Dually Hierarchical Drift Adaptation for Online Configuration Performance Learning

Zezhen Xiang*
zz.xiang.work@gmail.com
School of Computer Science and
Engineering
University of Electronic Science and
Technology of China
Chengdu, China

Jingzhi Gong
j.gong@leeds.ac.uk
School of Computer Science
University of Leeds
Leeds, UK

Tao Chen†
t.chen@bham.ac.uk
IDEAS Lab
School of Computer Science
University of Birmingham
Birmingham, UK

Abstract

Modern configurable software systems need to learn models that correlate configuration and performance. However, when the system operates in dynamic environments, the workload variations, hardware changes, and system updates will inevitably introduce *concept drifts* at different levels—*global drifts*, which reshape the performance landscape of the entire configuration space; and *local drifts*, which only affect certain sub-regions of that space. As such, existing offline and transfer learning approaches can struggle to adapt to these implicit and unpredictable changes in real-time, rendering configuration performance learning challenging. To address this, we propose DHDA, an online configuration performance learning framework designed to capture and adapt to these drifts at different levels. The key idea is that DHDA adapts to both the local and global drifts using dually hierarchical adaptation: at the upper level, we redivide the data into different divisions, within each of which the local model is retrained, to handle global drifts only when necessary. At the lower level, the local models of the divisions can detect local drifts and adapt themselves asynchronously. To balance responsiveness and efficiency, DHDA combines incremental updates with periodic full retraining to minimize redundant computation when no drifts are detected. Through evaluating eight software systems and against state-of-the-art approaches, we show that DHDA achieves considerably better accuracy and can effectively adapt to drifts with up to 2× improvements, while incurring reasonable overhead and is able to improve different local models in handling concept drift.

CCS Concepts

• **Software and its engineering** → **Software performance**; **Software configuration management and version control systems**; *Software performance.*

*Zezhen Xiang is also supervised in the IDEAS Lab.
†Tao Chen is the corresponding author.

Keywords

configuration performance modeling, online learning, concept drift, performance engineering, configurable system, surrogate model

1 Introduction

Configurations are essential elements that impact the performance of modern software systems, e.g., latency and throughput [15, 53]. As a few examples, for video encoders such as x264, inappropriate configurations might cause 10× performance degradation [40]; configuring the query_cache_type setting in MySQL can result in a performance boost greater than 11 times for specific workloads [63].

Since how configurations can affect performance is complex [53], a crucial task therein is to model the correlation between configurations and performance [8, 34]. Much work has been proposed, including white-box analytical models [35, 59], learning-based approaches [11, 18, 32, 33, 55], and hybrid ones [19]. Recent research in this field has been focusing on learning in multiple environments, such as workloads, hardware, or versions [27, 39]. This makes sense, as real-world software systems are inevitably required to run under diverse conditions [47, 49]. Therefore, it is essential to ensure that configuration performance models trained in one environment remain effective when the system operates in another, previously unseen environment.

However, an unaddressed limitation of existing work is that they mostly assume offline learning/modeling, i.e., models are trained/built under historical environments using a fixed set of sample data and then applied to predict performance for the same or new environment, after which no further updates are considered. This ignores the true nature of running configurable systems: we do not often have a fixed set of configuration data, but those samples are collected continuously in an uncertain and dynamic manner [62], especially when the measurement of configurations is highly expensive [32, 37, 52]. For example, when self-adapting the configurable system at runtime [10, 12], a model, which informs planning decisions, can be continuously updated as measured data is collected for system adaptation under time-varying workloads. Yet, in the other case, profiling the system at deployment time might require changing the version and/or hardware, from which new

data is collected, and the model can be updated to expedite the relevant testing [46]. In those cases, the produced model should also be available anytime. This means that current approaches will struggle to adapt to the continuously changing and evolving characteristics of configuration performance over time, in which the concept to be learned is time-varying—a general phenomenon known as *concept drift* [20, 45]. For example, an empirical study on the performance problems in configurable file systems [38] reveals that:

> "Anomalies like disk-hogs and packet-loss from concept drift disrupt software performance symmetry, underscoring the need for adaptive algorithms."

The above calls for an online configuration performance modeling approach as opposed to the current mainstream "offline efforts" [9]. Yet, learning configuration performance online is challenging due to the interplay between uncertain concept drifts and high sparsity of configuration data, where both the configuration options and samples can have radically different impacts on the performance [47].

While general online learning approaches exist [23, 45, 47], they do not fit well with our case because (1) they often fail to handle the sparsity of configuration data which harms their accuracy [47]. (2) They do not consider the interplay between concept drift and data sparsity, such that the concept drifts might only occur locally in the configuration space [29]. (3) These approaches rely on costly full retraining or ignore the expensive sample measurement of configurations [47].

To fill the above gaps, in this paper, we present Dually Hierarchical Drift Adaptation, dubbed DHDA, for learning configuration performance online. DHDA relies on DaL [26], a recent offline approach that handles sparsity in configuration data by building multiple local models for divided areas of the data, but significantly extends it for online learning. The unique property of DHDA is that it adapts to concept drift in a dually hierarchical way: *global drifts* might be detected at the upper level, which triggers a full-scale adaptation of all local models. The lower level, i.e., at each local model, detects and adapts to *local drifts* asynchronously and locally. In a nutshell, our key contributions include:

- An upper level drift adaptation that monitors the states of all local models, then adapts to global drifts only when it is necessary.
- A lower level drift adaptation, through an extended drift detector, that only detects and adapts to local drifts that are happening in the local area of the configuration data, making drift adaptation more asynchronous and effective.
- To maintain local models under normal conditions, we design a hybrid model maintenance mechanism that relies on both incremental updates and full retraining for consolidating the models in the absence of concept drifts.
- We evaluate DHDA by comparing it with nine baseline/state-of-the-art approaches and a diverse set of local models, together with in-depth ablation analysis.

The results demonstrate that DHDA achieves the best accuracy on 75% cases (6/8) with statistical significance and up to 2× improvements. The model adaptation times are also highly acceptable, in a matter of seconds. All data and source code can be found at our repository: https://github.com/ideas-labo/dhda.

The remainder of this paper is organized as follows. Section 2 presents the problem definitions and characteristics in online software performance learning. Section 3 introduces DHDA , detailing its key components. Section 4 describes the experimental setup, while Section 5 evaluates DHDA comprehensively. Section 6 discusses key findings, whether DHDA is practically useful, and threats to validity. Section 7 reviews related work, and Section 8 concludes the paper.

2 Background

2.1 Configuration Performance Learning via DaL

Without loss of generality, configuration performance learning seeks to build a function f such that:

$$\mathcal{P} = f(\boldsymbol{S}), \mathcal{P} \in \mathbb{R} \tag{1}$$

whereby $\boldsymbol{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_k\}$ is a set of training samples such that $\bar{s}_k = \{\bar{x}_k, p_k\}$. $\bar{x}_k = \{x_{1,k}, x_{2,k}, ..., x_{n,k}\}$ is a configuration and x_i is the value of the ith configuration option for sample k, which might be binary, categorical or enumerative. $\mathcal{P}$ is a performance metric, and the value for sample k is p_k (e.g., latency or throughput), measured under an environment.

Many solutions have been proposed to learn such a f [5, 26, 29, 33, 36]. More recently, one of the successful attempts is DaL [26, 29], a framework that caters for the ***feature*** and ***sample sparsity*** in configuration data. DaL reformulates Equation 1 from a regression problem to a mixed two-level problem such that the upper level is classification while the lower level is regression:

$$\boldsymbol{D} = g(\boldsymbol{S}) \tag{2}$$

$$\forall D_i \in \boldsymbol{D}: \mathcal{P} = f(D_i), \mathcal{P} \in \mathbb{R} \tag{3}$$

where D_i is a division that consists of proportional samples from the training dataset. In a nutshell, DaL works as follows:

(1) **Dividing:** Divide the training samples into diverse divisions $\boldsymbol{D}$ using Classification and Regression Tree (CART) [41] with a depth d (building function g). The samples-division mappings are used to train a Random Forest classifier.

(2) **Training:** Train a dedicated local model for each division D_i, containing samples extracted at the d depth from the trained CART (building function f).

(3) **Predicting:** Leverage the Random Forest classifier to assign a newly coming configuration into the right model for prediction (using functions g and f).

DaL, as a generic framework, has been shown to be superior to many state-of-the-art approaches. The unique property is that DaL can be paired with an arbitrary approach as the local model, hence improving its ability to handle sample sparsity to consolidate accuracy. As such, we leverage DaL as the foundation in this work for online scenarios.

2.2 Online Configuration Performance Learning

To cater for the dynamically evolving environment under uncertainty, i.e., workloads, versions, and hardware, of a configurable system, the configuration performance learning can be best modeled as learning online. Formally, this means

$$\mathcal{P} = f_t(\boldsymbol{S}_t), \langle\{\bar{x}_1, p_1\}, \{\bar{x}_2, p_2\}, ..., \{\bar{x}_m, p_m\}\rangle \rightarrow \boldsymbol{S}_t \tag{4}$$

Similar to the offline scenario in Equation 1, online configuration performance learning also seeks to build the function f. However, since the new configuration samples are collected continuously as a data stream in a dynamic and uncertain manner: at time t, the training sample is enriched in time-varying paces with one or more newly measured samples ($\mathcal{S}_t$) and hence the function f_t needs to be updated. This is the root cause of concept drift [1, 38, 45], where the concept of correlation between a performance metric and configuration can change over time, depending on the time-varying environment. By online, we refer to either system design time or runtime, in which the former often occurs in Continuous Integration/DevOps such that the system is continuously profiled under different environments [36, 49] while the latter means runtime monitoring/sampling of a system (or its digital twin) [10, 13].

It is worth noting that online configuration performance learning differs from the existing multi-tasks/transfer configuration performance learning [27, 39]. This is because, in the latter, the different environments can be modeled explicitly and with certainty, i.e., we know when and how the environment changes. However, in online configuration performance learning, the environment changes implicitly and in an uncertain way, e.g., a system might run under a workload A for some time before experiencing another workload B, after which workload A reoccur again, or the system might require to conduct a live migration to a (virtual) machine with different hardware. Therefore, we do not know when and at what scale those changes occur.

Yet, most existing works and `DaL` have not been designed for such an online learning scenario, which is what we seek to address in this work.

2.3 Problem Characteristics

To understand the challenges and characteristics of online configuration performance learning, we study the datasets collected from commonly used configurable systems and their workloads in prior studies [36, 39, 60]. Figure 1 shows the common examples of systems under continuous environment changes and we observe:

- **Characteristic 1:** It is uncertain when the concept drift occurs due to time-varying environments and sample measurements (Figures 1a and 1b).
- **Characteristic 2:** Because of configuration data sparsity, concept drifts might occur globally (Figures 1b and d) or only in local areas (Figures 1a and c). For example, we see that for system NGINX, the discrepancy in the landscape only occurs in local areas when the version changes. This can also be reflected in the actual data stream, where concept drift might occur in one area of the configuration sample space but not the other.

In addition, the nature of configurable systems and their quality assurance pipeline implies specific requirements on the overhead of online configuration performance learning:

- **Characteristic 3:** Due to costly measurements [37] and the dynamic nature of system performance, the learning needs to be efficient for real-time computation, allowing "anytime" property for model exploitation.

Therefore, simply retraining `DaL` (or other offline learning approaches) in an ad-hoc manner would certainly fail to handle

(a) Local drifts in NGINX   **(b) Global drifts in SQLITE**

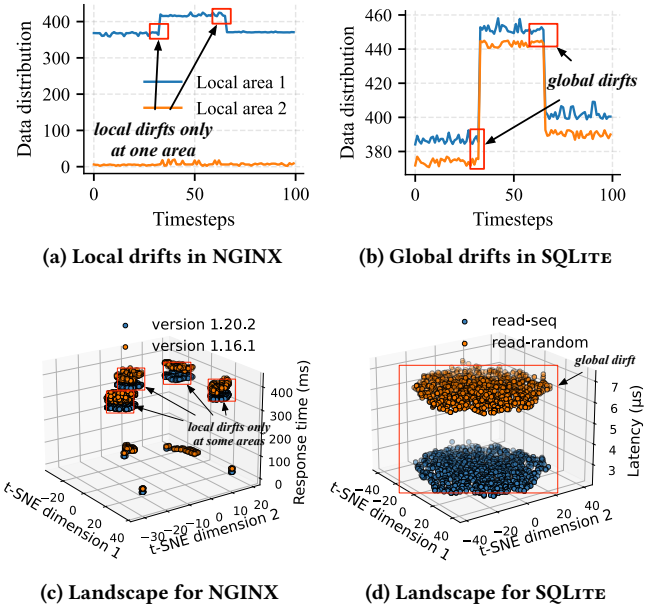**(c) Landscape for NGINX**   **(d) Landscape for SQLITE**

Figure 1: Examples of local and global drifts for systems. (c) and (d) apply t-SNE [58] to project the high-dimensional configuration space onto two representative dimensions.

the above challenges, which, as we will show, can lead to devastating outcomes. In contrast, the general online learning approaches [24, 25] can fail to handle the specific sparsity issue in configuration data, and they are not tailored to the patterns of concept drifts that are most common for online configuration performance learning (**Characteristics 2**).

3 Online Learning with DHDA

Drawing on the characteristics, we propose DHDA for online configuration performance learning leveraging `DaL`. Like all online learning approaches, DHDA maintains a sliding window $\mathcal{W}$ that contains q incrementally measured configuration samples. As such, DHDA changes the problem at timestamp t as:

$$\mathcal{D}_t = g_t(\mathcal{W}) \tag{5}$$

$$\forall D_{i,t} \in \mathcal{D}_t : \mathcal{P} = f_t(D_{i,t}), \mathcal{P} \in \mathbb{R} \tag{6}$$

Here, each local model would also have their own sliding window $\mathcal{W}_i$. The key idea is that although updating the CART (g_t) and the changes of the local models (f_t) based on their divisions of samples $D_{i,t}$ happen at the same timestamps, the actual samples used and the mechanism of model updates could differ among different local models, leading to different extents/aspects of local model change. This dually hierarchical adaptation in handling concept drifts is our key to fit the characteristics observed from Section 2.3.

Figure 2 and Algorithm 1 show the workflow of DHDA with the following key components for learning a continuously measured stream of configuration data online:

- **Upper Level Online Learner:** At lines 12-17, we detect global drift at the upper level, and trigger the full-scale local models adaptation only if needed (**Characteristics 1-2**).
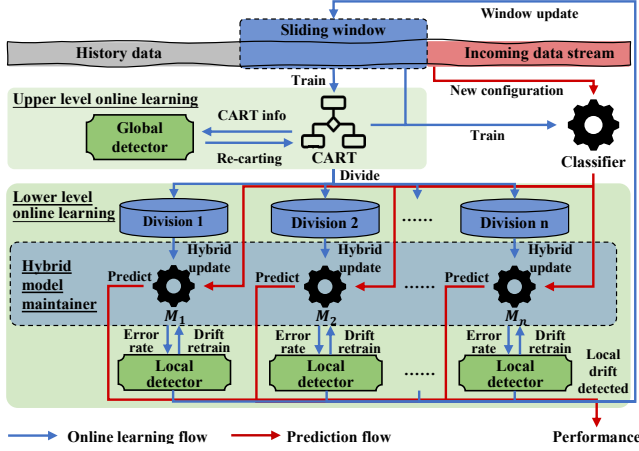
Figure 2: The general overview of DHDA.

Algorithm 1: Pseudo code of online learning in DHDA

Input: The depth d; and number of timesteps of retaining α; Sliding window at upper level $\mathcal{W}$; windows at lower level for each local model $\mathcal{W}_i$; a new batch of samples at timestep t $\mathcal{S}_t$

1 **for** *every timestep t* **do**
2     $\mathcal{W} = \mathcal{W} \cup \mathcal{S}_t$
3     $\mathcal{C}_t \leftarrow$ TRAINCART($\mathcal{W}$)
4     $\mathcal{D} \leftarrow$ extract all divisions from $\mathcal{C}_t$ under d
5     $\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_n \leftarrow$ assign the new samples into the windows of the local model based on $\mathcal{D}$
6     $\mathcal{F} \leftarrow$ TRAINRANDOMFOREST($\mathcal{D}$)
7   /* the initialization at the first timestep only.   */
8   **if** $\mathcal{C}_c = \emptyset$ **then**
9        $\mathcal{C}_c = \mathcal{C}_t$
10   **for** $\forall D_i \in \mathcal{D}$ **do**
11          $\mathcal{M} \leftarrow$ TRAINLOCALMODEL(D_i)

12   /* upper level online learning.   */
13   **if** $G(\mathcal{C}_t) - G(\mathcal{C}_c) > \epsilon$ **then**
14   clear $\mathcal{M}$ and all local windows
15   **for** $\forall D_i \in \mathcal{D}$ **do**
16          $\mathcal{M} \leftarrow$ TRAINLOCALMODEL(D_i)
17       $\mathcal{C}_c = \mathcal{C}_t$
18   **else**
19   **for** $\forall \mathcal{M}_{i,c} \in \mathcal{M}$ **do**
20   /* lower level online learning.   */
21   **if** *ADWIN detects drifts on $\mathcal{M}_{i,c}$ over $\mathcal{W}_i$* **then**
22   discard out-of-date samples in $\mathcal{W}_i$
23             $\mathcal{M}_{i,t} =$ TRAINLOCALMODEL($\mathcal{W}_i$)
24   **else**
25   /* hybrid model maintaining.   */
26             $t' \leftarrow$ count the number of timesteps that have new data samples assigned to $\mathcal{W}_i$ since last drift
27   **if** t' mod $\alpha \neq 0$ **then**
28                $\mathcal{W}'_i \leftarrow$ extract the new data from $\mathcal{W}_i$
29                $\mathcal{M}_{i,t} =$ UPDATELOCALMODEL($\mathcal{M}_{i,c}, \mathcal{W}'_i$)
30   **else**
31                $\mathcal{M}_{i,t} =$ TRAINLOCALMODEL($\mathcal{W}_i$)
32   replace $\mathcal{M}_{i,c}$ with $\mathcal{M}_{i,t}$

- **Lower Level Online Learner:** At the lower level, we detect local drift for the model of each division. Adaptation of

---

Algorithm 2: Pseudo code of prediction in DHDA

Input: A new configuration $\bar{c}$ to be predicted; a random forest $\mathcal{F}$, and the set of local models $\mathcal{M}$
Output: The predicted performance of $\bar{c}$
1 $D_i =$ PREDICT($\mathcal{F}, \bar{c}$)
2 $\mathcal{M} =$ get the model from $\mathcal{M}$ that corresponds to the predicted division D_i
3 **return** PREDICT($\mathcal{M}, \bar{c}$)

---

different models is considered in isolation, as shown in lines 20-23 (**Characteristics 1-2**).

- **Hybrid Model Maintainer:** At lines 25-32, during normal operation, we adopt a hybrid training mechanism that combines incremental updates and re-training to consolidate the local models (**Characteristic 3**).

In this way, global drifts occurring at the upper level would be detected and trigger a full-scale adaptation while local drifts happening in each individual local model can be detected and handled locally, without affecting the entire DaL. Local model adaptation can also benefit from hybrid adaptation.

Upon prediction, DHDA basically follows the procedure of DaL to assign the given configuration into the suitable division and its local model, which then predicts the performance (Algorithm 2).

3.1 Learning Upper Level Online (Global Drifts)

The natural design of CART in DaL is that it determines the produced divisions, which then affect the local models trained. Therefore, any change at the upper level would require sustainable retraining of all the local models at the lower level (**Characteristic 2**).

3.1.1 Drift Detection. To avoid excessive full scale adaptation and meet **Characteristic 3**, as a batch of new samples arrives at time t, we compute the additional information entropy, quantified by Gini importance [4], when using all new data samples arrived in $\mathcal{W}$ so far to retrain the CART (since retraining CART using $\mathcal{W}$ is cheap). The result of this is compared against an extended Hoeffding Bound [57]:

$$\epsilon = \sqrt{\frac{\ln(\frac{1}{\delta})}{2h}} \text{ s.t., } h = \frac{L}{\sum_{i=1}^{L} \frac{1}{n_i}} \tag{7}$$

ϵ is the threshold that caps the independent random variables that deviate from their expected values by more than a certain amount. δ is the significance level ($\delta = 0.05$ in our work). In the original Hoeffding Bound, h denotes the total number of samples, but it is ill-fitted to our case as (1) it has not considered the contributions of data belonging to individual divisions at depth d in DaL and (2) the data/drifts in different divisions can be highly imbalanced, e.g., most newly arriving samples might belong to one division more than the others. Therefore, we set h as the harmonic mean of sample numbers from all divisions, where n_i is the number of samples belongs to the ith division; L is the number of divisions at depth d and $L \in [d+1, 2^d]$ [29]. For example, if $d = 1$, then certainly we only have two divisions.

When all new samples cumulated so far lead to additional information entropy beyond the extended Hoeffding Bound, it means that those new data samples should cause sustainable, often sudden, changes of the divisions, and hence updating all local models is necessary (**Characteristic 1-2**).

Specifically, DHDA adapts a drift detector that works as follows:

(1) Use all samples in $\mathcal{W}$ to retain a new CART $\mathcal{C}_t$.
(2) Calculate the deviation between the total Gini importance [4] G of $\mathcal{C}_t$ and that of the current CART $\mathcal{C}_c$:

$$\Delta G = G(\mathcal{C}_t) - G(\mathcal{C}_c) \tag{8}$$

(3) If $\Delta G > \epsilon$, then we detect a drift at the upper level that needs to be handled.

3.1.2 Drift Mitigation. When a drift at the upper level is detected, DHDA replaces $\mathcal{C}_c$ with $\mathcal{C}_t$ and performing a "redividing" process (updating g_t), producing a new set of divisions and training all local models for the divisions. Here, every local model needs to be completely retrained, as drift detected at the upper level often implies a sudden and global drift. The Random Forest classifier in DaL is retrained with the new data samples regardless of the global drift since the local models need to be adapted under normal situations (see Section 3.3).

3.2 Learning Lower Level Online (Local Drifts)

At the lower level, each division is trained with a local model, hence the localized drifts are detected and handled within the division.

3.2.1 Drift Detection. To detect local drifts, we use the ADWIN [2, 22]—a widely used solution—as the drift detector to monitor the extents of local drifts (**Characteristic 1-2**). However, ADWIN only detects the performance change of a model regardless of whether the change is beneficial. This is problematic since if the error is generally decreasing, then it often means that the model is gradually adapting to the concept rather than a concept drift. As such, we pair a trace filter [31] that compares the traces of two time-series and determines whether they are increasing or decreasing; only the increasing one is of concern.

In a nutshell, at timestep t (if no global drift detected), DHDA uses a retained CART to assign the new samples into the window $\mathcal{W}_i$ for the ith current model $\mathcal{M}_{i,c}$ adapted at timestep c. ADWIN, paired with the trace filter, splits $\mathcal{W}_i$ into two parts that deviate the most in terms of model performance, i.e., the order samples ($\mathcal{W}_{i,l}$) and the newer samples ($\mathcal{W}_{i,r}$), and monitors the performance of $\mathcal{M}_{i,c}$ in two states (with default settings):

- **Warning state:** If the overall degradation of accuracy from $\mathcal{W}_{i,l}$ to $\mathcal{W}_{i,r}$, as identified by the filter, is more than a relaxed threshold (90%), then we flag a warning.
- **Drift state:** If there is already a previously flagged warning state and the overall degradation of accuracy from $\mathcal{W}_{i,l}$ to $\mathcal{W}_{i,r}$, as identified by the filter, is more than a stricter threshold (99%), then we detect a drift.

To measure accuracy, we adopt the widely used, scale-invariant Mean Absolute Percentage Error (MAPE) in DHDA: $\frac{1}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{y_i}$, where y_i and $\hat{y}_i$ are the actual and predicted performance, respectively, for a total of N samples.

ADWIN also helps DHDA to determine which samples in the sliding window $\mathcal{W}_i$ should be discarded when there is a drift: any samples between the latest warning state and the drift are considered as timely data, while any previous data samples in $\mathcal{W}_i$ is the out-of-date data and can be discarded—this is the key to keep the
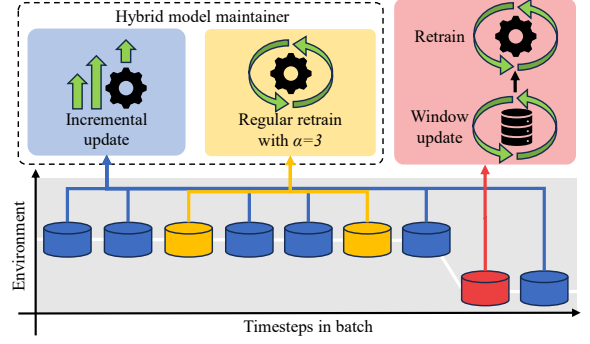

Figure 3: Hybrid model maintenance in DHDA.

model always learning the "right" concepts. The same samples in the global window $\mathcal{W}$ are also removed.

3.2.2 Drift Mitigation. Once a local drift is detected for $\mathcal{M}_{i,c}$, DHDA resets the detector, discards out-of-date data, and adapts $\mathcal{M}_{i,c}$ using the remaining data in $\mathcal{W}_i$. The model adaptation follows the standard retraining process. We would like to stress that the local drift at different local models can be detected and mitigated asynchronously, hence fitting **Characteristics 2**.

3.3 Hybrid Model Maintaining

The above describes the need for model adaptation when concept drifts occur. In addition, we also observe that even in a normal situation, newly arriving data can be helpful in improving the performance of configuration performance prediction, even though the concept remains unchanged. As such, DHDA also maintains each local model under normal operation using new samples in $\mathcal{W}_i$ without discarding any data. However, given **Characteristic 3**, adapting the model by frequent retraining is not ideal. To that end, at each timestep, DHDA applies a hybrid adaptation mechanism that adapts the model using both incremental update and retraining complementarily.

Specifically, for the incremental update, we adapt the current model $\mathcal{M}_{i,k}$ whose weights/structure are denoted by θ, adjusting them with the newly arrived samples $\mathcal{W}'_i$, thus yielding the updated model $\mathcal{M}_{i,t}$ (the UPDATELOCALMODEL function in Algorithm 1):

$$\mathcal{M}_{j,k} + \theta \cdot f(\mathcal{W}'_i) \Rightarrow \mathcal{M}_{i,t} \tag{9}$$

In contrast, for retraining, we use all samples in the local windows ($\mathcal{W}_i$) accumulated so far for building a new model:

$$f(\mathcal{W}_i) \Rightarrow \mathcal{M}_{i,t} \tag{10}$$

As in Figure 3, whenever a batch of new data samples arrives, we adapt the local models. The decision as to whether the model is adapted following incremental update (based on the current model) or completely retrained is controlled by a parameter α, which is the number of timesteps where there are new samples assigned to the window of the local model before the next retraining cycle. It is worth noting that this differs from a full-scale adaptation triggered by global drifts in several aspects:

- Updating the local models under normal conditions can be done incrementally, as there is no concept drift. Full-scale

adaptation, in contrast, requires retraining to handle the global drifts.

- Since there is no drift, there is no need to redivide the data samples.
- When retraining is needed, the local models are retained asynchronously in the absence of concept drift, i.e., not all the models might be adapted at the same timestep depending on how the new data samples are assigned to their windows, since some local models might not have any new data for a timestep. In full-scale adaptation, all models need to be retained at the same time.

In this way, we enable local models to keep track of the new data information (which might not cause drifts) while minimizing the overhead incurred in model adaptation.

4 Experiment Setup

In this section, we describe the setup used to evaluate DHDA in online configuration performance learning scenarios. All experiments are conducted on a server with Ubuntu, Intel(R) Xeon(R) Platinum, 40-core CPUs and 500GB RAM.

4.1 Research Questions

To comprehensively assess DHDA, we seek to answer the following research questions (RQs):

- **RQ1:** How does DHDA compare to state-of-the-art approaches in terms of accuracy and computational cost?
- **RQ2:** How does DHDA impact different local models used for online performance learning?
- **RQ3:** How do the key designs in DHDA contribute?
- **RQ4:** What is the sensitivity of DHDA to its parameter α?

In all cases, unless otherwise stated, we pair DHDA with Random Forest as the default local model due to its high model adaptation efficiency for online learning.

4.2 Subject Configurable Systems

We rely on commonly used datasets from prior studies [36, 37, 39, 60], covering eight widely used configurable software systems across diverse domains, including deep learning frameworks, web servers, cloud computing tools, database management systems, and big data analyzers. They are selected based on their prevalence in state-of-the-art performance learning, the diverse behaviors in performance, and the presence of multiple environments. Table 1 provides an overview of the subject systems.

4.2.1 Noise and Bias Mitigation. The chosen datasets (systems and their environments) have been rigorously curated and assessed, ensuring that the evaluation remains reliable, unbiased, and comparable with prior research [36, 37, 39, 60]. Specifically, the datasets have been collected with the following major strategies:

- To reduce measurement noise, during dataset collection, each configuration was measured three times under the same conditions, and the mean execution time was stored.
- Random or specific sampling was applied in systems with large configuration spaces to ensure coverage of diverse performance behaviors while maintaining feasibility. For

Table 1: Details of the subject systems. ($|\mathcal{B}|/|\mathcal{N}|$) denotes the number of binary/numerical options, $|\mathcal{C}|$ denotes the number of valid configurations per environment (full sample size), and $|\mathcal{E}|$ denotes the total number of environments in the data streams. $|\mathcal{H}|$, $|\mathcal{W}|$, and $|\mathcal{V}|$ respectively denotes the number of hardware, workload, and version considered.

System	Domain	$|\mathcal{B}|/|\mathcal{N}|$	$|\mathcal{C}|$	$|\mathcal{E}|$	Ref.	Environments $|\mathcal{H}|$	$|\mathcal{W}|$	$|\mathcal{V}|$
SaC	Cloud tool	58/0	4999	10	[36]	1	10	1
x264	Video encoder	16/0	2047	21	[36]	6	3	2
Storm	Big data analyzer	1/11	2048	8	[37]	1	4	1
SPEAR	Audio editor	14/0	16384	10	[36]	3	4	2
SQLite	Database	14/0	1000	15	[39]	2	14	2
NGINX	Web server	16/0	1104	4	[60]	1	1	4
ExaStencils	Code generator	8/4	4098	4	[60]	1	4	1
DeepArch	DNN tool	12/0	4096	3	[37]	3	1	1

instance, in SQLite, 1000 configurations were sampled to capture variations in indexing and compression settings.
- Environment diversity was explicitly considered to simulate different real-world scenarios[1]. For example, SQLite was measured on query types covering sequential reads, random reads, sequential writes, and batch writes, which can change unexpectedly.

Indeed, the scale of the dataset might not be as large as some other software engineering problems [42, 43]. The key reason is the expensive data collection for configurable systems: it has been reported that even for a single configuration, it can take up to 166 minutes to complete one measurement run [7, 14], let alone the fact that each sample in the datasets is averaged from 5 repeats/measurements. As such, the datasets with up to 16, 000 samples per environment (e.g., up to 14 diverse workloads, 2 versions, and 2 hardware) are already promising.

For the dimensions of options, we follow what has been collected/identified in the datasets [36, 37, 39, 60] and those options that are insensitive to performance have been removed. For example, a recent study [44] shows that NGINX has 79 options, but only 15% are performance sensitive, which can be easily filtered out by reading the documentation/code. Therefore, we hope to avoid creating an artificially complex problem with dimensionality more than necessary.

4.3 Experiment Procedure with Data Streams

To emulate real-world concept drifts, we randomly mix the samples from different environments of a system (for each run) to form a data stream (e.g., system with 5000 configurations and 12 environments would have $12 \times 5000 = 60000$ samples), since real-life cases are chaotic/uncertain: we do not know when a workload will change nor when we migrate the hardware. The steps are:

(1) We randomly select the batches of samples (with 32 samples per batch) from the mixed workloads and sequentially feed them into a model for prediction[2]. The data stream is built without replacement, i.e., the sample that has been fed will

---

[1]The detailed environments for each system can be found at:
https://github.com/ideas-labo/dhda/tree/main/drift_data.
[2]To ensure the realism of the experiment under our resources, we constrain the data samples from each environment to be 4000.

not be used again. In total, for a data stream with 3 environments, we have $\frac{3\times4000}{32} = 375$ timesteps, each with 32 newly arrived samples.

(2) By the end of timestep t, the model might be updated using data up to t (all or partial); the updated model will predict all samples for the newly arrived batch at timestep $t + 1$ and compute the accuracy/MAPE for 32 new samples.

(3) The process is repeated for the next timestep—a standard procedure for online learning/concept drifts [23].

As such, the evaluation always focuses on the generalization error: predictions on data not used for model updating/training.

4.4 Metrics and Statistical Validation

To evaluate the prediction accuracy for learning configuration online, we rely on the widely adopted MAPE [18, 26, 27, 33]. While MAPE has limitations, such as the asymmetry in penalizing overestimations and underestimations, it remains the best choice for our evaluation because it provides an intuitive measure of scale-invariant error, enabling meaningful comparisons across the systems with varying performance magnitudes [8, 28].

However, since we need to measure the accuracy of predicting different batches of samples across the data stream, we extend the MAPE (denoted as mMAPE) as:

$$\text{mMAPE} = \frac{1}{T} \sum_{t=1}^{T} MAPE_t \tag{11}$$

whereby $MAPE_t$ is the MAPE measured for the tth batch for a total of T timesteps/batches.

To ensure statistical robustness, each experiment is repeated 30 times with different random seeds. To compare the performance of multiple approaches for **RQ1** and **RQ2**, we use the Scott-Knott Effect Size Difference (Scott-Knott ESD) test [54], which is commonly used in software performance studies [8, 26–29] because it considers both statistical significance and practical effect sizes, avoiding misleading conclusions from minor but statistically significant differences. Scott-Knott ESD iteratively partitions approaches into distinct groups in a hierarchical manner based on effect sizes. For example, Scott-Knott ESD test might rank three approaches as two groups: $\{A, B\}$ and $\{C\}$, such that the former is ranked 1 while the latter has 2, then we say that A and B are statistically similar but they are significantly better than C.

For pairwise comparisons in the ablation study of **RQ3**, we utilize the non-parametric Wilcoxon Signed-Rank test [51] to assess whether the performance differences between DHDA and its ablated variants are statistically significant. We use a confidence level of 0.05, i.e., the comparison is statistically significant if $p < 0.05$.

5 Evaluation

5.1 RQ1: DHDA against the State-of-the-art

5.1.1 Method. To assess the effectiveness of DHDA in online software performance learning, we compare it against several state-of-the-art approaches[3] that adopt adaptive learning, online learning, meta-learning, and transfer learning:

- **Adaptive Random Forest (ARF)** [24]: A general adaptive learning method designed for online regression tasks using an ensemble of randomized trees with adaptive mechanisms to handle concept drift.
- **Streaming Random Patches (SRP)** [25]: A general online learning approach that dynamically creates and updates ensemble models using subsets of features and data patches.
- **Sequential Meta-Performance Learning (SeMPL)** [27]: A meta-learning approach for configuration performance learning across multiple environments in an offline setting. We adapt it by periodically retraining the model every timestep to accommodate streaming performance data.
- **Bellwether Transfer Learner (BEETLE)** [39]: A transfer learning method for configuration performance learning. Again, we extend it by allowing it to retrain every timestep.

together with the baselines below that follow offline, periodic, or aggressive full retraining:

- **Divide-and-Learn (DaL)** [26]: The baseline approach that DHDA extended from (Section 2.1). We compare DHDA with the naive use of DaL for online configuration performance learning that is completely retrained every timestep.
- **DaL$_{fixed}$**: A DaL that only trains on the first 50 samples and never update.
- **DaL$_\alpha$**: A DaL that is retrained at every α timesteps as DHDA.
- **RF$_r$**: A Random Forest retrained at every timestep.
- **RF$_\alpha$**: A Random Forest that is retrained at every α timesteps as DHDA.

The compared approaches all use their default settings. For DaL, we set $d = 1$ and for DHDA, we set $d = 1$ and $\alpha = 3$, because $d = 1$ is generally recommended [29] and $\alpha = 3$ achieves the best trade-off between quality and time (see Section 5.4). We also use $\alpha = 3$ for DaL$_\alpha$ and RF$_\alpha$. All approaches are run under the same data stream, and we report the median and interquartile range (IQR) of the mMAPE over 30 runs, together with the ranks from the Scott-Knott ESD test. We also assess the clock time required to model adaptation per timestep.

5.1.2 Results. As can be seen from Table 2, DHDA achieves the best median mMAPE across 6/8 subject systems, considerably improving the others. The largest improvement is observed in SPEAR, where DHDA achieves a median mMAPE of 0.29, whereas the second best model, SPR, causes 0.87—a 2× accuracy improvement. In terms of ranking, DHDA achieves the best rank in 7/8 systems—an average Scott-Knott rank of 1.25, which is substantially better than the best opponents RF$_r$ (2.375) and DaL (2.5). To understand the stability of the results across the timesteps, Figure 4 demonstrates the traces for two example systems. We see that DHDA constantly maintains a better and more stable accuracy than the others under various levels of drift.

For the remaining two systems, DHDA is also ranked the first in SQLite, even though the median mMAPE is slightly worse than SPR. The only system where DHDA does not achieve the best median mMAPE nor is ranked the best is NGINX. This is attributed to NGINX's relatively small magnitude of concept drifts, where periodic retraining for approaches like DaL is suitable.

---

[3]Note that it has been shown that simply consider environmental features in the model is devastating [27].

Table 2: The median and interquartile range of mMAPE, denoted as Med (IQR), for DHDA and the state-of-the-art approaches for all the subject systems over 30 runs. For each case, blue cells mean DHDA has the best median mMAPE; or red cells otherwise. The one(s) with the best rank (_r_) from the Scott-Knott ESD test is in bold.

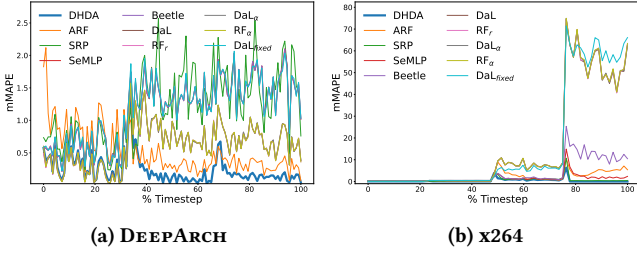| System	DHDA		ARF		SPR		SeMPL		BEETLE		DaL		DaL_fixed		DaL_α		RF_r		RF_α	
	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)	_r_	Med (IQR)
SaC	**1**	**4.36e+0 (7.74)**	3	14.11e+0 (11.57)	4	16.92e+0 (15.52)	6	52.3e+0 (78.13)	6	52.3e+0 (78.23)	4	11.02e+0 (15.61)	5	37.94e+0 (64.77)	4	11.50e+0 (16.12)	2	5.50e+0 (7.33)	4	11.66e+0 (15.89)
x264	**1**	**0.32e+0 (1.05)**	2	1.62e+0 (3.09)	**1**	**0.91e+0 (0.89)**	3	1.66e+0 (5.01)	3	2.5e+0 (4.62)	4	6.39e+0 (18.97)	4	4.60e+0 (14.90)	4	6.44e+0 (19.13)	4	6.39e+0 (18.97)	4	6.45e+0 (19.14)
Storm	**1**	**27.99e+0 (60.13)**	3	246.04e+0 (131.90)	4	459.93e+0 (408.32)	4	195.13e+0 (1157.71)	4	223.09e+0 (1155.11)	2	109.34e+0 (123.62)	2	64.96e+0 (243.48)	2	109.31e+0 (125.00)	2	111.06e+0 (123.47)	2	110.69e+0 (125.73)
SPEAR	**1**	**0.29e+0 (0.13)**	2	1.48e+0 (0.56)	4	0.87e+0 (3.61)	2	1.08e+0 (1.00)	2	1.2e+0 (1.08)	2	1.24e+0 (1.17)	3	1.27e+0 (1.86)	2	1.25e+0 (1.18)	2	1.23e+0 (1.17)	2	1.24e+0 (1.18)
SQLite	**1**	**0.35e+0 (0.33)**	3	1.43e+0 (3.72)	2	0.31e+0 (0.58)	4	1.77e+0 (5.81)	4	3.36e+0 (6.05)	4	1.88e+0 (8.20)	5	1.05e+0 (9.61)	4	1.90e+0 (8.28)	4	1.88e+0 (8.20)	4	1.91e+0 (8.28)
NGINX	3	0.11e+0 (0.03)	6	0.9e+0 (0.31)	7	1.87e+0 (0.17)	4	0.17e+0 (0.19)	4	0.17e+0 (0.19)	**1**	**0.09e+0 (0.06)**	5	0.39e+0 (0.59)	2	0.10e+0 (0.06)	**1**	**0.09e+0 (0.06)**	2	0.10e+0 (0.06)
ExaStencils	**1**	**2.64e-2 (0.00)**	5	0.12e+0 (0.01)	7	0.15e+0 (2.06)	4	0.1e+0 (0.01)	4	0.1e+0 (0.01)	2	2.66e-2 (0.00)	6	0.15e+0 (0.02)	3	2.69e-2 (0.00)	2	2.65e-2 (0.00)	3	2.69e-2 (0.00)
DeepArch	**1**	**0.24e+0 (0.02)**	3	0.54e+0 (0.06)	5	1.01e+0 (0.30)	4	0.55e+0 (0.74)	4	0.55e+0 (0.74)	2	0.37e+0 (0.32)	5	0.78e+0 (0.61)	2	0.38e+0 (0.32)	2	0.37e+0 (0.32)	2	0.38e+0 (0.33)
Average _r_	**1.25**		3.375		4		3.875		3.875		2.5		4.625		2.875		2.375		2.875	

(a) DeepArch

(b) x264

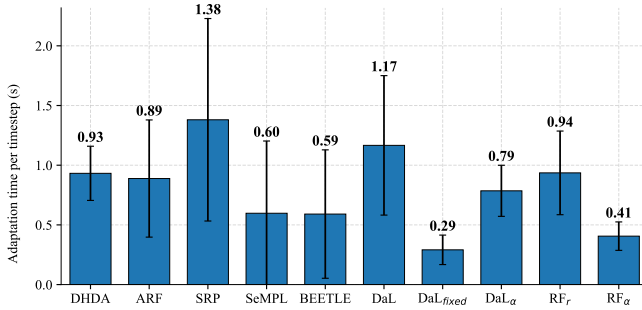**Figure 4: Exampled accuracy traces for DeepArch and x264.**

Figure 5: The mean model adaptation time and standard deviation per timestep over all systems and runs.

When compared with the baselines, DHDA achieves better ranks in seven systems, while DaL and RF_r are ranked the best in NGINX. DHDA's superior performance stems from two key factors: (1) the underlying DaL framework handles sparse configuration data better than RF [26, 29], and (2) DHDA adapts hierarchically with proper forgetting strategies, unlike aggressive retraining where certain old data might become noise (DaL and RF_r) or periodic retraining that suffers from accumulated drifts between updates ($DaL_α$ and $RF_α$).

Figure 5 shows the model adaptation time per timestep. Notably, DHDA achieves an average of 0.93 seconds, which is highly competitive compared with the others, while maintaining the generally best accuracy across almost all systems. Overall, while SeMPL, BEETLE, DaL_{fixed}, and $RF_α$ may be considered if lower time costs are preferred at the expense of accuracy, DHDA remains the most balanced approach for high-accuracy, real-time requirements. Thus, we say

> **RQ1:** DHDA *achieves statistically the best accuracy in six out of eight cases with up to* 2× *accuracy gain, while incurring competitive model adaptation time.*

5.2 RQ2: DHDA under Different Local Models

5.2.1 Method. Since DHDA can be paired with different local models, we evaluate its effectiveness when combined with different independent local models, including tree-based (RF [4], HT [57]), distance-based (kNN [50]), and linear model (LR [30]). We compare those variants of DHDA with the standalone local models, each of which is incrementally updated at every timestep by refining the existing model's sub-trees or weights with the newly arrived batch of data (after the MAPE for the new batch has been assessed). The evaluation follows **RQ1** with 30 runs.

5.2.2 Results. The results in Table 3 demonstrate that DHDA consistently improves accuracy across all local model types. For instance, when paired with RF, DHDA achieves an average rank of 1.25, significantly outperforming the standalone RF for 1.2×, which has an average rank of 2.75. Notably, the best local model for DHDA is RF, which is also our default. In NGINX, DHDA_RF outperforms the second-best model RF for 2.7×.

In terms of subject systems, DHDA_RF achieves the best Scott-Knott ESD rank among seven out of eight systems. This is particularly evident in systems such as Storm, SPEAR, NGINX, ExaStencils, and DeepArch, where it consistently ranks as the sole best one across all other models. However, in some cases, the improvements are less pronounced. For example, in SQLite, DHDA with RF does not outperform DHDA_HT, HT, and kNN. These cases could be due to the stability of performance patterns in certain systems, where frequent adaptation provides limited additional benefits. As a result, we have

> **RQ2:** DHDA *is model-agnostic and consistently advances various local models with the ability to handle concept drifts, leading to better accuracy. When paired with* RF*, it achieves the best rank in seven out of eight systems, with up to 2.7× improvements.*

5.3 RQ3: Effectiveness of the Key Components

5.3.1 Method. To assess the robustness of DHDA, we conduct an ablation study by disabling key components and measuring their impact on accuracy. Specifically, we evaluate four ablated versions of DHDA, each with one key component removed:

Table 3: The median and IQR of mMAPE of DHDA under different local models and their counterparts paired with DHDA over 30 runs. The blue cells denote the best median mMAPE. The one(s) with the best rank (r) from the Scott-Knott ESD test is in bold.

System	DHDA$_{RF}$		RF		DHDA$_{HT}$		HT		DHDA$_{kNN}$		kNN		DHDA$_{LR}$		LR	
	r	Med (IQR)	r	Med (IQR)	r	Med (IQR)	r	Med (IQR)	r	Med (IQR)	r	Med (IQR)	r	Med (IQR)	r	Med (IQR)
SaC	1	**4.36e+0 (7.74)**	2	8.95e+0 (16.49)	3	9.72e+0 (16.37)	3	11.38e+0 (15.65)	2	5.84e+0 (8.99)	2	5.14e+0 (11.35)	1	**4.72e+0 (7.48)**	3	10.68e+0 (18.64)
x264	1	**0.32e+0 (1.05)**	1	**0.37e+0 (0.88)**	1	**30.60e-2 (1.03)**	1	**0.33e+0 (0.91)**	1	**30.99e-2 (1.08)**	1	**0.46e+0 (0.89)**	1	**0.70e+0 (0.26)**	1	**0.71e+0 (0.69)**
Storm	1	**27.99e+0 (60.13)**	4	131.67e+0 (255.81)	3	83.65e+0 (94.93)	5	615.09e+0 (433.86)	2	45.26e+0 (53.63)	3	73.66e+0 (96.82)	3	105.05e+0 (92.63)	6	800.30e+0 (338.78)
SPEAR	1	**0.29e+0 (0.13)**	4	1.13e+0 (1.02)	2	0.64e+0 (0.46)	6	5.85e+0 (3.56)	5	4.18e+0 (0.88)	3	0.80e+0 (0.27)			7	15.20e+0 (5.02)
SQLite	3	0.35e+0 (0.33)	2	0.28e+0 (0.42)	1	**0.27e+0 (0.21)**	1	**0.31e+0 (0.27)**	2	0.30e+0 (0.37)	1	**0.30e+0 (0.29)**	4	0.61e+0 (0.13)	4	0.47e+0 (0.26)
NGINX	1	**0.11e+0 (0.03)**	4	0.41e+0 (0.94)	3	0.59e+0 (0.22)	7	2.61e+0 (1.66)	2	0.42e+0 (0.1)	5	1.18e+0 (0.23)	6	1.66e+0 (0.09)	8	5.37e+0 (0.30)
ExaStencils	1	**0.03e+0 (0.00)**	2	0.05e+0 (0.02)	4	0.09e+0 (0.01)	6	0.11e+0 (0.01)	3	0.08e+0 (0.00)	5	0.09e+0 (0.00)	8	0.62e+0 (0.01)	7	0.34e+0 (0.00)
DeepArch	1	**0.24e+0 (0.02)**	3	0.56e+0 (0.22)	3	0.53e+0 (0.07)	2	0.37e+0 (0.10)	4	0.6e+0 (0.10)	5	0.85e+0 (0.10)	6	0.99e+0 (0.03)	7	1.57e+0 (0.25)
Average r	1.25		2.75		2.5		3.875		2.25		3.375		4		5.375	

Table 4: The median and IQR of mMAPE in the ablation study of DHDA over 30 runs. For each case, blue cells mean DHDA has the best median mMAPE; or red cells otherwise. The $p < 0.05$ cases are in bold.

System	DHDA	DHDA$_{NU}$	DHDA	DHDA$_{NL}$	DHDA	DHDA$_{NH}$
SaC	4.36e+0 (7.74)	4.30e+0 (4.65)	4.36e+0 (7.74)	3.83e+0 (6.29)	**4.36e+0 (7.74)**	6.40e+0 (15.29)
x264	**0.32e+0 (1.05)**	**0.37e+0 (1.10)**	**0.32e+0 (1.05)**	**1.10e+0 (2.00)**	**0.32e+0 (1.05)**	0.55e+0 (1.16)
Storm	27.99e+0 (60.13)	29.98e+0 (62.87)	27.99e+0 (60.13)	44.61e+0 (45.40)	**27.99e+0 (60.13)**	**99.91e+0 (244.14)**
SPEAR	**0.29e+0 (0.13)**	**0.28e+0 (0.15)**	**0.29e+0 (0.13)**	**0.57e+0 (0.29)**	**0.29e+0 (0.13)**	**1.65e+0 (2.39)**
SQLite	0.35e+0 (0.33)	0.36e+0 (0.32)	**0.35e+0 (0.33)**	**0.78e+0 (1.73)**	0.35e+0 (0.33)	0.37e+0 (0.66)
NGINX	0.11e+0 (0.03)	0.12e+0 (0.06)	**0.11e+0 (0.03)**	**0.07e+0 (0.02)**	**0.11e+0 (0.03)**	0.83e+0 (1.33)
ExaStencils	**2.64e-2 (0.00)**	2.65e-2 (0.00)	**0.03e+0 (0.00)**	**0.05e+0 (0.00)**	**0.03e+0 (0.00)**	0.14e+0 (0.05)
DeepArch	0.24e+0 (0.02)	0.24e+0 (0.02)	**0.24e+0 (0.02)**	**0.36e+0 (0.03)**	**0.24e+0 (0.02)**	**0.95e+0 (0.25)**

- **DHDA$_{NU}$ (No upper level online learning):** Remove upper-level global drift detection and adaptation, restricting DHDA to only react to local drifts.
- **DHDA$_{NL}$ (No lower level online learning):** Disable lower-level drift detection, forcing the model to rely solely on global drift detection/adaptation.
- **DHDA$_{NH}$ (No hybrid maintaining):** The models are not updated unless there is a concept drift.

Each ablation variant is evaluated using the same setup as in **RQ1** over 30 runs. Yet, since we are interested in pair-wise comparisons and the data stream sequence used in the updating is unchanged, we use Wilcoxon Signed-Rank test to detect less biased statistical significance.

5.3.2 Results. Results in Table 4 highlight the necessity of each key component in DHDA. In most of the systems, DHDA consistently achieves the better median mMAPE with statistical significance ($p < 0.05$), confirming that removing any component degrades performance.

DHDA$_{NU}$ (without upper level learning) performs worse than DHDA in median mMAPE for six out of eight systems, four of which show statistical significance. For the only cases where DHDA$_{NU}$ performs better (SPEAR), the reason is that the sudden global drifts rarely occur, and hence local drift detection/adaptation is sufficient.

Subsequently, DHDA$_{NL}$ (without lower level learning) exhibits degradation in mMAPE for six systems, while five of them show $p < 0.05$. In particular, in x264, removing local drift adaptation severely impacts prediction accuracy, causing the median mMAPE to worsen from 0.32 in DHDA to 1.10 in DHDA$_{NL}$—a degradation by a factor of 2.44. These results indicate that without local adaptation, DHDA$_{NL}$ fails to respond rapidly to frequent small-scale shifts, degrading overall prediction accuracy. Yet, for NGINX, DHDA is worse because although NGINX mostly suffers local drifts, the magnitude

is however relatively small, hence the lower level drift adaptation in DHDA does not bring many benefits but incurs disruption.

Finally, DHDA$_{NH}$ (without hybrid maintaining) suffers severe accuracy loss in all eight systems ($p < 0.05$ in seven). These results emphasize that hybrid model maintenance plays a crucial role in keeping local models stable even in cases when no concept drift is detected. Overall, we conclude that

> **RQ3:** *The key components in DHDA can significantly improve the accuracy in general, or at least do not harm the performance, especially when a certain level of concept drifts occur frequently.*

5.4 Sensitivity to the Parameter α

5.4.1 Method. The parameter α in DHDA determines how frequently local models undergo full retraining in the hybrid model maintenance process. To understand the sensitivity of DHDA to α, we test DHDA under different α settings, ranging from rather infrequent retraining ($\alpha = 7$) to full retraining ($\alpha = 1$). We report on both the mMAPE and the clock time required to model adaptation per timestep. Other settings are the same as **RQ1**.

5.4.2 Results. Figure 6 presents the effect of the parameter α on both mMAPE and clock time for model adaptation. From Figure 6a, we observe that the mean mMAPE remains relatively stable when $1 \leq \alpha \leq 3$, even though a bigger α implies less frequent retraining, hence the correlation between old and new data samples is learned less. In contrast, the accuracy starts to drop when $\alpha \geq 4$, suggesting that this is the point where only relying on incremental update becomes harmful. On the other hand, Figure 6b reveals a clear decreasing trend in model adaptation time as α increases, which is as expected, and we see that when $\alpha = 3$ the time reduction becomes much slower. This demonstrates that reducing retraining
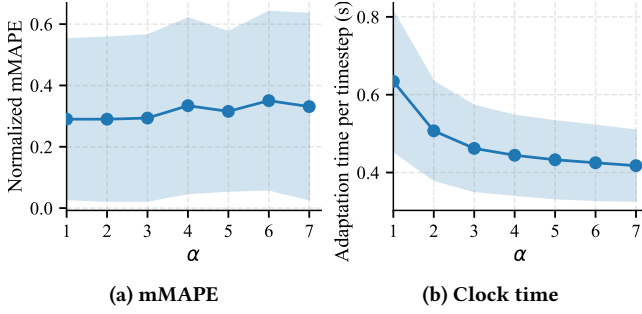
Figure 6: The mean mMAPE/model adaptation time and its standard deviation showing the sensitivity of DHDA to the parameter α over all systems and runs.
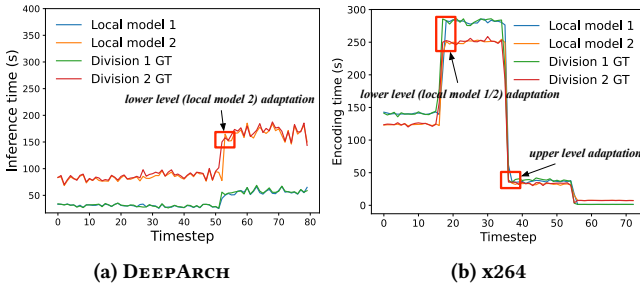

Figure 7: Examples of performance prediction traces of all divisions in DHDA. GT denotes ground truth.

frequency significantly improves computational efficiency, lowering the overall adaptation cost. As a result, we set $\alpha = 3$ as the default, serving as a reasonable trade-off between accuracy and overhead. Therefore, we say that

> **RQ4:** The accuracy of DHDA is sensitive to α when it is big enough; in contrast, α is negatively and monotonically correlated with model adaptation time. Yet, setting $\alpha = 3$ leads to a reasonable trade-off.

6 Discussion

6.1 Why does DHDA Work?

To understand why DHDA work, Figure 7 shows the common performance prediction traces within each division of DHDA for two systems. We can see that, for Figure 7a, DHDA is able to adapt to local drift that occurred only within one of the divisions, while the other model is unchanged and can still maintain good predictions. In contrast, for Figure 7b, DHDA adapts at the lower level on both local models initially, but later on the concept drifts become much more globally harmful, DHDA then adapts at the upper level, redividing the samples and retraining all local models again. These examples evidence the superiority of DHDA which is consistent with our theory presented in Section 2.

Table 5: The median accuracy improvements of DHDA over the second best approach that can lead to statistically significant benefits in practical use of the resulted model. We consider only the systems that DHDA is ranked the best. Full data can be accessed here: https://github.com/ideas-labo/model-impact/blob/main/RQ_supplementary/RQ5/supplement.pdf.

System	Second Best	Δ vs. DHDA	The Least Δ [8]	Useful?
SaC	RF_r	1.14e+0	0.68e+0	✔
x264	SPR	0.59e+0	0.35e+0	✔
Storm	DaL_{fixed}	36.97e+0	0.68e+0	✔
SPEAR	SPR	0.58e+0	0.38e+0	✔
ExaStencils	RF_r	0.01e-2	0.07e-1	✗
DeepArch	DaL	0.13e+0	0.13e+0	✔

6.2 Is DHDA Practically Useful?

Being a model, DHDA cannot make decisions, but it serves as an important foundation for such when paired with heuristic/deterministic algorithms [16]. For example, in configuration tuning with Bayesian optimization, where a single configuration measurement might take up to hours [15, 17], DHDA can serve as the surrogate model that cheaply evaluates the configurations' performance [6]. It also allows rapid updates/adaptations to new data via learning online. For self-adaptation, a model, which can be updated as the system runs, can help assess system performance at runtime without affecting the system, e.g., model-assisted genetic-algorithm planner [10, 13, 61]. Yet another example is that DHDA can also help with prioritization of configuration performance testing [46], e.g., we can use DHDA to evaluate the configuration, rank them, and choose the one with worse performance to measure/test first, which increases the likelihood of finding performance bugs earlier.

A related challenging question to answer for the above is whether the accuracy improvements observed from DHDA is practically beneficial [8]. Indeed, the applications of the approach can be vast, and it is almost impossible to foresee how the resulting model can be used. Yet, to provide insights on this aspect, we refer to the data collected by Chen et al. [8] from a large-scale empirical study, which provides "hints" on how much accuracy improvement tends to be practically useful.

Chen et al. [8] summarize the average extents of accuracy (MAPE) improvement of an approach A over the other B can lead to statistically significant benefits when being used to guide a model-based configuration tuning. For example, if B has an MAPE of 0.15 (15% $\in [20\%, 30\%]$), then A needs to reduce that by at least the threshold of 0.093 for one to observe statistically significant improvements from the tuning when using A instead of B.

Drawing on the above, we showcase the systems where DHDA performs the best and compare the accuracy improvement to the second best approach. As we can see from Table 5, clearly, the accuracy improvement made by DHDA over the second best approach has exceeded the Δ thresholds on five out of six cases, meaning that, in general, the improved accuracy from DHDA can be useful and beneficial for practical model-based configuration tuning.

6.3 Assumptions and Limitations

A key assumption of DHDA is that the data from different environments have uncertain/varying correlations, i.e., data from one environment cannot always be useless (suitable for incremental learning), nor consistently useful (suitable for simple retraining), for the others. There would be some uncertain dynamics involved, hence the dynamic adaptation in DHDA is meaningful. Prior studies have revealed those for configuration data [36, 49].

Several limitations should be acknowledged. (1) DHDA is less effective for abrupt changes, e.g., rapid migration between completely different hardware, as the drift detection/adaptation needs time to reflect; (2) DHDA cannot cold start, i.e., it requires some accumulated data for the drift detector to work properly; (3) Although we found $\alpha = 3$ is generally reasonable, it might not be universally optimal.

6.4 Threats to Validity

Internal Threats. Internal threats to validity are related to the parameter settings used in DHDA and its baseline comparisons. To ensure fairness, we strictly follow their original parameter settings and ensure their effectiveness in the online setting. We also conducted a sensitivity analysis on the parameter α across multiple subject systems and found that $\alpha = 3$ is a reasonable trade-off between accuracy and time. Yet, we agree that those parameter settings might not be optimal for all scenarios.

Construct Threats. Threats to construct validity may lie in the choice of evaluation metrics. We use MAPE as the primary metric due to its wide adoption in software performance learning [27, 39]. However, MAPE is sensitive to small actual values and penalizes over- and under-estimations asymmetrically. To mitigate this, we conduct statistical validation using the Scott-Knott ESD test and Wilcoxon Signed-Rank test for 30 runs.

External Threats. External validity could be raised from the subject systems used. While we use eight commonly studied configurable systems from diverse domains [37, 39, 60]. We have also assessed DHDA paired with different local models. Indeed, evaluating more systems might further consolidate our results.

7 Related Work

Single-environment Offline Configuration Performance Model. Machine learning approaches have been widely applied to predict configuration performance in a single static environment. SPLConqueror [55] applies a performance-influence model based on linear regression to capture the impact of configuration options. DECART [32] extends CART with a novel sampling technique to improve performance learning efficiency. Deep learning-based methods have also been explored, such as DeepPerf [33], which leverages regularized deep neural networks to learn the sparse interactions between options, and HINNPerf [18], which employs hierarchical interaction neural networks to capture the hierarchical dependencies between configurations and performance. DaL [26] further improves the accuracy level by handling both feature and data sparsity.

However, these approaches assume an overly optimistic static environment and do not consider the evolving performance landscapes in dynamic software systems.

Multi-environment Offline Configuration Performance Model. To extend performance prediction across multiple environments, meta-learning and transfer learning techniques have been introduced. For example, transfer learning methods such as BEETLE [39] and the other similar approach tEAMS [48] aim to transfer knowledge from one environment to another, improving model adaptability. SeMPL [27] further employs historical knowledge from multiple environments in sequential meta-learning to better learn configuration performance.

While these methods enhance generalization across environments, they rely on offline training, requiring documented data from known environments before making predictions, which limits their applicability in real-time evolving settings.

General Online Learning Models. Online learning models exist for handling concept drifts in different forms [3, 21, 24, 25, 56]. Among others, ARF [24] uses dedicated detectors to monitor data distribution changes and trigger model updates when drift is identified. Examples include Adaptive Random Forest, which integrates drift detectors into an ensemble framework. SRP [25] is a framework that dynamically updates model components using randomly selected features and data subsets, and Hoeffding Adaptive Tree (HAT) [3] is another similar model that applies the Hoeffding bound to detect significant changes in the feature distribution.

However, unlike DHDA, those approaches cannot handle the unique sparsity and the inherited hierarchical concept drifts presented in configuration data. Further, they have ignored the expensive measurements of configuration data; hence, they do not work well in small datasets like configuration data.

8 Conclusion

This paper presents DHDA, a framework for online configuration performance learning, handling the unique global and local drifts presented in configuration data. The key novelty of DHDA is that, relying on the two level structure from DaL, it performs dually hierarchical drift adaptation: at the upper level, the configuration data can be re-divided and all local model is retained only when needed to handle global drifts; at the lower level, the local model of each division can detect and adapt to local drifts. By evaluating DHDA on eight systems and against five baseline/state-of-the-art, we demonstrate that DHDA:

- considerably improves the accuracy against concept drifts in 6 out of 8 systems with up to 2× improvements;
- can enhance the concept drift adaptation ability of various local models;
- and leads to competitive model adaptation time in the magnitude of seconds.

Future work along this line of research can be fruitful, including dynamically reusing old local models and more advanced estimation of concept drift types.

Acknowledgment

References

[1] Supriya Agrahari and Anil Kumar Singh. 2022. Concept Drift Detection in Data Stream Mining : A literature review. *J. King Saud Univ. Comput. Inf. Sci.* 34, 10 Part B (2022), 9523–9540. doi:10.1016/J.JKSUCI.2021.11.006

[2] Albert Bifet and Ricard Gavaldà. 2007. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA.* SIAM, 443–448. doi:10.1137/1.9781611972771.42

[3] Albert Bifet and Ricard Gavaldà. 2009. Adaptive Learning from Evolving Data Streams. In *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5772),* Niall M. Adams, Céline Robardet, Arno Siebes, and Jean-François Boulicaut (Eds.). Springer, 249–260. doi:10.1007/978-3-642-03915-7_22

[4] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32. doi:10.1023/A:1010933404324

[5] 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. doi:10.1109/ICSE43902.2021.00110

[6] Pengzhou Chen and Tao Chen. 2026. PromiseTune: Unveiling Causally Promising and Explainable Configuration Tuning. In *48th IEEE/ACM International Conference on Software Engineering (ICSE).* ACM.

[7] Pengzhou Chen, Tao Chen, and Miqing Li. 2024. MMO: Meta Multi-Objectivization for Software Configuration Tuning. *IEEE Trans. Software Eng.* 50, 6 (2024), 1478–1504. doi:10.1109/TSE.2024.3388910

[8] Pengzhou Chen, Jingzhi Gong, and Tao Chen. 2025. Accuracy can lie: On the impact of surrogate model in configuration tuning. *IEEE Transactions on Software Engineering* (2025).

[9] 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. doi:10.1109/SEAMS.2019.00029

[10] Tao Chen. 2022. Lifelong Dynamic Optimization for Self-Adaptive Systems: Fact or Fiction?. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022.* IEEE, 78–89. doi:10.1109/SANER53432.2022.00022

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

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

[13] 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.

[14] 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.* 453–465. doi:10.1145/3468264.3468555

[15] Tao Chen and Miqing Li. 2023. 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. doi:10.1145/3571853

[16] Tao Chen and Miqing Li. 2023. The Weights Can Be Harmful: Pareto Search versus Weighted Search in Multi-objective Search-based Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 32, 1 (2023), 5:1–5:40. doi:10.1145/3514233

[17] Tao Chen and Miqing Li. 2024. Adapting Multi-objectivized Software Configuration Tuning. *Proc. ACM Softw. Eng.* 1, FSE (2024), 539–561. doi:10.1145/3643751

[18] Jiezhu Cheng, Cuiyun Gao, and Zibin Zheng. 2023. HINNPerf: Hierarchical Interaction Neural Network for Performance Prediction of Configurable Systems. *ACM Trans. Softw. Eng. Methodol.* 32, 2 (2023), 46:1–46:30. doi:10.1145/3528100

[19] Diego Didona and Paolo Romano. 2014. On Bootstrapping Machine Learning Performance Predictors via Analytical Models. *CoRR* abs/1410.5102 (2014). arXiv:1410.5102 http://arxiv.org/abs/1410.5102

[20] Anton Dries and Ulrich Rückert. 2009. Adaptive concept drift detection. *Stat. Anal. Data Min.* 2, 5-6 (2009), 311–327. doi:10.1002/SAM.10054

[21] Ryan Elwell and Robi Polikar. 2011. Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Trans. Neural Networks* 22, 10 (2011), 1517–1531. doi:10.1109/TNN.2011.2160459

[22] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17.* Springer, 286–295.

[23] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.* 46, 4 (2014), 44:1–44:37. doi:10.1145/2523813

[24] Heitor Murilo Gomes, Jean Paul Barddal, Luis Eduardo Boiko Ferreira, and Albert Bifet. 2018. Adaptive random forests for data stream regression. In *26th European Symposium on Artificial Neural Networks, ESANN 2018, Bruges, Belgium, April 25-27, 2018.* https://www.esann.org/sites/default/files/proceedings/legacy/es2018-183.pdf

[25] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. 2019. Streaming Random Patches for Evolving Data Stream Classification. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019,* Jianyong Wang, Kyuseok Shim, and Xindong Wu (Eds.). IEEE, 240–249. doi:10.1109/ICDM.2019.00034

[26] 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. doi:10.1145/3611643.3616334

[27] Jingzhi Gong and Tao Chen. 2024. Predicting Configuration Performance in Multiple Environments with Sequential Meta-Learning. *Proc. ACM Softw. Eng.* 1, FSE (2024), 359–382. doi:10.1145/3643743

[28] Jingzhi Gong and Tao Chen. 2025. Deep Configuration Performance Learning: A Systematic Survey and Taxonomy. *ACM Trans. Softw. Eng. Methodol.* 34, 1 (2025), 25:1–25:62. doi:10.1145/3702986

[29] Jingzhi Gong, Tao Chen, and Rami Bahsoon. 2025. Dividable Configuration Performance Learning. *IEEE Trans. Software Eng.* 51, 1 (2025), 106–134. doi:10.1109/TSE.2024.3491945

[30] J. Brian Gray. 2002. Introduction to Linear Regression Analysis. *Technometrics* 44, 2 (2002), 191–192. doi:10.1198/TECH.2002.S714

[31] Nuwan Gunasekara, Heitor Murilo Gomes, Albert Bifet, and Bernhard Pfahringer. 2022. Adaptive Neural Networks for Online Domain Incremental Continual Learning. In *Discovery Science - 25th International Conference, DS 2022, Montpellier, France, October 10-12, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13601),* Pascal Poncelet and Dino Ienco (Eds.). Springer, 89–103. doi:10.1007/978-3-031-18840-4_7

[32] 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).

[33] 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, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019,* Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1095–1106. doi:10.1109/ICSE.2019.00113

[34] 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. doi:10.1145/2961111.2962602

[35] Xue Han, Tingting Yu, and Michael Pradel. 2021. ConfProf: White-Box Performance Profiling of Configuration Options. In *ICPE '21: ACM/SPEC International Conference on Performance Engineering, Virtual Event, France, April 19-21, 2021,* Johann Bourcier, Zhen Ming (Jack) Jiang, Cor-Paul Bezemer, Vittorio Cortellessa, Daniele Di Pompeo, and Ana Lucia Varbanescu (Eds.). ACM, 1–8. doi:10.1145/3427921.3450255

[36] 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, 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. doi:10.1109/ASE.2017.8115661

[37] 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. doi:10.1145/3236024.3236074

[38] Michael P. Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. 2010. Black-Box Problem Diagnosis in Parallel File Systems. In *8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 23-26, 2010,* Randal C. Burns and Kimberly Keeton (Eds.). USENIX, 43–56. http://www.usenix.org/events/fast10/tech/full_papers/kasick.pdf

[39] 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. doi:10.1109/TSE.2020.2983927

[40] Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2023. Input sensitivity on the performance of configurable systems an empirical study. *J. Syst. Softw.* 201 (2023), 111671. doi:10.1016/J.JSS.2023.111671

[41] Roger J Lewis et al. 2000. An introduction to classification and regression tree (CART) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, Vol. 14. Department of Emergency Medicine Harbor-UCLA Medical Center Torrance San ....

[42] Ke Li, Zilin Xiang, Tao Chen, and Kay Chen Tan. 2020. BiLO-CPDP: Bi-Level Programming for Automated Model Discovery in Cross-Project Defect Prediction. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 573–584. doi:10.1145/3324884.3416617

[43] Ke Li, Zilin Xiang, Tao Chen, Shuo Wang, and Kay Chen Tan. 2020. Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: an empirical study. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 566–577. doi:10.1145/3377811.3380360

[44] Hongyuan Liang, Yue Huang, and Tao Chen. 2025. The Same Only Different: On Information Modality for Configuration Performance Analysis. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2522–2534. doi:10.1109/ICSE55347.2025.00212

[45] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *IEEE Trans. Knowl. Data Eng.* 31, 12 (2019), 2346–2363. doi:10.1109/TKDE.2018.2876857

[46] Youpeng Ma, Tao Chen, and Ke Li. 2025. Faster Configuration Performance Bug Testing with Neural Dual-Level Prioritization. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 988–1000. doi:10.1109/ICSE55347.2025.00201

[47] Sandeep Madireddy, Prasanna Balaprakash, Philip H. Carns, Robert Latham, Glenn K. Lockwood, Robert B. Ross, Shane Snyder, and Stefan M. Wild. 2019. Adaptive Learning for Concept Drift in Application Performance Modeling. In *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, Kyoto, Japan, August 05-08, 2019*. ACM, 79:1–79:11. doi:10.1145/3337821.3337922

[48] 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. doi:10.1109/TSE.2021.3116768

[49] Stefan Mühlbauer, Florian Sattler, Christian Kaltenecker, Johannes Dorn, Sven Apel, and Norbert Siegmund. 2023. Analysing the Impact of Workloads on Modeling the Performance of Configurable Software Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2085–2097.

[50] Leif E Peterson. 2009. K-nearest neighbor. *Scholarpedia* 4, 2 (2009), 1883.

[51] Denise Rey and Markus Neuhäuser. 2011. Wilcoxon-Signed-Rank Test. In *International Encyclopedia of Statistical Science*, Miodrag Lovric (Ed.). Springer, 1658–1659. doi:10.1007/978-3-642-04898-2_616

[52] Atrisha Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 342–352. doi:10.1109/ASE.2015.45

[53] 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. doi:10.1109/TSE.2018.2867847

[54] Andrew Jhon Scott and Martin Knott. 1974. A cluster analysis method for grouping means in the analysis of variance. *Biometrics* (1974), 507–512.

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

[56] Symone G. Soares and Rui Araújo. 2015. A dynamic and on-line ensemble regression for changing environments. *Expert Syst. Appl.* 42, 6 (2015), 2935–2948. doi:10.1016/J.ESWA.2014.11.053

[57] Yingying Tao. 2011. *Mining Time-Changing Data Streams*. Ph. D. Dissertation. University of Waterloo, Ontario, Canada. https://hdl.handle.net/10012/6374

[58] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[59] Max Weber, Sven Apel, and Norbert Siegmund. 2021. White-Box Performance-Influence Models: A Profiling and Learning Approach. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1059–1071. doi:10.1109/ICSE43902.2021.00099

[60] 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. doi:10.1109/ICSE48619.2023.00177

[61] Yulong Ye, Tao Chen, and Miqing Li. 2025. Distilled Lifelong Self-Adaptation for Configurable Systems. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 1333–1345. doi:10.1109/ICSE55347.2025.00094

[62] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. 2015. Performance Prediction of Configurable Software Systems by Fourier Learning (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 365–373. doi:10.1109/ASE.2015.15

[63] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. BestConfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*. ACM, 338–350. doi:10.1145/3127479.3128605