17)—the most useful meta environment is left to the last.

4.1.3 How to guarantee efficiency? Ideally, the single-model $\mathcal{M}_i$ used when assessing the usefulness of the meta environments for a target environment should be also learned by the base learner. Yet, training some base learners, such as DNN, can be expensive, especially when we need to train a single-model for each meta environment. To ensure efficiency, in SeMPL, we use linear regression as a surrogate of the base learner to build $\mathcal{M}_i$ since it has negligible overhead.

Indeed, the MRE of linear regression can differ from that of the base learner. However, through Scott-Knott test, we are mainly interested in the relative ranks between the MREs from different single-models rather than their residual accuracy. As a result, simple models like linear regression can still produce similarly coarse-grained ranks to that of a complex model [Nair et al. 2017].

Although the sequence selection needs to be rerun for every new target environment, linear regression renders the process very fast, ranging from 2 to 60 seconds for 2 to 9 meta environments.

4.2 Meta-Training

The *Meta-Training* in SeMPL learns the dataset of each meta environment sequentially according to the optimal sequence (Algorithm 2). Notably, the model parameter values trained under a preceding meta environment serve as the starting point for learning the data of the succeeding one.

In this work, we use DeepPerf [Ha and Zhang 2019]—a regularized Deep Neural Network (rDNN)—as the default base learner since it is a state-of-the-art model used for configuration performance learning. However, we would like to stress that SeMPL is agnostic to the base learner,

Table 1. Details of the subject systems and the training sizes (for fine-tuning) when an environment is used as the target. $|\mathcal{H}|$, $|\mathcal{W}|$, and $|\mathcal{V}|$ respectively denotes the number of hardware, workload, and version considered for the environments, ($|\mathcal{B}|/|\mathcal{N}|$) denotes the number of binary/numerical options, and $|\mathcal{C}|$ denotes the number of valid configurations per environment (full sample size).

System	Domain	$\lvert\mathcal{B}\rvert/\lvert\mathcal{N}\rvert$	$\lvert\mathcal{C}\rvert$	Used by	Environments			Training Size				
					$\lvert\mathcal{H}\rvert$	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{V}\rvert$	S_1	S_2	S_3	S_4	S_5
DeepArch	DNN tool	12/0	4096	[Jamshidi et al. 2018]	3	1	1	12	24	36	48	60
SaC	Cloud tool	58/0	4999	[Krishna et al. 2021]	1	3	1	58	116	174	232	290
SQLite	DBMS	14/0	1000	[Krishna et al. 2021]	2	1	2	14	28	42	56	70
NGINX	Web server	16/0	1104	[Weber et al. 2023]	1	1	4	16	32	48	64	80
SPEAR	Audio editor	14/0	16385	[Krishna et al. 2021]	3	1	1	14	28	42	56	70
Storm	Big data analyzer	1/11	2048	[Jamshidi et al. 2018]	3	1	1	158	211	522	678	1403
ImageMagick	Image editor	0/5	100	[Lesoil et al. 2023]	1	4	1	11	24	45	66	70
ExaStencils	Code generator	8/4	4098	[Weber et al. 2023]	1	4	1	106	181	366	485	695
x264	Video encoder	11/13	201	[Lesoil et al. 2023]	1	10	1	24	53	81	122	141

hence the choice of using DeepPerf is mainly due to its empirical superiority on accuracy over the others [Ha and Zhang 2019]. The base learner can be replaced when other concerns, e.g., training overhead, are of higher priority.

DeepPerf is trained in the same process as used by Ha and Zhang [2019] with hyperparameter tuning for building the single-models. We kindly refer interested readers to their work for details.

4.3 Fine-Tuning

Since we seek to exploit the learned model parameter values from the trained meta-model as the starting point in fine-tuning, SeMPL adopts the same DeepPerf as the base learner for the target environment. As in Algorithm 3, the fine-tuning follows standard training of a typical machine learning model using samples for the target environment; yet, instead of using an initial model with random parameter values, the meta-model directly serves as the starting point. As a result, learning under the target environment can be greatly expedited and improved as the knowledge from meta environments should have been generalized. The training sample size from the target environment can vary, for which we examine different sizes as what will be discussed in Section 5. The newly given configurations can be fed into the fine-tuned meta-model to predict their performance.

5 EXPERIMENT SETUP

Our experiments evaluate SeMPL by answering the following research questions (RQs):

- **RQ$_1$:** How does SeMPL perform compared with existing single environment models?
- **RQ$_2$:** How does SeMPL perform compared with the state-of-the-art models that handle multiple environments?
- **RQ$_3$:** How effective is the sequence selection in SeMPL?
- **RQ$_4$:** What is the sensitivity of SeMPL's accuracy to the sample size used in pre-training?

5.1 Subject Software Systems and Environments

In the experiments, we study widely-used multi-environment configuration datasets collected from real-world systems[3] [Jamshidi et al. 2017a, 2018; Krishna et al. 2021; Lesoil et al. 2023]. These systems are selected based on the following:

- To ensure diversity, systems with less than three environments are removed.

[3]To ensure reliability, the data has been collected with repeated measurements.

- In the presence of multiple datasets for the same system, the one that contains the most deviated measurements between the environments is used.

Within the identified systems, we rule out the data of environments that (1) do not measure all valid configurations; (2) contain invalid measurements; or (3) lack detailed specifications.

As shown in Table 1, the above process leads to nine systems of diverse domains, scales, and option types, together with configurations measured in distinct valid environments of hardware, workloads, and/or versions. To further improve the external validity, we examine the models under different training sample sizes from the target environment. For binary systems (only binary options), we use five sizes as prior work [Ha and Zhang 2019]: $\{x, 2x, 3x, 4x, 5x\}$ where x is the number of options. For mixed systems (both binary and numeric options), we use five sizes as produced by the sampling method from the work of Siegmund et al. [2012b]. These are the common methods used in the field [Ha and Zhang 2019; Shu et al. 2020].

5.2 Procedure, Metric and Statistical Validation

5.2.1 Procedure. For each system and comparative approach, we follow the steps below:

(1) Pick an environment as the target $\mathcal{E}_{target}$ and set all remaining ones as meta environments.
(2) If applicable, pre-training on all data samples of the meta environments (except for RQ_4).
(3) Pick a training sample size S_i for $\mathcal{E}_{target}$.
(4) Train/fine-tune a model under S_i randomly selected samples and test it on all the remaining (unforeseen) samples of $\mathcal{E}_{target}$.
(5) To mitigate bias, repeat (4) for 30 runs via bootstrapping without replacement[4].
(6) Repeat from (3) to cover all training sample sizes of $\mathcal{E}_{target}$.
(7) Repeat from (1) till every environment has served as the target environment once. In this way, we avoid bias towards the prediction of a particular environment.

As such, for each of the nine systems, we have five different training sizes and 3–10 alternative target environments, leading to 15–50 cases of comparisons. In SeMPL, we set the iteration limit for the outer loop as 1, which is the same default for MAML.

Noteworthily, in this work, we assume the configurations are the same for different meta environments for all approaches, which is derived from the existing datasets and is the common assumption from existing work, e.g., BEETLE [Krishna et al. 2021]. However, since the learning of data from the meta environments is independent of each other in SeMPL (i.e., the information sharing happens at the model parameter level rather than at the data level), it is not restricted to this assumption and can directly learn data from the meta-environments for which the configurations in the data samples are not the same.

5.2.2 Accuracy. As with the sequence selection, we use MRE to assess the accuracy, which is the standard practice [Chen et al. 2021; Guo et al. 2018; Ha and Zhang 2019; Queiroz et al. 2016; Valov et al. 2015]. Further, MRE is insensitive to the scale of performance metrics.

5.2.3 Speedup. As in prior work [Chen et al. 2021; Chen and Li 2021, 2024], to assess the training speedup achieved by SeMPL for a system, we use a baseline, b, taken as the smallest training size that the best state-of-the-art counterpart consumes to achieve its best mean MRE. We then find the smallest size for SeMPL to achieve the same accuracy, denoted as s. The ratios, i.e., $sp = \frac{b}{s}$, are reported. Clearly, if SeMPL is more data-efficient, then we would expect $sp \geq 1\times$. When SeMPL cannot achieve the same mean MRE for all sample sizes, we have $sp = N/A$.

[4] 30 runs is the most common setting in the field of configuration performance learning [Ha and Zhang 2019; Shu et al. 2020]. This number of runs is merely a pragmatic choice given the resource constraint.

5.2.4 Statistics. To make a fair comparison between more than two models, we again use the Scott-Knott test [Mittas and Angelis 2013] to evaluate their statistical significance on the difference of MRE over 30 runs, as recommended by Mittas and Angelis [2013]. In a nutshell, Scott-Knott test provides the following benefits over the other statistical methods:

- Unlike those parametric tests such as the t-test [Student 1908], it has no assumption on the data distribution.
- It allows the assessment of more than two models as opposed to the other pair-wise nonparametric tests such as the Wilcoxon test [Wilcoxon 1945] or Mann-Whitney U test [Mann and Whitney 1947].
- It naturally ranks the model while other multiple comparison tests, e.g., Kruskal-Wallis test [McKight and Najab 2010], only assess the differences between models without comprising the better or worse. Further, the Scott-Knott test does not need post-hoc correlation.

6 EVALUATION

We now present and discuss the experiment results for the RQs.

6.1 SeMPL against Single Environment Models

6.1.1 Method. To answer $\mathbf{RQ}_1$, we compare SeMPL against five state-of-the-art single environment models, i.e., DeepPerf, DECART, RF, SPLConqueror, and XGBoost, as discussed in Section 2.1. To ensure fairness, we use the code published by their authors. All settings from Section 5 are used.

6.1.2 Results. As can be seen in Figure 5, on nearly all systems, SeMPL significantly improves the MRE by up to 81%, 87%, 84%, 99% and 91% over DeepPerf, RF, DECART, SPLConqueror and XGBoost, respectively. As the default base learner for SeMPL, DeepPerf might occasionally perform worse even with more data. This is because the target environments exhibit diverse data patterns, which can easily cause the hyperparameter tuning to be trapped at local optima. Such an issue can be mitigated when paired with SeMPL since the meta-model is fine-tuned from some good starting points of the parameters. In particular, SeMPL often achieves considerably better MRE in very few data samples, commonly with a significant speedup that can be up to 3.86×. This confirms the generalization efficiency of sequential meta-learning. Remarkably, from the average Scott-Knott ranks, SeMPL is the best for 8 out of 9 systems (i.e., 89%), demonstrating its superiority over the single environment models for learning configuration performance, thanks to “learning to learn".

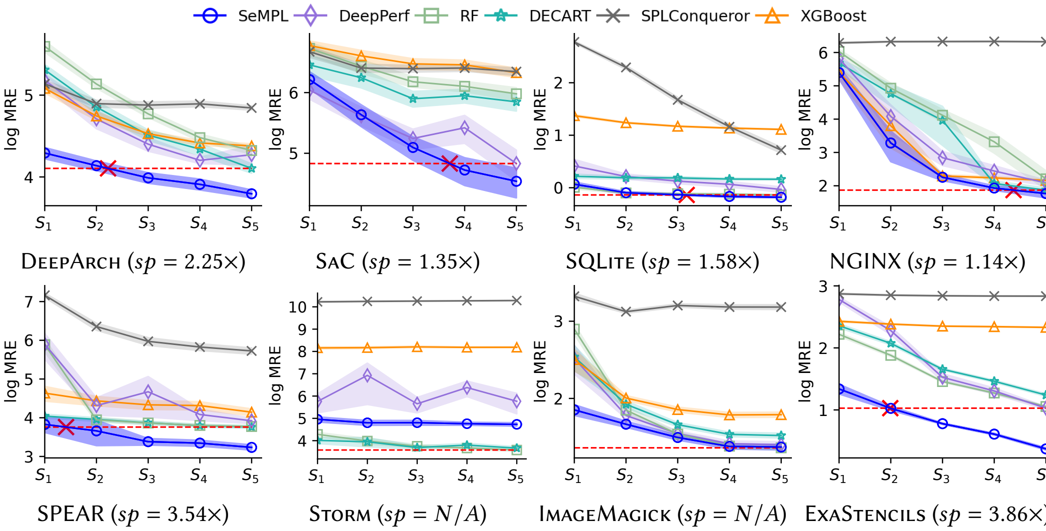
Noteworthy, SeMPL does not achieve the highest rank for Storm, which can be attributed to a combination of factors including the sparsity of samples (due to the nature of the system) and the presence of more diverse environments, primarily limited to hardware changes.

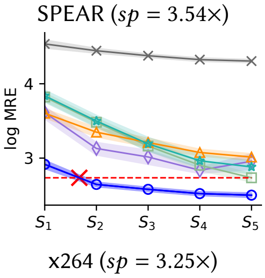
> ***RQ_1:*** SeMPL *performs significantly better than the state-of-the-art single environment performance models in 8 out of 9 systems with the best MRE improvement of* 99%*; it is also generally data-efficient with up to* 3.86× *speedup.*

6.2 SeMPL against Multi-Environment Models

6.2.1 Method. To study $\mathbf{RQ}_2$, we assess SeMPL against state-of-the-art models based on transfer learning (BEETLE and tEAMS), multi-task learning (MORF), and meta-learning (MAML and MetaSGD), as discussed in Section 2.3 and 2.4. Again, the same source code published by their authors is used. We also examine the approaches that take environmental features as additional inputs, i.e., DeepPerf+$_e$, RF+$_e$, and SPLConqueror+$_e$ (see Section 2.2).

Since MAML, MetaSGD, BEETLE, and tEAMS are agnostic to the base learner, we pair them with DeepPerf, which is the same default for SeMPL. Their pre-training data is also identical to that of





Model	DeepArch	SaC	SQLite	NGINX	SPEAR	Storm	ImageMagick	ExaStencils	x264
SeMPL	**1.0**	**1.53**	**1.15**	**1.45**	**1.07**	3.2	**1.3**	**1.0**	**1.02**
DeepPerf	2.4	2.27	2.8	2.55	3.13	3.53	1.9	3.2	2.26
RF	4.0	2.47	1.4	3.45	2.13	**1.13**	2.05	2.1	2.78
DECART	2.87	1.67	3.15	2.4	1.87	1.73	3.05	3.35	3.18
SPLConqueror	4.0	4.2	5.05	4.65	4.4	5.87	5.0	5.7	4.62
XGBoost	2.93	3.87	4.6	1.95	3.0	4.87	3.8	4.5	3.04

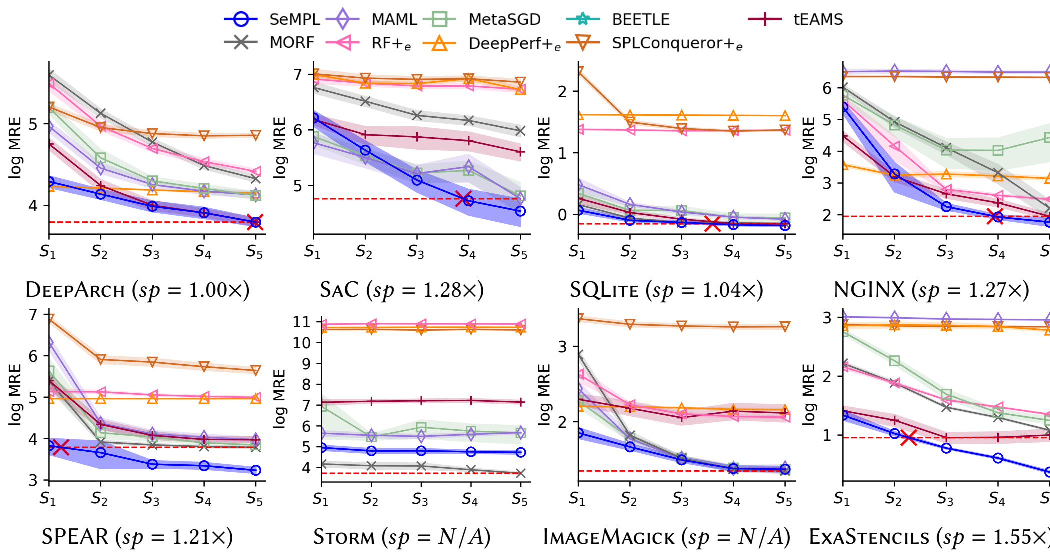
Mean Scott-Knott ranks over all cases/runs. blue cell denotes the best.

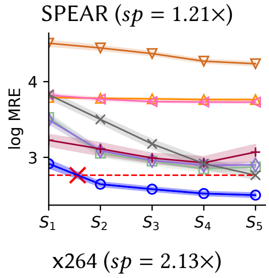
Fig. 5. SeMPL versus single environment models. For the simplicity of exposition, we report the log-transformed average MRE (and its standard error) of all target environments and runs. For speedup ($sp = \frac{b}{s}$), - - - denotes the mean MRE for b; × indicates the point of s. Detailed data can be accessed at: https://github.com/ideas-labo/SeMPL/blob/main/Figure5_full.pdf.

SeMPL where applicable. All other settings are the same as **RQ**$_1$. Note that given such a setting, all the compared approaches use the same amount of data in training/fine-tuning as that of SeMPL, including those from the target environment.

6.2.2 Results. From Figure 6, we see that in general, SeMPL is mostly more accurate than, or at least similar to, the best state-of-the-art model that handles multiple environments. The best improvement ranges from 74% (for SPEAR) to 99% (for NGINX). In terms of data efficiency, SeMPL shows considerable speedup with up to 2.13×. We observe similar findings on the Scott-Knott test: overall, SeMPL is ranked the best for 89% of the systems (8 out of 9 systems). Meanwhile, although the best rank for Storm is MORF, we can still see that SeMPL ranks the second in general.

These results match with our theory: transfer learning models like BEETLE and tEAMS are restricted by their weak generalizability; while multi-task learning models like MORF overcomplicate the training, which makes the accuracy suffer; meta-learning models like MAML and MetaSGD fail to discriminate the contributions between different meta environments when pre-training the meta-model. All of the above shortcomings are what SeMPL seeks to tackle. In particular, models that take the environments as additional features generally perform badly (e.g., DeepPerf+$_e$), due to the fact that this unnecessarily increases the difficulty of training.





Model	DeepArch	SaC	SQLite	NGINX	SPEAR	Storm	ImageMagick	ExaStencils	x264
SeMPL	**1.67**	**1.2**	**1.2**	**1.55**	**1.6**	2.73	**1.9**	**1.15**	**1.48**
MAML	3.53	2.67	3.95	7.0	4.6	3.73	2.8	7.65	4.16
MetaSGD	4.2	1.93	3.4	4.65	3.87	3.4	2.5	4.0	3.9
BEETLE	3.67	2.4	2.25	3.1	2.07	3.4	2.5	4.75	2.04
tEAMS	1.8	3.47	2.55	1.85	4.67	4.87	4.4	1.85	4.2
MORF	5.73	2.4	1.85	4.25	3.07	**1.0**	3.25	2.95	4.78
RF+$_e$	5.93	5.07	5.15	2.8	4.47	7.73	4.7	3.65	4.18
DeepPerf+$_e$	2.8	5.8	6.5	3.0	4.6	7.07	3.7	6.3	4.06
SPLConqueror+$_e$	6.13	6.4	6.0	6.0	7.13	6.4	6.7	6.4	7.2

Mean Scott-Knott ranks over all cases/runs. blue cell denotes the best.

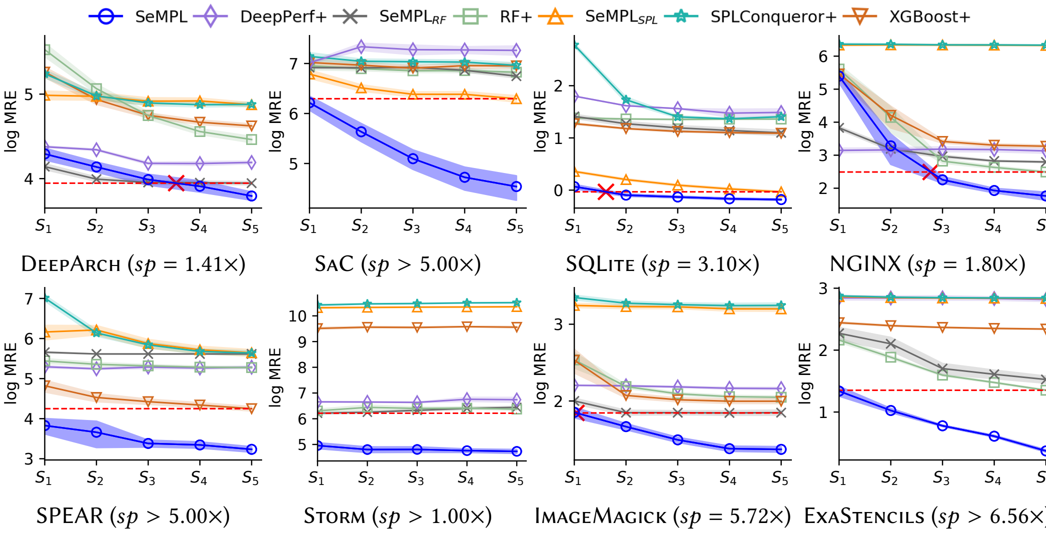
Fig. 6. SeMPL versus multi-environment models. The format is the same as Figure 5. Detailed MRE and ranks can be accessed at: https://github.com/ideas-labo/SeMPL/blob/main/Figure6_full.pdf.

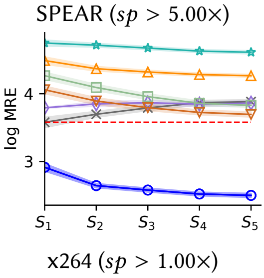
> ***RQ$_2$:*** Compared with the state-of-the-art models that handles multiple environments, SeMPL is more effective on 89% of the systems with the best MRE improvement from 74% to 99% while being data-efficient with mostly $sp \geq 1$ and the best speedup of 2.13×. In particular, this evidences that the sequential training in SeMPL, which discriminates the contributions of each meta environment, is more beneficial for configuration performance learning than the parallel training (e.g., in MAML) that treats all meta environments equally.

6.3 Effectiveness of Sequence Selection

6.3.1 Method. To confirm the necessity of sequence selection in SeMPL, for **RQ$_3$**, we equip the other models from **RQ$_1$** with the sequential meta-learning process using all data of the meta environments in random orders, denoted as DeepPerf+, RF+, SPLConqueror+, and XGBoost+. Thus, they will have access to exactly the same amount of data as SeMPL, including those from the target enforcement for fine-tuning. Note that we cannot consider DECART because its mechanism requires initializing the model from scratch, which is incompatible with the concept of fine-tuning.

To eliminate the noise caused by the default base learner, we also additionally pair SeMPL with two alternative base learners, i.e., RF and SPLConqueror, denoted as SeMPL$_{RF}$ and SeMPL$_{SPL}$, respectively. All other settings are the same as the previous RQs.





Model	DeepArch	SaC	SQLite	NGINX	SPEAR	Storm	ImageMagick	ExaStencils	x264
SeMPL	1.53	**1.0**	**1.0**	**1.6**	**1.73**	**1.33**	**1.3**	**1.0**	**1.18**
DeepPerf+	2.47	5.8	5.05	2.85	2.8	3.4	3.0	4.85	2.92
SeMPL$_{RF}$	**1.47**	3.53	4.1	2.4	3.47	2.2	2.9	2.45	2.82
RF+	5.0	3.07	3.7	2.65	3.07	2.67	3.45	2.15	3.82
SeMPL$_{SPL}$	4.93	2.27	2.0	5.4	5.0	5.6	5.75	5.05	5.26
SPLConqueror+	5.4	5.27	5.25	5.9	5.47	6.27	5.95	5.25	6.04
XGBoost+	4.4	4.6	4.8	3.95	2.67	5.07	3.7	3.5	4.02

Mean Scott-Knott ranks over all cases/runs. blue cell denotes the best.

Fig. 7. Optimal sequence versus random order in SeMPL. The format is the same as Figure 5. Detailed MRE and ranks can be accessed at: https://github.com/ideas-labo/SeMPL/blob/main/Figure7_full.pdf.

6.3.2 Result. From Figure 7, we see that SeMPL, SeMPL$_{RF}$ and SeMPL$_{SPL}$ generally perform better than their counterparts on 9 (100%), 5 (56%), and 9 (100%) out of 9 systems, respectively. From this, we confirm the importance of obtaining the optimal sequence for the sequential learning in SeMPL, as a random order would likely amplify the side-effect of some badly performing meta environments. Clearly, using DeepPerf as the base learner in SeMPL dramatically boosts the accuracy across nearly all training sizes on 8 out of 9 systems with the best speedup of more than 6.56×. Note that while for DeepArch, SeMPL has the second-best overall rank, it is very close to the best counterpart.

Interestingly, SeMPL$_{RF}$ might be exacerbated with more training data, e.g., for x264. This is because the base learner, RF, still suffers from the issue of overfitting under the sample sizes considered, which, when updating the meta-models over multiple meta environments, can be harmful due to the cumulatively overfitted model parameters. It is also the reason that SeMPL$_{RF}$ does not perform as well as RF+ on some systems.

> ***RQ$_3$:*** The sequence selection helps to improve the results on between 56% and 100% of the systems based on the base learner. Notably, pairing SeMPL with DeepPerf achieves the best results overall and leads to considerable speedup.

6.4 Sensitivity to the Pre-Training Size

6.4.1 Method. Instead of full pre-training sizes, in **RQ**$_4$, we examine the MRE at 0%, 25%, ..., 100% of the random samples from the full datasets of meta environments, where 0% means no meta-learning.

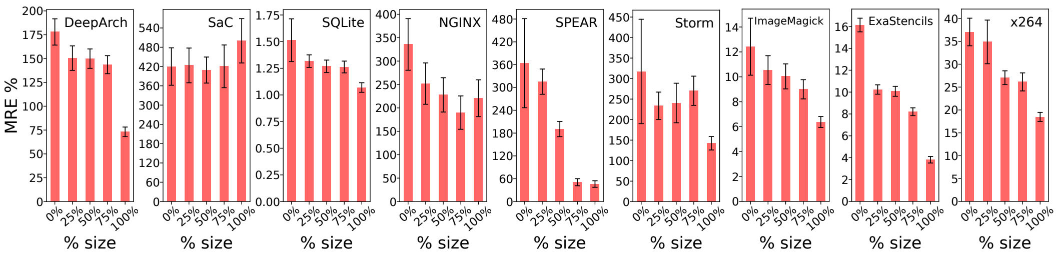


Fig. 8. Mean MRE and standard error under different percentages of pre-training sizes over 30 runs (the sample size at the fine-tuning stage is S_1, which is the smallest since we seek to stress test the effect).

6.4.2 Results. In Figure 8, we note that there is generally a monotonic correlation between the sample size of meta environments and the accuracy: the MRE decreases with more data samples used in the pre-training of SeMPL. In particular, the case of 0% leads to the worst results on 8 out of 9 systems, suggesting the importance of meta-learning. The only exception is SaC which makes SeMPL suffer the "curse of dimensionality" more since it has 58 configuration options, whereas the others range from 5 to 24. Yet, the biggest improvement is case-dependent since the relative difference in contributions of the meta environments to the learning can vary across the systems.

> ***RQ*$_4$*:*** While more data samples in the pre-training can be generally beneficial to SeMPL but the biggest improvement is case-dependent.

7 DISCUSSION

7.1 What Make SeMPL Special for Configuration Performance Learning?

The proposed sequential pre-training is a specialization of meta-learning for the unique characteristics of configuration data between different environments (as discussed in Section 3). As such, SeMPL is specifically designed based on our understanding of the software engineering task at hand.

Indeed, we do not claim that SeMPL can completely eliminate the potentially misleading meta environments, nor it is feasible to do so since this may incur significant overhead. However, with the sequential pre-training, SeMPL is able to at least mitigate the effect of those misleading meta environments by placing them earlier in the sequence. This, as we have shown, can obtain significant improvement in accuracy over classic frameworks like MAML, which treats all the meta environments equally. As we discussed in Section 3, even though certain meta environments may be misleading overall, some of their parameter distribution could still fit well with those needed for the target environment. Hence, mitigating their effect could be more suitable than their complete omission.

7.2 Practical Application and Overhead of SeMPL

SeMPL was designed under the assumption that there are available data measured under different environments of a running configurable software system, which is not uncommon [Jamshidi et al. 2018; Krishna et al. 2021; Lesoil et al. 2023; Weber et al. 2023]. Therefore, the applications of SeMPL in real-world cases are straightforward. Thanks to the sequential pre-training, the contributions of meta environments can be discriminated with no manual selection required, hence the more useful ones will contribute more to the learning. Of course, like any learning models, software engineers would need to decide on the training sample size to be measured for the target environment, together with whether the default base learner in SeMPL, i.e., DeepPerf that favors accuracy, is suitable for the specific scenario as the training of the base learner would constitute the majority of the SeMPL's overhead during pre-training.

On a machine with CPU 2GHz and 16GB RAM, SeMPL only needs 2 and 60 seconds to find the optimal sequence of learning meta environments on a system with 3 and 10 meta environments, respectively, thanks to the linear regression. Of course, the overhead can increase with more meta environments, but the magnitude is still acceptable since the pre-training is an offline process.

7.3 Threats to Validity

Internal threats is concerned with the parameter settings, e.g., the training size. We mitigate this by following the same settings used in the state-of-the-art studies [Guo et al. 2013; Ha and Zhang 2019; Shu et al. 2020; Siegmund et al. 2012b]. We also examined the sensitivity of SeMPL to the pre-training sample size in RQ_4. SeMPL is tested with DeepPerf as the base learner, which can be flexibly replaced. Of course, as in RQ_3, replacing DeepPerf could lead to different results but this is up to the software engineers to decide if other factors, such as training efficiency, are more important. The use of linear regression as a surrogate of the base learner might also raise threats to internal validity since it might not be perfectly accurate. However, our results show this still leads to significant improvement compared with the state-of-the-art.

Construct threats are mainly related to the evaluation metric. In this work, we use MRE as the main accuracy metric as it is less sensitive to the scales in different environments and systems while being the most common metric from the literature of configuration performance learning [Guo et al. 2018; Ha and Zhang 2019; Shu et al. 2020]. To measure the speedup, we follow the same protocol as used in prior work [Chen et al. 2021; Chen and Li 2021] from the field.

External threats could lie in the systems and environments studied. To mitigate such, we exploit the most commonly used datasets for diverse systems [Jamshidi et al. 2018; Krishna et al. 2021; Lesoil et al. 2023]. For each of these, we select all the eligible environments for evaluation. The experiments are also repeated 30 times, which tends to be sufficient to reduce randomness. The Scott-Knott test is adopted to ensure the statistical significance of the results. However, we acknowledge that using more systems and environments might be beneficial.

8 CONCLUSION

To deal with multiple environments when predicting performance for configurable software systems, this paper proposes a new category of framework that leverages meta-learning, dubbed SeMPL. What makes SeMPL unique is that it learns the meta environments, one at a time, in a specific order according to the likely usefulness of the meta environment to the unforeseen target environment. Such sequential learning, unlike the existing parallel learning, naturally allows the pre-training to discriminate the contributions between meta environments, thereby handling the largely deviated samples from different environments—a key characteristic of the configuration data. Extensive experiments on nine widely-used systems in current studies demonstrate that SeMPL significantly outperforms state-of-the-art single/multi-environment models with considerable speedup. As such, SeMPL is more useful for predicting configuration performance in new environments.

In the future, we hope that the concept behind SeMPL will inspire a vast direction of research for meta-performance learning, including heterogeneous meta-learning and more precise management of the sample variations between environments in configuration data.

DATA AVAILABILITY

All source code and data can be found at our repository: https://github.com/ideas-labo/SeMPL.

ACKNOWLEDGMENTS

This work was supported by a UKRI Grant (10054084) and a NSFC Grant (62372084).

REFERENCES

Sami Alabed and Eiko Yoneki. 2021. High-Dimensional Bayesian Optimization with Multi-Task Learning for RocksDB. In *EuroMLSys@EuroSys 2021, Proceedings of the 1st Workshop on Machine Learning and Systemsg Virtual Event, Edinburgh, Scotland, UK, 26 April, 2021*, Eiko Yoneki and Paul Patras (Eds.). ACM, 111–119. https://doi.org/10.1145/3437984.3458841

Leo Breiman. 2017. *Classification and regression trees.* Routledge.

Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, Pooyan Jamshidi, Reiner Jung, Jóakim von Kistowski, Anne Koziolek, Johannes Kroß, Simon Spinner, Christian Vögele, Jürgen Walter, and Alexander Wert. 2015. Performance-oriented DevOps: A Research Agenda. *SPEC-RG-2015-01, RG DevOps Performance* abs/1508.04752 (2015). arXiv:1508.04752 http://arxiv.org/abs/1508.04752

Yitian Chai, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2022. Cross-Domain Deep Code Search with Meta Learning. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 487–498. https://doi.org/10.1145/3510003.3510125

Junjie Chen, Ningxin Xu, Peiqi Chen, and Hongyu Zhang. 2021. Efficient Compiler Autotuning via Bayesian Optimization. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1198–1209. https://doi.org/10.1109/ICSE43902.2021.00110

Lisha Chen and Tianyi Chen. 2022. Is Bayesian Model-Agnostic Meta Learning Better than Model-Agnostic Meta Learning, Provably?. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event (Proceedings of Machine Learning Research, Vol. 151)*, Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (Eds.). PMLR, 1733–1774. https://proceedings.mlr.press/v151/chen22d.html

Tao Chen. 2019. All versus one: an empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Marin Litoiu, Siobhán Clarke, and Kenji Tei (Eds.). ACM, 157–168. https://doi.org/10.1109/SEAMS.2019.00029

Tao Chen. 2022. Planning Landscape Analysis for Self-Adaptive Systems. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2022, Pittsburgh, PA, USA, May 22-24, 2022*, Bradley R. Schmerl, Martina Maggio, and Javier Cámara (Eds.). ACM/IEEE, 84–90. https://doi.org/10.1145/3524844.3528060

Tao Chen and Rami Bahsoon. 2017a. Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services. *IEEE Trans. Software Eng.* 43, 5 (2017), 453–475. https://doi.org/10.1109/TSE.2016.2608826

Tao Chen and Rami Bahsoon. 2017b. Self-Adaptive Trade-off Decision Making for Autoscaling Cloud-Based Services. *IEEE Trans. Serv. Comput.* 10, 4 (2017), 618–632. https://doi.org/10.1109/TSC.2015.2499770

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 785–794. https://doi.org/10.1145/2939672.2939785

Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Trans. Softw. Eng. Methodol.* 27, 2 (2018), 5:1–5:50.

Tao Chen and Miqing Li. 2021. Multi-objectivizing software configuration tuning. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 453–465. https://doi.org/10.1145/3468264.3468555

Tao Chen and Miqing Li. 2023a. Do Performance Aspirations Matter for Guiding Software Configuration Tuning? An Empirical Investigation under Dual Performance Objectives. *ACM Trans. Softw. Eng. Methodol.* 32, 3 (2023), 68:1–68:41. https://doi.org/10.1145/3571853

Tao Chen and Miqing Li. 2023b. The Weights Can Be Harmful: Pareto Search versus Weighted Search in Multi-Objective Search-Based Software Engineering. *ACM Transactions on Software Engineering and Methodology* 32, 1, Article 5 (feb 2023), 40 pages. https://doi.org/10.1145/3514233

Tao Chen and Miqing Li. 2024. Adapting Multi-objectivized Software Configuration Tuning. *FSE'24: Proceedings of the ACM on Software Engineering (PACMSE)* 1, FSE. https://doi.org/10.1145/3643751

Kurtland Chua, Qi Lei, and Jason D. Lee. 2021. How Fine-Tuning Allows for Effective Meta-Learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 8871–8884. https://proceedings.neurips.cc/paper/2021/hash/4a533591763dfa743a13affab1a85793-Abstract.html

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1126–1135. http://proceedings.mlr.press/v70/finn17a.html

Jingzhi Gong and Tao Chen. 2022. Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 482–494. https://doi.org/10.1145/3524842.3528431

Jingzhi Gong and Tao Chen. 2023. Predicting Software Performance with Divide-and-Learn. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, Satish Chandra, Kelly Blincoe, and Paolo Tonella (Eds.). ACM, 858–870. https://doi.org/10.1145/3611643.3616334

Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, Ewen Denney, Tevfik Bultan, and Andreas Zeller (Eds.). IEEE, 301–311. https://doi.org/10.1109/ASE.2013.6693089

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).

Huong Ha and Hongyu Zhang. 2019. DeepPerf: performance prediction for configurable software with deep sparse neural network. In *Proceedings of the 41st International Conference on Software Engineering*. 1095–1106.

Xue Han and Tingting Yu. 2016. An Empirical Study on Performance Bugs for Highly Configurable Software Systems. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016, Ciudad Real, Spain, September 8-9, 2016*. ACM, 23:1–23:10. https://doi.org/10.1145/2961111.2962602

Francesco Iorio, Ali B. Hashemi, Michael Tao, and Cristiana Amza. 2019. Transfer Learning for Cross-Model Regression in Performance Modeling for the Cloud. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, Australia, December 11-13, 2019*. IEEE, 9–18. https://doi.org/10.1109/CloudCom.2019.00015

Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. 2022. Unicorn: reasoning about configurable system performance through the lens of causality. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis (Eds.). ACM, 199–217. https://doi.org/10.1145/3492321.3519575

Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017a. Transfer learning for performance modeling of configurable systems: an exploratory analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 497–508. https://doi.org/10.1109/ASE.2017.8115661

Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to sample: exploiting similarities across environments to learn performance models for configurable systems. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 71–82. https://doi.org/10.1145/3236024.3236074

Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017b. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE*. 31–41.

Ugur Koc, Austin Mordahl, Shiyi Wei, Jeffrey S. Foster, and Adam A. Porter. 2021. SATune: A Study-Driven Auto-Tuning Approach for Configurable Software Verification Tools. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 330–342. https://doi.org/10.1109/ASE51524.2021.9678761

Rahul Krishna, Vivek Nair, Pooyan Jamshidi, and Tim Menzies. 2021. Whence to Learn? Transferring Knowledge in Configurable Systems Using BEETLE. *IEEE Trans. Software Eng.* 47, 12 (2021), 2956–2972. https://doi.org/10.1109/TSE.2020.2983927

Philipp Leitner and Jürgen Cito. 2016. Patterns in the Chaos - A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Trans. Internet Techn.* 16, 3 (2016), 15:1–15:23. https://doi.org/10.1145/2885497

Christian Lengauer, Sven Apel, Matthias Bolten, Armin Größlinger, Frank Hannig, Harald Köstler, Ulrich Rüde, Jürgen Teich, Alexander Grebhahn, Stefan Kronawitter, Sebastian Kuckuk, Hannah Rittich, and Christian Schmitt. 2014. ExaStencils: Advanced Stencil-Code Engineering. In *Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 8806)*, Luís M. B. Lopes, Julius Zilinskas, Alexandru Costan, Roberto G. Cascella, Gabor Kecskemeti, Emmanuel Jeannot, Mario Cannataro, Laura Ricci, Siegfried Benkner, Salvador Petit, Vittorio Scarano, José Gracia, Sascha Hunold, Stephen L. Scott, Stefan Lankes, Christian Lengauer, Jesús Carretero, Jens Breitbart, and Michael Alexander (Eds.). Springer, 553–564. https://doi.org/10.1007/978-3-319-14313-2_47

Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2023. Input sensitivity on the performance of configurable systems an empirical study. *Journal of Systems and Software* (2023), 111671. https://doi.org/10.1016/j.jss.2023.111671

Miqing Li, Tao Chen, and Xin Yao. 2022. How to Evaluate Solutions in Pareto-Based Search-Based Software Engineering: A Critical Review and Methodological Guidance. *IEEE Trans. Software Eng.* 48, 5 (2022), 1771–1799. https://doi.org/10.1109/TSE.2020.3036108

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2017. Meta-SGD: Learning to Learn Quickly for Few Shot Learning. *CoRR* abs/1707.09835 (2017). arXiv:1707.09835 http://arxiv.org/abs/1707.09835

K. Madhumathi and T. Suresh. 2018. Node Localization in Wireless Sensor Networks Using Multi-output Random Forest Regression. In *Soft Computing for Problem Solving - SocProS 2018, Volume 2, VIT Vellore, India, December 17-19, 2018 (Advances in Intelligent Systems and Computing, Vol. 1057)*, Kedar Nath Das, Jagdish Chand Bansal, Kusum Deep, Atulya K. Nagar, Ponnambalam Pathipooranam, and Rani Chinnappa Naidu (Eds.). Springer, 177–186. https://doi.org/10.1007/978-981-15-0184-5_16

Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics* 18, 1 (1947), 50–60.

Hugo Martin, Mathieu Acher, Juliana Alves Pereira, Luc Lesoil, Jean-Marc Jézéquel, and Djamel Eddine Khelladi. 2022. Transfer Learning Across Variants and Versions: The Case of Linux Kernel Size. *IEEE Trans. Software Eng.* 48, 11 (2022), 4274–4290. https://doi.org/10.1109/TSE.2021.3116768

Patrick E McKight and Julius Najab. 2010. Kruskal-wallis test. *The corsini encyclopedia of psychology* (2010), 1–1.

Nikolaos Mittas and Lefteris Angelis. 2013. Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm. *IEEE Trans. Software Eng.* 39, 4 (2013), 537–551. https://doi.org/10.1109/TSE.2012.45

Stefan Mühlbauer, Florian Sattler, Christian Kaltenecker, Johannes Dorn, Sven Apel, and Norbert Siegmund. 2023. Analyzing the Impact of Workloads on Modeling the Performance of Configurable Software Systems. In *ICSE'23: Proc. of the 45th International Conference on Software Engineering*. ACM.

Vivek Nair, Rahul Krishna, Tim Menzies, and Pooyan Jamshidi. 2018. Transfer Learning with Bellwethers to find Good Configurations. *CoRR* abs/1803.03900 (2018). arXiv:1803.03900 http://arxiv.org/abs/1803.03900

Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using bad learners to find good configurations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 257–267. https://doi.org/10.1145/3106237.3106238

Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2020. Finding Faster Configurations Using FLASH. *IEEE Trans. Software Eng.* 46, 7 (2020), 794–811. https://doi.org/10.1109/TSE.2018.2870895

Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. In *ICPE '20: ACM/SPEC International Conference on Performance Engineering, Edmonton, AB, Canada, April 20-24, 2020*, José Nelson Amaral, Anne Koziolek, Catia Trubiani, and Alexandru Iosup (Eds.). ACM, 277–288. https://doi.org/10.1145/3358960.3379137

Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2016. Towards predicting feature defects in software product lines. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development, FOSD@SPLASH 2016, Amsterdam, Netherlands, October 30, 2016*, Christoph Seidl and Leopoldo Teixeira (Eds.). ACM, 58–62. https://doi.org/10.1145/3001867.3001874

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2020. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=rkgMkCEtPB

Mohammed Sayagh, Noureddine Kerzazi, Bram Adams, and Fábio Petrillo. 2020. Software Configuration Engineering in Practice Interviews, Survey, and Systematic Literature Review. *IEEE Trans. Software Eng.* 46, 6 (2020), 646–673. https://doi.org/10.1109/TSE.2018.2867847

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, Bari, Italy, October 5-7, 2020*, Maria Teresa Baldassarre, Filippo Lanubile, Marcos Kalinowski, and Federica Sarro (Eds.). ACM, 16:1–16:11. https://doi.org/10.1145/3382494.3410677

Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don S. Batory, Marko Rosenmüller, and Gunter Saake. 2012a. Predicting performance via automated feature-interaction detection. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, Martin Glinz, Gail C. Murphy, and Mauro Pezzè (Eds.). IEEE Computer Society, 167–177. https://doi.org/10.1109/ICSE.2012.6227196

Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012b. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Softw. Qual. J.* 20, 3-4 (2012), 487–517. https://doi.org/10.1007/s11219-011-9152-9

Trevor Standley, Amir Zamir, Dawn Chen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. 2020. Which Tasks Should Be Learned Together in Multi-task Learning?. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 9120–9132. http://proceedings.mlr.press/v119/standley20a.html

Student. 1908. The probable error of a mean. *Biometrika* (1908), 1–25.

Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 242–264.

Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. 2015. Empirical comparison of regression methods for variability-aware performance prediction. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, Douglas C. Schmidt (Ed.). ACM, 186–190. https://doi.org/10.1145/2791060.2791069

Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*, Walter Binder, Vittorio Cortellessa, Anne Koziolek, Evgenia Smirni, and Meikel Poess (Eds.). ACM, 39–50. https://doi.org/10.1145/3030207.3030216

András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong.

Ricardo Vilalta and Youssef Drissi. 2002. A Perspective View and Survey of Meta-Learning. *Artif. Intell. Rev.* 18, 2 (2002), 77–95. https://doi.org/10.1023/A:1019956318069

Max Weber, Christian Kaltenecker, Florian Sattler, Sven Apel, and Norbert Siegmund. 2023. Twins or False Friends? A Study on Energy Consumption and Performance of Configurable Software. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2098–2110. https://doi.org/10.1109/ICSE48619.2023.00177

Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods.

Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. 2018. Hyperparameter Optimization for Effort Estimation. *CoRR* abs/1805.00336 (2018). arXiv:1805.00336 http://arxiv.org/abs/1805.00336

Yongxin Yang and Timothy M. Hospedales. 2017. Deep Multi-task Representation Learning: A Tensor Factorisation Approach. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SkhU2fcll

Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. 2019. Hierarchically Structured Meta-learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 7045–7054. http://proceedings.mlr.press/v97/yao19b.html

Received 2023-09-28; accepted 2024-01-23