Deep Configuration Performance Learning: A Systematic Survey and Taxonomy

JINGZHI GONG[*], University of Electronic Science and Technology of China, China; Loughborough University, UK
TAO CHEN[†], University of Birmingham, UK

Performance is arguably the most crucial attribute that reflects the quality of a configurable software system. However, given the increasing scale and complexity of modern software, modeling and predicting how various configurations can impact performance becomes one of the major challenges in software maintenance. As such, performance is often modeled without having a thorough knowledge of the software system, but relying mainly on data, which fits precisely with the purpose of deep learning. In this paper, we conduct a comprehensive review exclusively on the topic of deep learning for performance learning of configurable software, covering 1,206 searched papers spanning six indexing services, based on which 99 primary papers were extracted and analyzed. Our results outline key statistics, taxonomy, strengths, weaknesses, and optimal usage scenarios for techniques related to the preparation of configuration data, the construction of deep learning performance models, the evaluation of these models, and their utilization in various software configuration-related tasks. We also identify the good practices and potentially problematic phenomena from the studies surveyed, together with a comprehensive summary of actionable suggestions and insights into future opportunities within the field. To promote open science, all the raw results of this survey can be accessed at our repository: https://github.com/ideas-labo/DCPL-SLR.

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

Additional Key Words and Phrases: Configuration Performance, Deep Learning, Configurable Software, Performance Modeling, Performance Prediction, Software Engineering

ACM Reference Format:
Jingzhi Gong and Tao Chen. 2024. Deep Configuration Performance Learning: A Systematic Survey and Taxonomy. *ACM Trans. Softw. Eng. Methodol.* 37, 4, Article 111 (September 2024), 62 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Configuration is pervasive in software systems, ranging from enterprise web applications to robotic software. The intention thereof is simple: by allowing software systems to be flexibly configured in different ways, configuration enables them to have much better applicability over a wide range of domains and a better ability to cope with varying performance requirements, such as latency, throughput, and energy consumption [23, 38, 77, 101, 102]. For example, a popular web server Tomcat can adjust different configuration options (e.g., `maxThreads`), the values of which are

[*]This research was conducted when Jingzhi Going visited the University of Electronic Science and Technology of China.
[†]Corresponding author: Tao Chen, t.chen@bham.ac.uk.

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

ACM 1049-331X/2024/9-ART111
https://doi.org/10.1145/nnnnnnn.nnnnnnn

likely to profoundly influence its throughput. Nevertheless, excessive configurability comes with a cost. Han and Yu [66] have discovered that over 59% of the performance bugs nowadays are due to inappropriate configurations. On the other extreme, software engineers find it generally difficult to adjust the configuration options in order to adapt the performance, therefore most of the options are often ignored, leaving the potential for performance boost untapped [21, 32, 33]. The key cause behind the aforementioned issues is the difficulty of knowing how configurations impact the performance beforehand, since software systems are complex in nature. But what if there is a model that can establish the correlation between configuration and performance?

Configuration performance modeling—a highly active research field over the last decade—has emerged as a crucial topic within the software performance research landscape. The goal therein is exactly to build a model that takes a configuration as input and predicts the likely performance before we deploy that configuration. This holds immense potential for advancements across various domains. For example, in software performance testing, one can easily identify the key configuration options that would likely cause a performance bottleneck by investigating a performance model. Similarly, in configuration tuning, it is straightforward to simply evaluate and compare different configurations on the model, instead of having to deploy and run the configuration on the actual systems, leading to a dramatic reduction in cost.

However, building an accurate performance model for the configuration of software systems is challenging. Classic performance models have been relying on analytical methods [36, 37, 42, 60, 94], but soon they become ineffective due primarily to the soaring complexity of modern software systems. In particular, there are two key reasons which prevent the success of analytical methods: (1) analytical models often work on a limited set of configurations options [26, 42], but the number of configurations options and the complexity continues to increase. For example, Hadoop has only 17 configurations options in 2006, but it was increased by 9× more to 173 at 2013 [178]; similarly, MySQL has 461 configuration options at 2014, in which around 50% of them are of complex types. (2) Their effectiveness is highly dependent on assumptions about the internal structure and the environment of the software being modeled. However, many modern scenarios, such as cloud-based systems, and virtualized and multi-tenant software, intentionally hide such information to promote ease of use, which further reduces the reliability of the analytical methods [23].

As an alternative, machine learning-based configuration performance models have been explored over the past decade, e.g., liner regression [150], decision tree [62], and random forest [56], which work on arbitrary types of configuration options and do not rely on heavy human intervention [137, 170]. Unlike analytical methods, machine learning is data-driven since it seeks to learn the patterns from the configuration data, hence generalizing the correlation between configuration and performance. This problem, namely configuration performance learning, has been gaining momentum in recent years [1, 38, 57, 64, 149].

Among others, a particular type of data-driven configuration performance learning relies on deep learning, i.e., those that make use of deep neural networks [176, 180]. Indeed, recent studies have demonstrated the benefits of deep learning for modeling configuration performance. For example, Ha and Zhang [64] propose `DeepPerf`, a deep neural network model combined with L_1 regularization to address the sparse performance functions, and Cheng et al. [38] invent a hierarchical interaction neural network model called `HINNPerf` that achieves state-of-the-art accuracy. In Section 2.2, we will further elaborate on the importance of deep learning for configuration performance and motivate this survey in detail.

However, despite the importance of such research direction, to the best of our knowledge, there has been little work on a systematic survey that covers the full spectrum of deep configuration performance learning. The current reviews related to this topic mainly focus on either general machine learning models [137], or deep learning in the general context of software engineering [176,

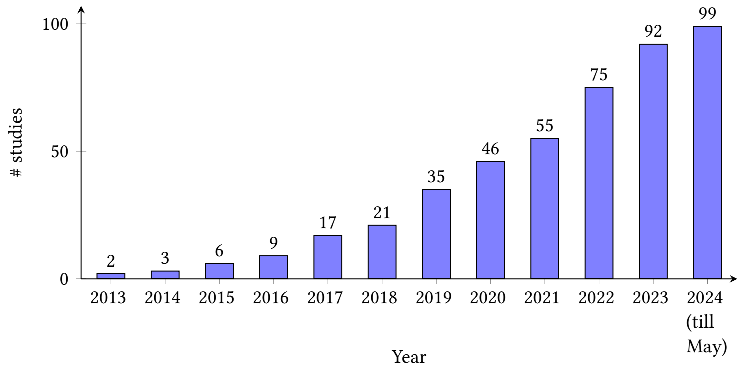


Fig. 1. Cumulative number of primary studies on deep configuration performance learning models.

180]. Undoubtedly, systematically reviewing state-of-the-art studies on this particular research field can provide vast benefits, including summarizing the common categories, revisiting the important concepts, and more importantly, discussing novel perspectives on the good practices and "bad smells[1]" of the field, and providing insights for future opportunities.

To bridge such a gap, in this paper, we conduct a systematic literature review that covers 1,206 papers from six online repositories and 78 venues, published between 2013 and 2024, based on which we extract 99 prominent studies for data extraction and analysis. The results also confirm that the significance and challenges of deep learning for configuration performance have led to a notable increase in research efforts within this field. Indeed, an overwhelming 79% of the reviewed publications have emerged since 2019, as shown in Figure 1.

In a nutshell, the major contributions of this survey include:

- An exhaustive automatic search on six indexing services using a rigorous search string, together with a "quasi-gold standard" validation that demonstrates the high sensitivity and sufficient precision of our search strategy.
- An extensive quality assessment of all the primary studies with 18 questions and scoring metrics, which results in a "golden set" of studies in the field, serving as a good starting point for new researchers.
- A taxonomy that categorizes the techniques used and key concerns in deep configuration performance learning with up to 13 findings of the trends and 18 actionable suggestions that can be leveraged by future researchers.
- Comprehensive summaries of the key approaches used in the deep learning pipeline for configuration performance, including preparation, modeling, evaluation, and application, together with discussions on their benefits, shortcomings, and best-suited scenarios.
- Articulation on the good practices and bad smells observed from the findings.
- Gaps identified from existing studies, offering insights into the future opportunities for this particular thread of research.

In particular, the review results highlight the following key observations that derive several actionable suggestions:

[1]In software engineering, the bad smell is a metaphor that denotes the symptoms of code/software that can lead to a deeper problem. In our context, it refers to problematic practices that could lead to serious threats to validity and/or to the sustainability of the research field.

- Random sampling, which is inefficient in finding the most informative configurations, is the most commonly used method, as used in 79 studies. This indicates the need to explore more informative sampling methods to enhance the quality of the collected data.
- A non-trivial number of studies (34 out of 99) fail to address the issues of sparsity and overfitting in configuration data, while 24 of the rest rely on inefficient manual feature selection methods. This reveals the lack of addressing sparsity issues and encourages researchers to explore alternative approaches.
- The majority of studies (63 out of 99) apply manual hyperparameter tuning, which relies heavily on human experts. Therefore, an actionable suggestion is that researchers should investigate automatic and heuristic hyperparameter tuning techniques to reduce tuning costs and enhance efficiency.
- Nearly half of the primary studies (43 out of 99) do not consider the challenge of dynamic environments. It is crucial for researchers to address this aspect to strengthen the generalizability of configuration performance models across different environments as it has been reported that they can profoundly impact the performance [58, 92, 152].

Furthermore, we highlight five specific directions for future research that are promising to produce fruitful outcomes, namely:

- Model-based algorithms to enhance configuration sampling.
- Explainable deep learning techniques for more reliable configuration performance modeling.
- Modeling configuration performance with deep few-shot learning.
- Interactive approaches for deep configuration performance prediction.
- Configuration performance learning under multiple and dynamic environments.

The rest of the paper is organized as follows. Section 2 presents the background information about deep learning for performance modeling and discusses related work. Section 3 presents our review methodology. Section 4, 5, 6, and 7 outlines the results of the research questions, the taxonomy and summaries of the strength, weakness, usage scenarios and actionable suggestions, along with the good practices and bad smells discovered. Sections 8 and 9 discuss future opportunities of the field, and threats to validity, respectively. Finally, Section 10 concludes the paper.

2 BACKGROUND AND RELATED WORK

In this section, we provide a brief overview of the configurable software systems and the deep learning pipeline. We also differentiate this work and other relevant surveys.

2.1 Configurations in Software Systems

To meet the performance requirements, configurable software systems often permit possible configuration options to be adjusted at design time or at runtime. For example, around one-third of the configuration options of MySQL, a popular configurable database system, can be changed at runtime, such as `max_connections`; the remaining ones, e.g., `autocommit`, need to be fixed a priori to the deployment[2]. Table 1 presents a dataset for VP8, which includes m configurations, each associated with a measured performance metric (runtime).

It has been widely acknowledged that the configuration options have great impacts on software performance [31, 35, 66, 130]. Indeed, inappropriate configurations often cause serious performance bugs, which is the key reason why the users became frustrated enough to (threaten to) switch to another product [188]. At the same time, simply using default configurations does little help, for instance, Herodotou et al. [68] show that the default settings on Hadoop can actually result in the

[2]https://dev.mysql.com/doc/refman/8.0/en/server-system-variable-reference.html

Table 1. An example of configurations and performance for VP8. x_i is the ith configuration option and y is the performance value (runtime).

x_1	x_2	x_3	$\cdots$	x_{n-2}	x_{n-1}	x_n	y
0	0	0	$\cdots$	0	0	1	8190.6 seconds
0	1	0	$\cdots$	0	0	2	6502.4 seconds
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
1	1	1	$\cdots$	1	0	3	29102.2 seconds
1	1	1	$\cdots$	1	1	4	25827.4 seconds

worst possible performance. In this regard, a fundamental question to ask is: *given a configuration, what is the performance of the software systems?*.

Indeed, one solution is to directly profile the software system for all possible configurations when needed. This, however, is impractical, because (1) the number of possible configurations may be too high. For example, MySQL has more than a million possible configurations. (2) Even when such a number is small, the profiling of a single set of configurations can be too expensive. Wang et al. [171] report that it could take weeks of running time to benchmark and profile even a simple system. All these issues have called for a computational model, which is cheap to evaluate yet accurate, that captures the correlation between configuration options to a performance attribute.

2.2 Motivation: Why Deep Learning for Configuration Performance?

In contrast to traditional analytical models, data-driven configuration performance modeling, such as machine learning, operates in a black-box manner by leveraging observations of a limited set of the software's actual performance behaviors and constructing a statistical model that can predict performance without extensive domain knowledge on configuration options and system characteristics that are prevalent in analytical methods [6, 31, 130, 159, 185].

As a result, numerous data-driven machine learning models have been utilized for configuration performance learning, showcasing the efficacy of this approach. For example, linear regression models like `SPLConqueror` have been utilized in different studies [84, 150, 151, 157], which combines linear regression with different sampling methods and step-wise feature selection to capture the interactions between configuration options. Tree structure models have also been employed in a number of research works [62, 71, 130, 148], e.g., `DECART` [62] improves upon `CART` by incorporating an efficient sampling method. In the work of Jamshidi et al. [74], Gaussian Process (GP) is employed to model configuration performance, which is updated incrementally through Bayesian Optimization. Moreover, Fourier-learning models have been applied in investigations by two studies [65, 194] to predict configuration performance by learning the Fourier coefficients of the performance function, and transfer learning techniques have been explored in several works [75, 76, 92] to reuse configuration data in difference environments on a desired target environment.

However, as software systems evolve and become more sophisticated, the number of configuration options and their interactions grow exponentially. This poses a significant obstacle for traditional machine learning models, which may struggle to capture the intricate relationships and interactions within the vast configuration space. For example, Siegmund et al. [151] illustrate how the complexity of configuration spaces can hinder the performance of linear regression models, necessitating the exploration of alternative approaches.

To address these challenges, researchers have been exploring novel learning paradigms that can handle the complexity and scale of modern software systems. Therein, one promising approach is deep learning, which employs deep neural networks. Deep learning models have shown remarkable success in various domains, including natural language processing, image recognition, and also software performance engineering tasks.

Notably, deep learning models share some commonalities with traditional machine learning models for configuration performance learning, i.e.:

- **Data-driven nature:** They both rely on historical data to learn patterns and relationships, which enable them to predict new configurations [56].
- **Common obstacles:** Mitigating overfitting is a common difficulty due to limited training samples and sparse configuration data [57]. Besides, maintaining robustness in unseen environments is also a joint challenge [58].
- **Similar learning pipeline:** They follow common learning procedures, including sampling, data preprocessing, encoding, hyperparameter tuning, model training, evaluation, and application.

Despite the above similarities, the specific structure of neural networks in deep learning equips it with some unique characteristics. For example, in hyperparameter tuning, deeper layers in deep learning mean that the dimension of the hyperparameters can be significantly higher than their machine learning counterparts [163]. It is also known that, while the sampling strategy is important for both types, deep learning tends to be less sensitive to the sampled data [64]. As a result, those differences render deep learning with remarkable advancements, offering numerous advantages over traditional machine learning approaches for configuration performance learning, including:

- **Ease representation handling:** Deep learning is capable of extracting the underlying representations from the configuration options even without careful feature engineering, which allows them to learn directly from raw data, enabling end-to-end learning.
- **Good at handling complex data:** Deep learning models consist of multiple layers of interconnected neurons, enabling them to capture the nonlinear and complex relationships between the configuration options and performance.
- **Better generalizability:** Deep learning can leverage pre-trained knowledge from related tasks and quickly adapt to the target task through fine-tuning, hence they often have better generalizability.
- **More robust to noisy data:** Their ability to learn complex representations helps them generalize well, even in the presence of noisy or incomplete data.
- **More consistent and flexible architectures:** Deep learning offers a wide range of neural network architectures, such as fully connected neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs). Those architectures are from the same origin but differ in their interconnections and can be tailored to specific problem domains and configuration data.

Indeed, recent state-of-the-art studies have demonstrated the effectiveness of deep learning models in configuration performance learning, which surpasses the accuracy of traditional machine learning approaches in their empirical experiments [38, 57, 58, 64]. For example, in 2019, Ha and Zhang [64] proposed `DeepPerf`, a deep neural network model combined with L_1 regularization to address the sparse performance functions, and their evaluation results demonstrate that `DeepPerf` is indeed more accurate than other machine learning models such as `DECART` and `SPLConqueror`. Next, in 2023, Cheng et al. [38] invented a hierarchical interaction neural network model called `HINNPerf` that achieves state-of-the-art accuracy when compared with `DeepPerf`, `DECART`, `SPLConqueror`, and `RF`. In the same year, Gong and Chen [57] proposed the idea of divide-and-learn (`DaL`), which mitigates the sparsity in configuration data and further enhances the accuracy of `DeepPerf` and other peer machine learning ones. Most recently, Gong and Chen [58] show that training a meta-deep neural network model in an optimal sequence under different environments leads to the best result.

Additionally, there has been an increasing trend from the community in exploiting deep learning for configuration performance learning. Recall from Figure 1, we see a significant growth in the number of research papers on deep configuration performance learning during the past decade, further demonstrating the increasing popularity of deep learning in this field. Noteworthily, over the past decade, 78 out of the 99 elected primary studies have emerged since 2019, indicating the rapid emergence of deep learning as a prominent paradigm for configuration performance learning. This trend also emphasizes the widespread adoption and recognition of deep learning's potential in configuration performance modeling.

Yet, despite the importance and potential of such research direction, to the best of our knowledge, there has been little work that focuses explicitly on a systematic survey for deep configuration performance learning, as we will discuss in Section 2.4. This observation serves as the primary motivation behind this survey.

2.3 Problem Formulation of Deep Configuration Performance Learning

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

$$y = f(\overline{\mathbf{x}}) \tag{1}$$

whereby y is the performance attribute that is of concern; $\overline{\mathbf{x}}$ is a configuration consists of the values for n configuration options, i.e., $\overline{\mathbf{x}} = \{x_1, x_2, ..., x_n\}$. Taking the simplest fully connected deep neural network as an example, f is represented as multiple layers of interconnected neurons, where neurons are activated as:

$$a_j^{l+1} = \sigma(\sum_i a_i^l w_{ij}^{l,l+1} + b_j^{l+1}) \tag{2}$$

where $\sum$ runs over all the lower layer neurons that are connected to neuron j. i is the activation of a neuron i in the previous layer, and where $a_i^l w_{ij}^{l,l+1}$ is the contribution of neuron i at layer l to the activation of the neuron j at layer $l+1$. The function σ is a nonlinear monotonously increasing activation function, e.g., a sigmoid function; $w_{ij}^{l,l+1}$ is the weight and b_j^{l+1} is the bias term.

To build function f, the training in deep learning aims to find the set of weights for different neurons from all the layers such that a loss function can be minimized. For example, the mean squared error below is one possible loss function since configuration performance learning is essentially a regression problem:

$$\mathcal{L}(\theta) = \frac{1}{m}\sum_{i=1}^{m}(f(\overline{\mathbf{x}}_i) - y_i')^2 \tag{3}$$

$f(\overline{\mathbf{x}}_i)$ and y_i' denote the predicted and actual performance values for the ith sample across m data points in the training dataset.

2.4 Related Surveys

The related surveys of this work mainly lie in two big categories, i.e., surveys on analytical/machine learning techniques for software configuration performance learning, or reviews on deep learning approaches for different tasks in software engineering.

2.4.1 Surveys on Configuration Performance Modeling. There are a few related surveys in the domain of configuration performance modeling. Among others, Balsamo et al. [11] conduct a review of configuration performance prediction using Queuing Network, and Balsamo et al. [10]

review how performance model can be used to help software development. Nambiar et al. [131] emphasize the significance of configuration performance modeling in SE tasks and calls for further research to enhance the predictive capabilities of performance models, and Pereira et al. [136] seek to understand to what extent are sampling strategies sensitive to configuration performance prediction of configurable systems. A recent work [67] classifies software configuration performance learning studies into 6 categories, provides a mapping of them within these categories, and highlights potential weaknesses of the literature. Further, surveys on configuration performance modeling for specific categories of configurable software systems exist, for example, there are reviews on the performance models for distributed systems [29, 113, 139] and Flores-Contreras et al. [50] survey the configuration performance prediction methods for parallel applications published between 2005 and 2020. Similarly, Frank et al. [51] review 34 studies on configuration performance modeling for multi-core systems, and configuration performance learning and tuning techniques for mobile application systems are reviewed by Hort et al. [69]. In recent years, Stradowski and Madeyski [156] emphasize the importance of machine learning techniques for defect prediction in real-life business scenarios by conducting a systematic mapping study on 32 primary studies, and Zain et al. [187] synthesize existing independent variables, modeling techniques, and performance evaluation criteria used in machine/deep learning software defect prediction research. However, the above work does not focus on how deep learning can be used in the performance model building for configurable software in general.

2.4.2 Surveys on Deep Learning for Software Engineering. On the other hand, several surveys have been conducted on the application of deep learning in software engineering. For example, Yang et al. [180] investigates the deep learning-related techniques for software engineering tasks, including the deep learning models, data preprocessing methods, SE tasks, and model optimization methods, while a review on deep learning is also conducted by Watson et al. [176], which provides a research roadmap and guidelines for future exploration in the context of software engineering. Liu et al. [111] investigates the reproducibility and applicability of deep learning studies in SE, and observes that a significant number of them have overlooked this challenge. A novel work [170] explores the use of machine/deep learning techniques in various software engineering tasks and the challenges and differences between machine and deep learning. Yet, these studies do not focus on a specific SE task like configuration performance learning but rather provide a broad review of software engineering with a restricted depth on each topic.

The most similar prior study to our work is probably a review by Pereira et al. [137] that explores the application objectives, sampling, learning and measuring methods, and evaluation related to machine learning models for configuration performance modeling. However, we focus on deep learning models, which is a specific type of machine learning, and cover a wider range of techniques such as preprocessing, hyperparameter tuning, and sparsity handling methods. Besides, they did not discuss and justify the good practices and bad smells in the field, as well as the actionable suggestions and summaries of the strengths, weaknesses, and best-suited usage scenarios.

2.4.3 Differences of Our Survey. In summary, our work differs from the above in the following aspects:

- We focus explicitly on deep learning-based configuration performance modeling for configurable software in general and capture the latest trends in the past 10 years.
- We review aspects that have not been summarized before, e.g., the use of data preprocessing, encoding scheme, sparsity handling, hyper-parameter tuning, statistical test and effect size test, runtime environments, and public artifact.

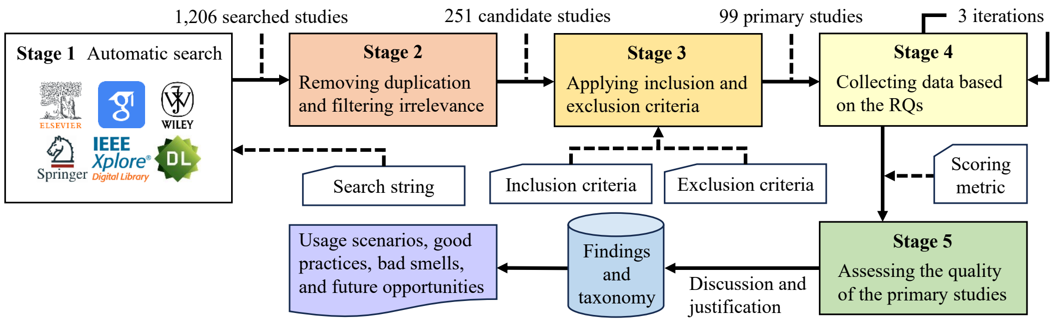

1,206 searched studies
251 candidate studies
99 primary studies
3 iterations
Stage 1 Automatic search
ELSEVIER
WILEY
Springer
IEEE Xplore Digital Library
DL
Stage 2
Removing duplication and filtering irrelevance
Stage 3
Applying inclusion and exclusion criteria
Stage 4
Collecting data based on the RQs
Search string
Inclusion criteria
Exclusion criteria
Scoring metric
Usage scenarios, good practices, bad smells, and future opportunities
Findings and taxonomy
Discussion and justification
Stage 5
Assessing the quality of the primary studies

Fig. 2. Overview of the literature review protocol.

- We present an innovative summary outlining the advantages, disadvantages, and recommended scenarios for each category of deep configuration performance learning techniques.
- We disclose and justify the positive and potentially problematic practices, which have not been revealed previously, and offer a list of actionable suggestions and knowledge gaps to shed light on future works in the field.

Additionally, we would like to kindly stress that the nature of this study is a systematic literature review specializing in deep learning techniques for software configuration performance learning, instead of a mapping study. While both systematic literature review and mapping studies aim to synthesize the current research literature, they differ in terms of purpose and methodology:

- Firstly, a systematic literature review is conducted by addressing specific research questions through a rigorous and comprehensive process. This involves conducting thorough searches, applying strict inclusion and exclusion criteria, performing quality assessment of primary studies, and conducting detailed data analysis, and the outcome is typically detailed findings and in-depth discussions based on the review results [89]. In contrast, a mapping study, or scoping review, typically includes a broader search strategy without strict inclusion and exclusion criteria, producing a mapping of the studies to different concepts, types of studies, and research trends [67]. In our survey, the concepts themselves have already been unified in the field and we conducted the study on different aspects of the known concepts, following specific criteria and protocols as discussed in Section 3.
- Secondly, while systematic literature reviews focus on detailed answers and high-quality analysis, mapping studies aim to offer a broader view of the research landscape, without the same depth of analysis or stringent quality assessment. We provided an in-depth analysis of different aspects of deep configuration performance learning, along with discussions of the good practices and bad smells.

3 RESEARCH METHODOLOGY

To bridge the aforementioned gap, we conduct a systematic literature review that covers the relevant papers published between 2013 and 2024. This period was chosen because we seek to concentrate on the latest trends, mitigating the noises from the old and disappeared practices in the field. In particular, our review methodology follows the best practice of systematic literature review for software engineering [89, 90], as shown in Figure 2.

3.1 Automatic Search

As can be seen, in Stage 1 we conducted an automatic search over six highly influential indexing services, i.e., ACM Library, IEEE Xplore, Google Scholar, ScienceDirect, SpringerLink, and Wiley

Online, as they have been used in a number of systematic literature reviews in the field of software engineering, software configuration, and performance modeling [34, 50, 67, 69, 104, 131, 136].

By establishing a set of benchmark studies through a manual search over the six indexing services and analyzing their scope and keywords, the aim of the search string was to identify primary studies that:

- are relevant to software systems research;
- incorporate deep learning or neural network as part of the model;
- utilize configurations as the input for deep configuration performance models;
- apply the deep configuration performance models in some practical domains, e.g., predicting the performance of software or systems.

To that end, different combinations of keywords in these four areas were explored, and the quality of the automated search results was evaluated using the quasi-sensitivity and quasi-precision metrics by Zhang et al. [190], which will be explained thoroughly in Section 3.6. This ensures that the search results exclude redundant and irrelevant works as much as possible, retrieving papers that align with the research focus on the intersection of software configuration performance prediction and deep learning models.

Specifically, we compile the search string as below:

Search String

(“software" OR “system") AND “configuration" AND (“performance prediction" OR “performance modeling" OR “performance learning" OR “configuration tuning") AND (“deep learning" OR “neural network")

For all indexing services, the search resulted in 1206 studies, accounting for duplicates and excluding non-English documents. Among these, a total of 540 studies were obtained from Google Scholar, 152 studies from SpringerLink, 137 studies from Wiley Online, 190 studies from ACM Library, 132 studies from ScienceDirect, and 55 studies from IEEE Xplore.

3.2 Removing Duplication and Filtering Irrelevant Studies

Next, in Stage 2, we aim to ensure that only unique and highly relevant studies will be considered for further analysis in our review. To achieve this, we first filter out any duplicate studies by carefully examining the titles of the identified papers. This ensures that each study included in our review is distinct and contributes unique insights to the body of knowledge. Subsequently, we conducted a brief evaluation to eliminate any documents that were clearly irrelevant to configuration performance learning. For instance, we excluded studies focused on human and student performance, which is solely relevant to educational outcomes. Similarly, we disregarded papers centered on the performance of, e.g., physical systems, fuel systems, and football systems, as they are not directly aligned with the scope of our investigation.

As a result of these rigorous filtering procedures, we identified 251 highly relevant candidate studies.

3.3 Applying Inclusion and Exclusion Criteria

Through a detailed review of the candidate studies, Stage 3 focuses on applying various criteria to further extract a set of more representative works.

Firstly, we design a set of inclusion criteria to ensure that studies selected for detailed analysis align closely with the core themes of configuration performance learning. Specifically, in our

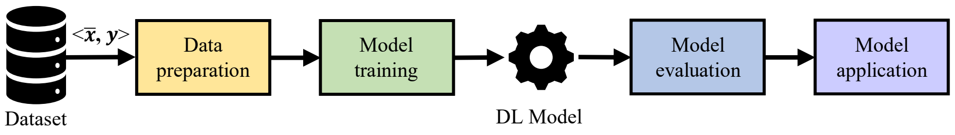


Fig. 3. Pipeline for deep configuration performance learning.

methodology, a study is temporarily selected as a primary study if it meets all of the following inclusion criteria:

- The paper presents a configuration performance modeling approach using deep learning algorithm(s).
- The paper explicitly states how the model built can be used, e.g., for predicting or analyzing the performance of a software system.
- The paper has at least one section that explicitly specifies the learning algorithm(s) used.
- The paper contains quantitative experiments in the evaluation with details about how the results were obtained.

The above is designed to maintain the relevance of the review and reduce the efforts required in the subsequent analysis. For example, the studies should ensure that they provide explicit details about the learning algorithms employed, as this is important for the understanding, analysis, and application of the model. Besides, it is essential for the inclusion of quantitative experiments to ensure a robust evaluation process, enhancing the depth and reliability of the selected studies.

Next, we applied the following exclusion criteria on the previously included study, which would be removed if it meets any:

- The paper is not software or system engineering-related.
- The paper is not published in a peer-reviewed public venue.
- The paper is a survey, review, tutorial, case study, or an empirical type of work.
- The paper is a short and work-in-progress work, i.e., shorter than 8 double-column or 15 single-column pages.

By excluding papers unrelated to software or systems, we ensure that the selected studies directly relate to the research focus of this survey. The requirement for peer-reviewed publication enhances the credibility of the included studies, while the exclusion of certain types of works (survey, review, tutorial, case study, or empirical) helps to filter out literature that may not align with our objective of investigating performance prediction using deep learning. Lastly, the limitation on the length of papers ensures that the selected studies possess sufficient depth and detail for a comprehensive analysis.

Finally, we identified 99 primary studies for detailed quality assessment and data extraction[3].

3.4 Collecting and Extracting Data

3.4.1 Research Questions. In the last stage, we derive the research questions (RQs) of this survey following the deep learning pipeline in Figure 3. Specifically, we formalize the workflow of deep learning for learning software configurations and performance into four key phases. The first phase is to process the raw configuration data collected and make preparations for their training with the deep learning models. To understand this, we ask the first research question:

[3]Raw data of all primary studies can be found at our repository: https://github.com/ideas-labo/DCPL-SLR.

RQ1: *How do the practitioners prepare raw configuration data for learning?*

Next, various deep learning algorithms might have been utilized to learn the preprocessed configuration data, handling different challenges in configuration performance prediction, such as sparsity, network structure, and network optimization. To summarize this, we seek to understand:

RQ2: *How do the practitioners build the deep configuration performance model?*

A crucial part of deep configuration performance learning is how to evaluate the trained deep configuration performance models, with a specific evaluation method, a set of testing samples, and a particular accuracy metric, where this process might be repeated for multiple subject systems. Therefore, the next question is:

RQ3: *How are the deep configuration performance models evaluated?*

Finally, the deep learning models need to be exploited in practical configuration scenarios, adapt to different environmental conditions, and enable researchers to replicate or reproduce the results. To that end, the last research question aims to understand:

RQ4: *How to exploit the deep configuration performance model?*

3.4.2 Data Items. After formulating the research questions, we have developed a comprehensive list of data items that we aim to collect during the review process. The summary of these data items is presented in Table 2, which includes a total of 22 data items, each serving a specific purpose related to the corresponding research question. In this section, we explain the design rationales behind them and clarify the procedure for extracting and classifying the data from each item.

In the initial stage, we employ data items I_1 to I_4 to gather meta-information of the reviewed studies, such as the title, author, and publication information. This meta-information enables us to accurately reference and organize the reviewed studies, thereby establishing a robust foundation for further analysis and interpretation.

To address RQ1, we seek to examine the key processes in preparing configuration data, where it is common to pre-process the raw configuration data, encode the data into the most appropriate form, and select a subset of informative samples for model training. To capture this information, we design data items I_5 to I_7 to extract the corresponding data preparation methods. For instance, by using I_5 we extract the pre-processing methods used in each primary study according to the fundamental types of methods, e.g., min-max scaling and z-score, which are all used to normalize the scale of the configuration options, can be classified into the same category.

For answering RQ2, we first review the methods used to deal with data sparsity in configuration data via I_8. Next, I_9 to I_{13} are for understanding how the deep learning models are chosen and trained according to their most general type. For example, on I_{11}, hyperparameter tuning methods can be set into three categories:

- using the default hyperparameter settings without any tuning;
- tuning the hyperparameters via manual effort and domain knowledge;
- or relying on automated heuristic methods in the tuning.

To examine the evaluation-related techniques in RQ3, we do not only examine the evaluation procedure (I_{14}) and metrics (I_{15}) but also the methods of statistical validation that ensure the statistical difference between the comparisons (I_{16} and I_{17}). The number and domain of subject

Table 2. The data items considered in this survey.

ID	Data Item	RQ	ID	Data Item	RQ
I_1	Authors	N/A	I_{12}	Activation functions	RQ2
I_2	Year	N/A	I_{13}	Hyperparameter tuning methods	RQ2
I_3	Title	N/A	I_{14}	Evaluation procedures	RQ3
I_4	Venue (conference or journal)	N/A	I_{15}	Accuracy metrics	RQ3
I_5	Data preprocessing methods	RQ1	I_{16}	Statistical tests	RQ3
I_6	Data encoding schemes	RQ1	I_{17}	Effect size measurements	RQ3
I_7	Data sampling strategies	RQ1	I_{18}	Number of subject systems	RQ3
I_8	Sparsity handling mechanisms	RQ2	I_{19}	Domain of subject systems	RQ3
I_9	Deep learning models	RQ2	I_{20}	Application categories	RQ4
I_{10}	Reasons for model selection	RQ2	I_{21}	Handling of dynamic environments	RQ4
I_{11}	Optimization algorithms	RQ2	I_{22}	Availability of code and dataset	RQ4

systems used in the evaluation, which may influence the generalizability of the conclusions, are also examined using I_{18} and I_{19}, respectively.

Lastly, for RQ4, we identify three data items to examine the exploitation information regarding the deep configuration performance models. In particular, I_{20} is employed to understand the application domains of the performance models, I_{21} summarizes the environmental conditions considered, as they significantly decide the generalizability and robustness of the deep configuration performance models, and I_{22} is applied to explore the data for replication and reproduction of the proposed models.

3.4.3 Data Collection. The data collection process aims to collect detailed information from the primary works with the data items presented in Table 2, and the collection protocol is the same as commonly used by the literature reviews in Software Engineering [104, 197]. Specifically, the protocol involves all the authors with three iterations.

- In the first iteration, each of the authors conducted initial data collection independently, read carefully throughout the 99 primary studies, extracted the data according to each data item, summarized the data into a table of 22 columns and 99 rows, and conducted preliminary classification. Notably, there were situations where certain data items could not be found (e.g., the encoding schemes were often ignored); the data items were not clearly stated (e.g., the sampling methods for training/testing data for evaluation were sometimes vague, with only abstract descriptive words like “case study” or “real dataset”); or the data needed to be conjectured from the contexts (e.g., the domains of subject software systems were often not specified and extra searches were needed to be performed for data collection). In these cases, the corresponding data items were marked for further investigation in the next iteration.
- In the second iteration, the authors reviewed and cross-checked each other’s data summary tables, ultimately integrating them into a unified table. The unclear data items that were marked in the previous iteration were rechecked and we arranged meetings to reach a common agreement for each table cell where the collected data from the authors were different. This is achieved through author discussions; further investigations from the existing literature; or consultation with external researchers/authors of the reviewed studies. For instance, the authors discussed and decided to add a category of “unknown” for encoding schemes since a large number of studies omitted the justification of the choice of their encoding method, and investigations on search engines like Google were conducted to collect the domain of subject systems.

Table 3. Quality assessment questions and scoring metrics.

ID	Question	Scoring Metric
Q_1	Does the study preprocess the configuration data to enhance learning?	No=0, Yes=2.
Q_2	Is the choice of data encoding methods clearly explained in the study?	Not explained=0, Implicitly explained=1, Clearly explained=2.
Q_3	Is the sampling strategy for training data outlined?	Not outlined=0, Implicitly outlined=1, Clearly outlined=2.
Q_4	Does the study address the sparsity/overfitting problem?	No=0, Yes=2.
Q_5	Does the study employ domain knowledge to improve the model architecture?	No=0, Yes=2.
Q_6	Are the reasons for selecting specific deep learning models provided?	Not provided=0, Implicitly provided=1, Clearly provided=2.
Q_7	Is the optimizer used in the deep learning models clearly stated?	Not stated=0, Implicitly stated=1, Clearly stated=2.
Q_8	Is the activation function employed in the deep models specified?	Not specified=0, Implicitly specified=1, Clearly specified=2.
Q_9	Does the study introduce the hyperparameter tuning method?	Not introduced=0, Implicitly introduced=1, Clearly introduced=2.
Q_{10}	Does the study outline the evaluation procedures for model performance?	Not outlined=0, Implicitly outlined=1, Clearly outlined=2.
Q_{11}	Are the accuracy metrics used for evaluation clearly defined?	Not defined=0, Implicitly defined=1, Clearly defined=2.
Q_{12}	Does the study employ statistical tests for significance analysis?	No=0, Yes=2.
Q_{13}	Does the study conduct effect size tests conducted to measure the practical significance?	No=0, Yes=2.
Q_{14}	Does the study consider multiple subject systems for evaluations?	Not mentioned=0, Only one system=1, More than one systems=2.
Q_{15}	Does the study consider multiple domains of subject systems?	Not mentioned=0, Only one domain=1, More than one domains=2.
Q_{16}	Does the study clarify the application purpose of	Not clarified=0, Implicitly clarified=1, Clearly clarified=2.
Q_{17}	Does the study consider the challenge of dynamic environments?	No=0, Yes=2.
Q_{18}	Are the codes and datasets available publicly for reproducibility?	No=0, Yes=2.

- Finally, the goals for the third iteration are to summarize the statistics of the integrated table from the second iteration; count the number of studies for each technique associated with every data item; and aggregate similar techniques into broader categories. For example, in the domain of subject systems, it was found that the number of studies that apply configuration performance models for video encoding software and image processing software are four and three, respectively, and they are combined as the category "multimedia processing" as these software systems share much more similar characteristics comparing with others.

This has finally led to the data for this work, which was discussed among all authors to make quality assessments and comprehensive classifications.

3.5 Quality Assessment

Based on the extracted information in the previous stage, we performed a quality assessment to evaluate the reliability and validity of the primary studies, which helps ensure that the studies selected for the review are of high methodological quality and provide reliable evidence for further data analysis.

In particular, following the guideline of SLR by Kitchenham and Charters [89], we defined a list of questions corresponding to the data items to quantify the comprehensiveness in each phase

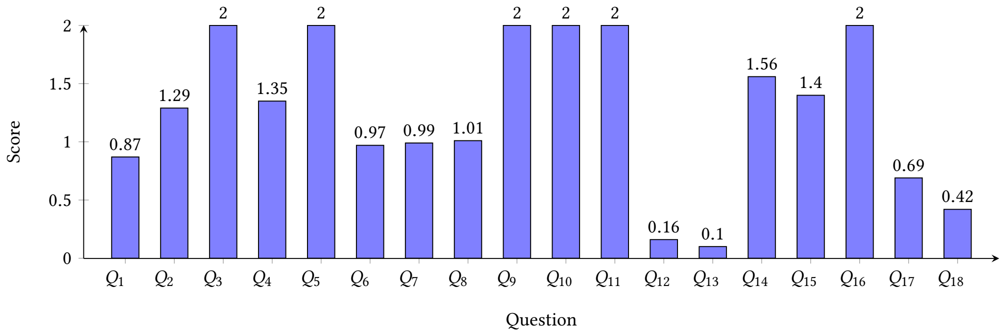


Fig. 4. Average scores for each quality assessment question.

Table 4. Detailed scoring of the top 10 primary studies in the quality assessment. Full scoring data can be found at: https://github.com/ideas-labo/DCPL-SLR/blob/main/Scoring.xlsb

Ref.	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}	Avg.
[58]	2	1	2	2	2	2	1	1	2	2	2	2	2	2	2	2	2	2	1.83
[57]	2	2	2	2	2	2	1	1	2	2	2	2	2	2	2	2	0	2	1.78
[56]	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	2	1.78
[64]	2	1	2	2	2	2	2	2	2	2	2	2	0	2	2	2	0	2	1.72
[38]	2	1	2	2	2	2	2	2	2	2	2	2	0	2	2	2	0	2	1.72
[13]	2	2	2	2	2	2	2	2	2	2	2	0	0	2	2	2	2	0	1.67
[191]	2	1	2	2	2	2	2	2	2	2	2	0	0	2	1	2	2	2	1.67
[161]	2	2	2	2	2	2	2	2	2	2	2	0	0	1	1	2	2	2	1.67
[107]	2	1	2	2	2	2	2	2	2	2	2	0	0	2	2	2	0	2	1.61
[26]	2	1	2	2	2	2	2	2	2	2	2	2	0	2	2	2	0	0	1.61

of the deep learning pipeline. Then, we scored each study based on the results of the assessment questions. In a nutshell, the questions and scoring metrics for assessing the primary studies are summarized in Table 3.

Remarkably, as depicted in Figure 4, all 99 studies achieved full scores in six questions, and the average score across all questions was 1.27 out of 2, indicating a high standard of the primary studies. Moreover, Table 4 provides a breakdown of the scores for the top 10 primary studies, where five of them reach a score of 1.72 or higher, accounting for 86% of the maximum score. It is worth noting that due to space constraints, the detailed scores of the 99 primary studies are included in the supplementary documents available on our public repository.

3.6 Quasi-Gold Standard Validation

Additionally, we conducted a validation of the sensitivity and precision of our automatic search strategy using the “quasi-golden standard” (QGS) by Zhang et al. [190]. Note that since the search covers a wide range of 78 venues, six indexing services, and 11 years of duration, it could be extremely expensive to conduct a grid search over the scope. Therefore, we only focused on primary studies that are published within the most recent five years (from 2019 to May 2024, where 79% of the primary studies are published according to Figure 1) and on five of the most representative and influential venues in the domain of software engineering for assessment [142], i.e., conferences including ESEC/FSE, ICSE and ASE, and journals including JSS and TOSEM (note that we did not choose TSE because none of the primary studies was published since 2019).

Table 5. The sensitivity and precision test using the quasi-gold standard in the latest five years and five top conferences/journals.

Publication Venue	Total # Studies Retrieved	# Relevant Studies Retrieved	# QGS Studies	QGS Studies
Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)	10	2	2	[58], [57]
International Conference on Software Engineering (ICSE)	3	2	2	[64], [53]
International Conference on Automated Software Engineering (ASE)	3	2	2	[12], [22]
Journal of Systems and Software (JSS)	8	1	1	[107]
Transactions on Software Engineering and Methodology (TOSEM)	7	1	1	[38]

In particular, by following the QGS procedures, we seek to verify the sensitiveness and exactness of our search string. We first manually defined a set of eight studies that are published in the selected venues and widely recognized as high quality, also known as the quasi-golden set [190], as specified in Table 5. Then, by performing an automatic search with our search string exclusively on the five conferences/journals using the ACM Digital Library, we retrieved a collection of 31 searched studies. After applying the inclusion and exclusion criteria, we identified eight relevant studies retrieved from the search. Subsequently, we calculated the sensitivity (%) s and the precision (%) p using the following formula as proposed by Zhang et al. [190]:

$$s = \frac{\text{\# relevant studies retrieved}}{\text{Total \# relevant studies in the QGS}} \times 100 \tag{4}$$

$$p = \frac{\text{\# relevant studies retrieved}}{\text{Total \# studies retrieved}} \times 100 \tag{5}$$

Notably, the sensitivity of our automatic search among the selected five venues is 100%—significantly better than the suggested 70% by Zhang et al. [190], demonstrating the outstanding effectiveness of our search strategy in identifying the most influential primary studies. On the other hand, the precision is 26%, which means there are some false positives. However, Zhang et al. [190] state that as a verification procedure, sensitivity is of the top importance in the literature review process.

3.7 The Taxonomy

With the collected statistics using the RQs, data items, and data collection protocol mentioned in the previous sections, we formalize into a taxonomy of deep configuration performance learning as shown in Figure 5. As can be seen, we found a variety of categories for the RQs and their data items. Some of these are specific algorithms (e.g., for the deep learning models used) while some others are of more general types (e.g., for the preprocessing methods). We will present detailed data statistics and introduce each category of techniques for deep configuration performance modeling in what follows.

4 RQ1: HOW TO PREPARE RAW CONFIGURATION DATA FOR LEARNING?

To improve the accuracy and reliability of the deep configuration performance model, data preparation is necessary [72]. In this section, we particularly review the preprocessing, encoding, and

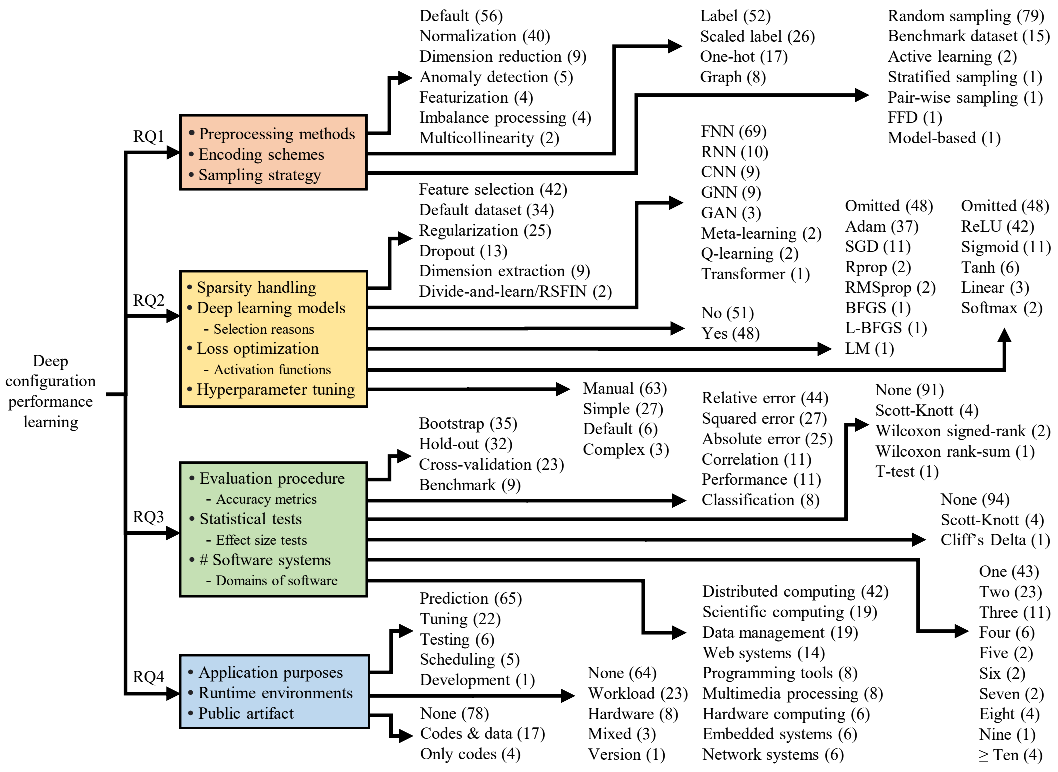


Fig. 5. A taxonomy of deep configuration performance learning. The number in the bracket denotes the corresponding number of studies.

sampling when preprocessing configuration data that are commonly used by the approaches of deep configuration learning in the community.

4.1 Preprocessing Configuration Data

Data preprocessing methods play a fundamental role in preparing data for deep learning problems [72], and understanding the preprocessing techniques employed in the context of configuration performance learning is crucial for researchers to gain insights and enhance the quality and reliability of the data.

In general, data preprocessing is the process of transforming raw data into a computable and noise-free format. This can be crucial for deep configuration performance learning since the configuration options might be of diverse scale, nature, and types. As an example, for the video encoding software VP8, there are binary configuration options like `allowResize` (0 or 1), categorical options like `sharpness` (no/low/medium/high sharpening), and numerical options such as `minGopSize` (from 0 to 1000).

Table 6 summarizes different preprocessing methods identified from our review. As can be seen, while various methods have been employed, a significant number of studies (56 out of 99) opt to use the **raw configuration performance dataset** without any preprocessing. Although this way may save effort and costs in the short term, it raises extra requirements on the quality of the datasets to ensure reliable results.

Table 6. Distribution of the preprocessing methods (one study might use items from multiple categories).

Category	Total #	Example	# Studies	References
Default	56	N/A	56	[153], [85], [146], [145], [12], [118], [182], [105], [54], [108], [119], [8], [86], [134], [22], [112], [56], [91], [42], [120], [147], [44], [15], [95], [158], [41], [48], [123], [49], [127], [169], [110], [96], [168], [87], [149], [46], [80], [40], [19], [109], [122], [43], [39], [141], [172], [138], [115], [47], [117], [128], [175], [106], [189], [103], [164]
Normalization	40	Min-max scaling	34	[57], [52], [195], [125], [184], [78], [53], [192], [186], [26], [144], [167], [163], [38], [45], [17], [93], [140], [116], [132], [61], [81], [161], [73], [64], [191], [160], [13], [7], [58], [183], [107], [174], [14]
		Standardization	6	[196], [132], [121], [13], [79], [177]
		Z-score	2	[81], [100]
		Shift-log transformation	2	[58], [174]
		Box-Cox transformation	1	[73]
		Centering	1	[73]
		Make unit consistent	1	[18]
Dimension reduction	9	PCA	8	[73], [61], [81], [134], [93], [196], [120], [79]
		CCA	1	[134]
		Knob2vec	1	[98]
Anomaly detection	5	Outlier detection	3	[81], [52], [18]
		Isolation forest algorithm	1	[162]
		Smoothing	1	[100]
Featurization	4	Feature construction	2	[2], [52]
		Discretization	1	[100]
		Alphanumeric cleaning	1	[18]
		Make columns categorical	1	[18]
Imbalance processing	4	Simple over-sampling	2	[100], [79]
		SMOTE	2	[57], [196]
Remove multicollinearity	2	Kendall's rank correlation	2	[18], [2]

In contrast, **normalization** techniques have been applied in 40 studies, which are fundamental in data preprocessing to ensure that the scales of configuration options are brought to a standardized range, thereby preventing any particular options from dominating the learning due to its larger values [152]. Several widely used normalization techniques for deep configuration performance learning include: Min-max scaling [56, 57, 64, 125], which transforms numerical options to a specific range, typically between 0 and 1, by subtracting the minimum value and dividing by the range of the option. Z-score [81, 100] transforms numerical options by subtracting the mean and dividing by the standard deviation of the option, resulting in a distribution with a mean of 0 and a standard deviation of 1, and allowing for easier comparison and interpretation across different options. Although normalization techniques have gained popularity and offer several advantages, it is crucial to acknowledge their limitations. For example, scaling the data might disrupt the original distribution and decrease the performance of certain deep learning algorithms that depend on the inherent characteristics of the original data. Additionally, normalization techniques like min-max scaling can be sensitive to outliers, as extremely big values of the configuration options can significantly impact the range of the scaled values, potentially compressing the majority of the data into a narrow range. Indeed, the `numCore` option of the stencil-grid solver HSMGP can range from 64 to 4096, while the largest value of the remaining configuration options is 6.

Methods for **dimensionality reduction** have been used in nine studies, which help to simplify the data representation, remove redundant or irrelevant options, and overcome the curse of dimensionality, thereby improving computational efficiency and visualization of the configuration data. For example, Principal Component Analysis (PCA) [61, 73, 81, 93, 134, 196] transforms the original options into a new set of uncorrelated variables called principal components, which are linear combinations of the original options and are ordered by the amount of variance they explain. Notably, Lee et al. [98] leverage an independent method, namely knob2vec, to process the workload-specific configuration option vectors that capture the unique characteristics and relationships of the configuration options in RocksDB, and then the data is fed into a separate deep learning model. We classify it as a preprocessing method because it is used to process the configuration data before the training phase. Indeed, Cheng et al. [38] also state that it is crucial to embed the extreme high-dimensional configuration space into low-dimension to better model the interactions between the options and performance. However, these methods may also require additional resources to find the best hyperparameters for the best reduction outcomes. They also could make the model difficult to interpret since the original option space is destroyed.

Further, five studies have focused on **anomaly detection** techniques, aiming at identifying data absences, errors, and outliers within a configuration dataset. For example, Cengiz et al. [18] filter out samples where the runtime is zero in the SPEC CPU 2017 dataset, and Fu et al. [52] opt to remove configurations with performance values beyond the 99th percentile. Examples include: Smoothing [100], which aims at capturing the underlying trends, patterns, or regularities in the configuration data while suppressing or filtering out the noise, and Isolation Forest [162] algorithm that constructs random decision trees to measure the average number of steps required to isolate an instance, allowing it to identify anomalies as instances of configuration that require fewer steps for isolation. Their limitations may include the potential loss of important details.

Another four studies have explored **featurization** approaches, where raw data unsuitable for machine/deep learning are transformed into meaningful options that can be used for modeling. For instance, discretization, as highlighted by Li et al. [100], involves converting continuous options into discrete and representative categories, to save efforts on exploring the huge configuration space. Particularly, the authors discretize the reach rate of a Content Delivery Network by calculating its fifth power, such that the reach rate is projected into several discrete intervals, and each interval is assigned a unique value. Yet, these methods may result in information loss, especially in situations where the software system is highly complex, and they often rely heavily on domain experts.

Besides, **imbalance processing** approaches have also been applied in four studies, which are crucial in dealing with imbalanced performance datasets with regard to the configurations, where the number of samples for different configuration values is significantly uneven. Indeed, in the configuration dataset of the widely used software Apache [38, 57, 64], there are 128 samples with value 1 for the configuration option InMemory, while only 64 samples with value 0, resulting in imbalance with this option. To handle this, oversampling methods could be used, which increase the number of instances from the minority class to balance the dataset. For example, Gong and Chen [57] employ SMOTE (Synthetic Minority Over-sampling Technique), which enhances the minority group by interpolating between neighboring instances, to balance the imbalanced divisions of samples within their "divide-and-learn" framework.

Lastly, two studies **remove multicollinearity** by addressing highly correlated options, e.g., Kendall's rank correlation [2, 18], which quantifies the strength and direction of the association between two ranked options, is used to remove highly correlated options and ensure that the predictors are independent or have a minimal correlation, which eventually reduces the cost of building deep configuration performance model.

Table 7. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the data preprocessing methods.

Category	**Strength**	**Weakness**	**Best-Suited Scenario**
Normalization	(1) Ensures options contribute equally. (2) Speeds up convergence in model training.	Sensitive to outliers.	Datasets with varying option scales.
Dimension reduction	(1) Removes redundant or irrelevant options. (2) Reduces computational cost.	Makes the model difficult to interpret since the original option space is destroyed.	High-dimensional and sparse datasets.
Anomaly detection	Identifies data absences, errors, and outliers.	The potential loss of important details.	Datasets with extreme noises.
Featurization	Captures meaningful features in raw data.	Requires domain knowledge.	When human experts are available.
Imbalance processing	Enhances accuracy when training samples are unevenly distributed.	Increases computational complexity.	Datasets with biased values.
Remove multicollinearity	Reduces variance in regression coefficients.	May exclude useful information.	When some options are highly correlated.

Finding 1: *The default datasets, without any preprocessing, are used by most studies (56 out of 99). Among the rest preprocessing methods, normalization is the most popular in handling configuration data, i.e., in 40 studies.*

4.1.1 Usage Scenario. Based on the review results, we summarize the strengths, weaknesses, and best-suited scenarios for each pre-processing method in Table 7, where researchers and practitioners can prepare their data in the most suitable way under different situations.

4.1.2 Good practice. Configuration performance data often exhibits some characteristics that are difficult to learn by deep learning models, and addressing these challenges can lead to improved model performance, which puts high demands on data preprocessing to improve the data quality. Therefore, it is a good practice that a variety of characteristics have been tackled in the primary studies. For example, the configuration space of software is demonstrated to be high-dimensional and sparse, which has been mitigated by regularization and dimension extraction techniques; and the primary studies also reveal that raw performance datasets often have outliers, imbalance, and multicollinearity issues. By addressing these issues, these studies have enhanced the effectiveness and efficiency of performance models and provided insights for future studies.

4.1.3 Bad Smell. It is observed that a majority of the primary studies (56 out of 99) have paid inadequate attention to the importance of preprocessing methods. Neglecting these crucial steps can severely limit the performance and effectiveness of deep learning models [137]. Therefore, it is imperative for future studies to recognize the significance of preprocessing techniques and incorporate them systematically to ensure optimal model performance.

Actionable Suggestion 1: *Incorporate different preprocessing techniques in the deep learning pipeline to address different data properties and ensure optimal deep configuration performance prediction performance.*

4.2 Encoding Configuration Options

The procedure immediately followed by data preprocessing is encoding, which is concerned with converting the configuration data into a more appropriate format for learning. Indeed, as discovered by Gong and Chen [56], the impacts of encoding schemes are non-trivial, thereby the encoding scheme can considerably influence the learning outcome.

Figure 6 provides a summary of the encoding schemes surveyed. Among others, **label encoding** [2, 12, 19, 153, 162, 172], emerges as one of the most popular schemes, employed by 52 primary studies. Label encoding assigns a unique numerical label to each value of the option. As such, label encoding preserves the ordinal relationship between categories but does not introduce additional dimensions like one-hot encoding. Although label encoding can simplify the representations of configuration options, it also introduces a random numerical order for those categorical options, e.g., for the highly configurable database system MongoDB, its configuration option `data_strategy` has a set of configurable values (str_l1, str_l2, str_l3), and label encoding will convert them into numeric values of $(0, 1, 2)$ [56]. This might potentially disrupt the inherent relationships between the configuration options and lead to worse predictions.

Similarly, **scaled label encoding** [17, 38, 45, 64, 73, 163], which is a common variation of label encoding, is chosen by 26 primary studies in our survey. Different from classic label encoding, the encoded configuration options of software systems are normalized into a specific range, e.g., between 0 and 1, to overcome the limitations of label encoding. For instance, the `minGopSize` option of VP8 will be label encoded from 0 to 1000, while the binary option can be only 0 and 1. This difference in scale may result in significant sparsity for performance models, which can be effectively handled by scaled label encoding.

In contrast, **one-hot encoding** [43, 53, 56, 61, 100, 161, 175], employed in 17 studies, transforms discrete options into a vector of binary values, where each element is represented as a binary form. For instance, for the video encoding system VP8, options like `sharpness`, which has categorical values no/low/medium/high sharpening, will be one-hot encoded into four binary variables. In this way, it enables deep learning models to appropriately interpret and capture relationships between different values. While one-hot encoding has been proven to be beneficial for the accuracy of performance models, it also leads to the highest training overheads [56].

We find that eight studies employ **graph encoding** that leverages graph representations, such as the computational graphs of neural networks [146, 153, 174], the directed acyclic graph (DAG) extracted from the workflows [106, 184], and structural graphs of applications [40, 44, 195], to represent configurations. For example, Singh et al. [153] consider deep neural networks as a configurable software system and use graphs to represent the computational stages and operations of a DNN to better capture the internal interactions while modeling the runtime of the deep learning pipeline. Graph encoding can lead to high-dimensional representations when dealing with a large set of options, potentially increasing the complexity and computational requirements.

Notably, a substantial portion of the work (i.e., 69 studies) opt to not explicitly specify encoding schemes used. Therefore, we have to inspect their codes or contact the authors to collect the data.

Finding 2: *Label encoding is applied to convert configuration values for learning in 52 studies, being the most common one, followed by scaled label encoding and one-hot encoding, used in 26 and 17 primary studies, respectively. Noteworthy, the majority of studies, 69 in particular, do not explicitly state their encoding schemes.*

4.2.1 Usage Scenario. Based on our review, we have summarized the strengths, weaknesses, and optimal use cases for each encoding method in Table 8. It serves as a guide for researchers and practitioners, helping them choose the most appropriate configuration data encoding scheme for various scenarios.

Table 8. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the data encoding methods.

Category	Strength	Weakness	Best-Suited Scenario
Label encoding	(1) Preserves ordinal relationships in categorical data. (2) Training efficiency and prediction accuracy were well-balanced.	Introduces a random numerical order for those categorical options.	When categorical options have a meaningful order.
Scaled label encoding	(1) Same strengths as label encoding. (2) Handles options in different scales to prevent dominating.	Sensitive to outliers.	For ordered categorical features with varying magnitudes.
One-hot encoding	(1) Captures the influence of each option value. (2) Beneficial for the accuracy of performance models.	Introduces extra training overheads.	When prediction accuracy is more important than training overhead.
Graph encoding	Captures complex relationships and structures between options.	Increases the complexity and computational requirements.	When data naturally forms a graph, such as computation graphs of deep learning models.

4.2.2 Good Practice. Encouragingly, it has been demonstrated that the choice of encoding scheme is critical since it profoundly impacts the prediction accuracy and time consumption of deep learning models [56]. For instance, one-hot encoding tends to yield the highest accuracy albeit with longer training times compared to other schemes, while label encoding often leads to faster training times. Thereby, they provide practical rules for choosing the best encoding scheme and enable future researchers to make more informative decisions.

4.2.3 Bad Smell. The influence of encoding schemes is underestimated by a majority of the studies, which solely rely on an ad-hoc way without providing justifications for the encoding schemes. This oversight can be detrimental, as it leaves other researchers unclear of which encoding schemes is suitable, potentially resulting in ambiguous data formats for deep learning tasks [5, 12].

This bad smell can be attributed to several reasons: (1) Researchers may lack awareness of the importance of encoding schemes in configuration performance learning, leading to a disregard for proper consideration and justification. (2) The absence of established guidelines or best practices regarding encoding schemes in the field can contribute to ad-hoc approaches and a lack of justification. (3) Time and resource constraints, limited emphasis on publications, and limited domain expertise in encoding schemes further compound this issue. These factors collectively leave researchers unsure of suitable encoding schemes, potentially resulting in ambiguous data formats for deep learning tasks, and highlighting the need for increased awareness, guidelines, and clearer justifications in this area.

Actionable Suggestion 2: *Justify the rationals behind the choice of encoding schemes for deep configuration performance prediction to facilitate the understanding of this topic.*
Actionable Suggestion 3: *Establish guidelines or best practices for encoding schemes for researchers in the community to avoid ad-hoc approaches and save trial-and-error costs.*

4.3 Sampling Configurations for Learning

Sampling, which is typically performed to guide the raw configuration data collection, is also a critical step in data preparation, as the quality of the collected configuration samples determines the prediction accuracy of the deep learning models [136].

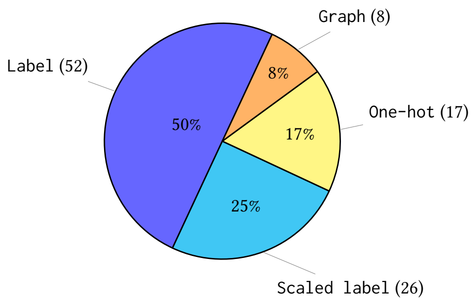


Fig. 6. Distribution of the encoding schemes (one study might use items from multiple categories).

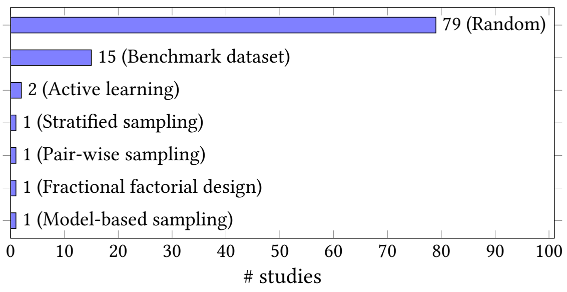


Fig. 7. Distribution of the sampling strategies (one study might use items from multiple categories).

Figure 7 shows the categories of the sampling strategies used in the reviewed studies. We can first notice that an overwhelming majority of the studies surveyed, specifically 79 out of 99, employ **random sampling** as a prevalent data sampling technique in their research. Random sampling involves selecting data instances from the original configuration dataset in an unbiased and random manner. The widespread adoption of random sampling highlights its efficiency and effectiveness in creating balanced and diverse samples for training [53, 108, 160, 167, 172, 192]. However, random sampling may not efficiently capture the less frequent values of options, and these values may be underrepresented in the selected set, leading to bad generalizability of the deep learning models. For instance, for the web server software Apache, the value of the option `InMemory` is 1 in 128 configurations, while the value 0 only appears 64 times in this option, therefore the training samples collected by random sampling will have a higher possibility to miss the value 0, especially when the training sample size is small, leading to bad accuracy when the model needs to predict new configurations with 0 for `InMemory`.

In contrast, 15 of the primary studies follow a common practice of utilizing the default sets of training and testing samples that are pre-defined from the **benchmark** dataset [18, 43, 46, 96, 125], which allows for a direct comparison of their models' performance against the existing results. For instance, Cengiz et al. [18] only consider the training and testing workloads from the SPEC CPU 2017 benchmark. While they provide a standardized reference for representing specific scenarios or workloads, their applicability to real-world or diverse environments may be limited.

Two studies apply **active learning** to obtain the samples [109, 146], which seek to select the most informative samples to be measured for training, thereby accelerating the learning process and reducing the costs of data collection. Yet, active learning methods may put high demand on uncertainty estimates or heuristics to select samples.

For the strategies that are part of the minority, each of the following is adopted by one primary study: **Stratified sampling** [163] involves dividing the population into homogeneous subgroups based on specific characteristics and then sampling from each subgroup proportionally to its representation in the overall population. It also requires prior knowledge about the population's stratification variables and their distribution. **Pair-wise sampling** [95] seek to sample data that achieve maximum coverage of interaction effects between pairs of options while minimizing the number of samples required to prevent biases. Indeed, when dealing with systems that have a large number of configurations, pair-wise sampling may become impractical due to the sheer number of possible combinations. **Fractional factorial design** [73], on the other hand, provides a way to select a subset of factor combinations that represent the most important effects. While it allows for investigating a large number of options with a reduced number of samples, its generalizability is

Table 9. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the data sampling methods.

Category	Strength	Weakness	Best-Suited Scenario
Random	(1) Simple and easy to implement. (2) Efficient and effective in creating balanced and diverse samples.	Can miss rare but important options.	For exploratory analysis with no prior knowledge.
Benchmark	Enables direct comparison against existing models.	May not be applicable to real-world scenarios.	In well-defined environments with established benchmarks.
Active learning	Can select the most informative samples.	Puts high demand on heuristics to select samples.	When collecting data is expensive.
Stratified	Ensures representation of all subgroups.	Requires knowledge of subgroup proportions.	When the dataset has distinct subgroups.
Pair-wise	Maximum coverage of interaction effects between pairs of options.	Impractical due to the sheer number of possible combinations.	When the options have strong correlations with each other.
Fractional factorial design	Reduces the number of experiments needed.	Generalizability is limited by the assumption of linearity and consistency of the options.	When the configuration space is large, linear, and has consistent options.
Model-based	Efficiently explores the search space.	The choice of the surrogate model is critical.	When there is an accurate surrogate model.

limited by the assumption of linearity and consistency of the configurations. Lastly, **Model-based sampling** [169] builds a surrogate model capable of estimating uncertainty to effectively guide the search for selecting optimal training configurations. As a result, the accuracy of the surrogate model is critical.

Finding 3: *Random sampling is the most widely employed strategy for selecting configuration samples as observed in 79 studies, followed by using the predefined samples from the benchmark as exploited by 15 studies. Other sampling strategies form the minority.*

4.3.1 Usage Scenario. Based on the comprehensive review, we have identified the advantages, limitations, and best-suited scenarios for each sampling method, which are summarized in Table 9. Thereby, this table serves as a valuable reference for researchers and practitioners to choose the most configuration data sampling approach based on specific situations and requirements.

4.3.2 Good Practice. The configuration space can be enormous, while the resources available for collecting training samples are often limited. Therefore, it is crucial to carefully select the most appropriate configurations to train performance models. However, one of the challenges lies in determining the criteria for sample selection. In the current literature, a good practice is that a few studies have addressed this issue by employing heuristic sampling methods tailored to specific requirements. Especially, Wang et al. [169] leverage SMBO to guide the sampling process with a surrogate Gaussian Process model to minimize the uncertainty of the samples in each iteration. By combining with various surrogate models, SMBO can potentially yield diverse sampling results, providing potential for future studies to investigate.

4.3.3 Bad Smell. Despite the availability of various sampling strategies, a significant portion of the studies tends to focus on the most straightforward approach, namely random sampling. While random sampling is effective in representing the distribution of the dataset, it can lead to overfitting in highly sparse systems, as many of the features may prove ineffective for software performance analysis [136]. This can be even more problematic considering the fact that there exist invalid configurations [75]. Therefore, researchers need to explore alternative sampling methods in such scenarios to enhance the accuracy of the performance models.

Table 10. Distribution of the sparsity handling mechanisms (one study might use items from multiple categories).

Category	Total #	Example	# Studies	References
Feature selection	42	Manual feature selection	24	[119], [105], [118], [184], [15], [112], [45], [123], [48], [52], [121], [86], [168], [8], [49], [158], [163], [138], [17], [127], [144], [100], [54], [95]
		Correlation-based feature selection	5	[167], [120], [162], [196], [183]
		Tree-based feature selection	4	[116], [61], [80], [2]
		NN-based feature selection	4	[160], [19], [53], [108]
		Filter-based feature selection	2	[192], [134]
		Wrapper-based feature selection	2	[161], [26]
		Metaheuristic feature selection	1	[196]
Default dataset	34	N/A	34	[40], [47], [147], [115], [18], [128], [96], [41], [78], [44], [91], [39], [122], [109], [43], [140], [12], [22], [85], [110], [153], [132], [42], [146], [169], [125], [46], [182], [106], [7], [98], [177], [107], [103]
Regularization	25	L_1 regularization	16	[57], [160], [172], [87], [149], [64], [38], [161], [127], [167], [141], [93], [56], [58], [189], [14]
		L_2 regularization	10	[160], [158], [80], [149], [161], [127], [167], [56], [79], [189]
		Early stopping	4	[13], [174], [164], [14]
		Noise regularization	1	[134]
		Bayesian regularization	1	[117]
		Laplacian regularization	1	[195]
Dropout	13	Dropout	12	[160], [112], [175], [145], [186], [149], [100], [191], [183], [189], [174], [14]
		Ensemble pruning of network	1	[117]
Dimension reduction	9	PCA	8	[73], [61], [81], [134], [93], [196], [120], [79]
		CCA	1	[134]
		Knob2vec	1	[98]
Divide-and-learn	1	N/A	1	[57]
RSFIN	1	N/A	1	[107]

Actionable Suggestion 4: *Avoid over-reliance on random sampling and explore alternative sampling methods to improve the quality of training data for better configuration performance prediction accuracy.*

5 RQ2: HOW CAN THE DEEP CONFIGURATION PERFORMANCE MODEL BE BUILT?

A deep learning model can be trained once the data preparation has been completed. To understand the state-of-the-art at this stage for deep configuration performance learning, this section summarizes details on how deep learning can be effectively built to learn configuration performance, offering insights into best practices and potential problems in this field.

5.1 Handling Configuration Sparsity and Preventing Overfitting

As large-scale software systems generate vast amounts of data, sparsity arises when valuable information is scarce in the options or unevenly distributed across the configuration space. As a result, overfitting occurs when deep learning models excessively learn the configuration data used for training, leading to poor generalization on unseen samples [2, 57, 61, 64, 120, 122, 146, 149, 162]. For instance, the option use_gpu which enables GPU processing for the video codec x264 has a large influence on the runtime, and deep learning models tend to assign a large weight to this option while ignoring the influence of others, leading to overfitted predictions.

Table 10 presents the survey results of mechanisms used to handle sparsity and overfitting. Notably, 34 out of 99 studies have **not adopted any specific mechanism** to handle sparsity, which could harm the accuracy of deep configuration performance models.

Among the remaining studies, 42 primary research papers have leveraged **feature selection** techniques as a means to deselect the undesired configuration options, hence mitigating sparsity. In particular, feature sparsity is observed when there exists a set of non-influential configuration options, which adds redundant parameters to the configuration performance model. Therefore, by eliminating these options with feature selection, the performance model will be resilient from sparsity. However, it is worth noting that 24 out of the 42 studies simply rely on human effort to select the most important options, which is inefficient and lacks generalizability. Among others, the following statistics are found:

- Five studies have applied correlation-based feature selection techniques, such as correlation analysis [120, 162, 167, 183, 196], that leverages statistical measures to determine the strength of relationships between options.
- Neural Network-based feature selection applies neural network layers to identify relevant features, which have the inherent ability to learn and extract informative options from high-level representations, as demonstrated by four studies. For example, CNNs [160] (Convolutional Neural Network) are commonly used in image-related tasks to extract local patterns and spatial hierarchies, and GNNs [19] (Graph Neural Networks) are particularly effective for structured data represented as graphs, which analyze relationships and dependencies among nodes in a graph to capture important patterns and interactions.
- Four other studies have used tree-based feature selection, which inherently enables analyzing the structure and splits of the trees to determine the most informative options for prediction. For instance, extra trees regression [116] involves constructing multiple decision trees with randomized splits to estimate feature importance, and Grohmann et al. [61] utilize the ensemble nature of random forest to rank options based on their contribution to the model's predictive power.
- Moreover, each of the following feature selection methods is used by two primary studies: (1) Filter-based feature selection techniques evaluate each feature independently of the target option and rank them based on predefined criteria. An example is the minimum Redundancy Maximum Relevance Feature Selection (mRMR) [192], which seeks a balance between selecting options that have high relevance to the performance while minimizing redundancy among selected options; (2) Wrapper-based feature selection [161] uses Recursive Feature Elimination (RFE) to iteratively eliminate options with the least importance, while evaluating the model's performance using cross-validation. In particular, Tousi et al. [161] select 10 most important options like `max_mhz`, `log_cpus` and `sockets` in the SPEC CPU 2017 dataset using RFE with four different estimators. Hybrid model also exists as proposed by Chen and Bahsoon [26]. The core is to combine the outputs of multiple learning algorithms to guide the feature selection process.
- Finally, metaheuristic feature selection [196], which combines a whale optimization algorithm and simulated annealing to construct a search-based option selection algorithm named EMWS, is explored in one study.

Alternatively, 25 studies have employed **regularization** techniques, which introduce additional constraints or penalties to the model's objective function, aiming to control sparsity, reduce overfitting, and enhance the robustness of the performance models. Several commonly used examples are:

- L_1 regularization [57, 64, 87, 149, 160, 172], also known as Lasso regularization, adds a penalty term proportional to the absolute value of the model's coefficients, encouraging sparsity in the model by driving some coefficients to zero and reducing the impact of irrelevant features. For instance, after performing L_1 regularization on the configuration options of the video encoding software VP8, the coefficients of options like `autoAltRef` and `allowResize` are reduced to zero because they have trivial influence on the performance (runtime), while the coefficient of options like `bestQuality` is changed from 10^{16} to 1.9×10^4, significantly reducing the sparsity of configurations.
- L_2 regularization [80, 149, 158, 160, 161], also called Ridge regularization, adds a penalty term proportional to the squared magnitude of the model's coefficients. It encourages smaller coefficients and can help mitigate multicollinearity issues by reducing the impact of highly correlated options. However, regularization methods come with the trade-off of introducing additional parameters that require tuning to achieve optimal results, which may incur higher costs in terms of time, computational resources, and expertise needed to find the best parameter values [57, 64].

Dropout techniques are widely used to eliminate insignificant model parameters during the neural network training process in 13 prominent studies. Particularly, dropout [100, 112, 145, 149, 160, 175, 186] works by randomly deactivating a fraction of neurons during the training phase of a neural network. By doing so, dropout prevents the network from relying too heavily on specific neurons, forcing it to learn more robust and generalized features. In addition, Mahgoub et al. [117] applied ensemble pruning of networks to prune the top 30% of the ensemble networks to prevent overfitting. Compared to feature selection and regularization, the advantages of dropout are its simplicity and computational efficiency, which is particularly useful for training large neural networks. Yet, this also causes a lack of interpretability, as dropout does not provide explicit information about which specific options or neurons are important or unimportant.

Meanwhile, data preprocessing techniques like **dimension extraction** have also been leveraged to eliminate unnecessary representations and information from the original dataset in nine studies of deep configuration performance learning. Unlike feature selection techniques, which only focus on mitigating specific options, dimension extraction approaches can achieve simultaneous handling of features, which is crucial when there are interactions or dependencies among features. This is because they work on the overall weight magnitudes, allowing the deep learning models to strike a balance between the negative impacts of the features and their potential interactions with others. For example, Lee et al. [98] embed knobs of configurable software systems into a latent space representation to extract the most informative features and use together for configuration performance modeling, thereby reducing the sparse representations of configuration data.

Furthermore, although a number of studies have focused on addressing feature sparsity, where a small subset of features holds significant influence, there are very few studies that deal with sample sparsity, in which samples exhibit substantial variations in performance, resulting in a non-smooth distribution. For example, Gong and Chen [57] have shown that by dividing the samples of the video codec VP8 according to the option `rtQuality`, there are two clusters of samples with significantly diverse distributions, i.e., the division with `rtQuality` = 0 has runtime between 0 and 10000 ms, while the other ranges from 10000 to 60000. In light of this gap, they proposed a framework based on the idea of **"divide-and-learn"**, i.e., `DaL`, which divides the original dataset into distinct

Table 11. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the sparsity handling techniques.

Category	**Strength**	**Weakness**	**Best-Suited Scenario**
Feature selection	Removes non-influential options.	Requires domain knowledge to select options.	When dealing with high-dimensional data.
Regularization	Mitigates overfitting by penalizing complex models.	Requires careful tuning of regularization parameters.	When facing overfitting with complex models.
Dropout	Prevents the network from relying too heavily on specific neurons.	Causes a lack of interpretability.	For models with large parameters.
Dimension reduction	Eliminates unnecessary representations and information.	May lose important information if over-reduced.	When handling complex and high-dimensional datasets.
Divide-and-learn	Addresses sample sparsity by dividing the samples into more focused divisions.	Requires tuning the number of divisions to divide.	When the samples are distributed into sparse clusters.
RSFIN	Captures the hidden structures and distributions in the configuration space.	Struggles to achieve ideal accuracy in highly complex and rugged configuration landscapes.	When the configuration space exhibits common patterns or structures.

divisions based on hidden characteristics and learns a local deep learning model for each division to mitigate sample sparsity [57]. In another study by Li et al. [107], **`RSFIN` (Rule Search-based Fuzzy Inference Network)** is designed to capture configurations with low information entropy, as they are more likely to represent common patterns or structures in the data, thereby, `RSFIN` can effectively capture the hidden structures and distributions in the configuration space, leading to a better understanding of the system behavior.

Finding 4: *A considerable number of studies do not explicitly handle the problem of sparsity and overfitting (34 out of 99). Among the studies that deal with this challenge, manual feature selection is the most common method (24 studies).*

5.1.1 Usage Scenario. Based on our comprehensive review, we have identified the distinct advantages, limitations, and optimal usage scenarios for each sparsity handling method. These findings are concisely summarized in Table 11, providing researchers and practitioners with a valuable reference to select the most appropriate approach for addressing the sparsity problem in their specific scenarios.

5.1.2 Good Practice. Encouragingly, a majority of studies recognize the significance of addressing sparsity and overfitting, which are critical for causing the degradation of predictive models, urging the application of a diverse range of handling techniques [2, 57, 61, 64, 120, 122, 146, 149, 162]. However, it is worth noting that most of these techniques are option selection methods as the configuration options often have redundant information and most of them have little influence on the software performance. Yet, Gong and Chen [57] discover that except for feature/option sparsity, which has been realized by most studies, the distribution of samples in the configuration landscape is also sparse—a significant property in the configuration performance learning problem that is worth further investigating.

5.1.3 Bad Smell. Given that sparsity has been addressed by 65 out of 99 papers, 24 of them rely on human efforts to select the correct feature sets to reduce sparsity, which is of low efficiency and low generalizability. In addition, there are still 34 works that did not acknowledge the sparsity issue. Moreover, only one study focused on solving the sparsity problem from the perspective of sample distributions by dividing the sparse clusters into more focused divisions each learned by a local deep model [58], which could be a key aspect of future research..

Table 12. Distribution of the deep learning models (one study might use items from multiple categories).

Category	Total #	Example	# Studies	References
Feedforward neural network	69	Multilayer perceptron	60	[132], [26], [46], [47], [91], [48], [144], [134], [49], [78], [117], [160], [141], [172], [8], [40], [42], [162], [125], [96], [43], [95], [115], [195], [138], [18], [17], [118], [120], [127], [110], [167], [140], [163], [161], [80], [147], [45], [52], [121], [86], [73], [119], [56], [93], [2], [61], [87], [109], [100], [39], [122], [81], [123], [13], [183], [189], [164], [14], [79]
		Regularized DNN	3	[64], [57], [58]
		Kernel extreme learning machines	2	[196], [168]
		Fuzzy neural network	1	[107]
		Hierarchical interaction neural network	1	[38]
		Dynamic neural networks	1	[145]
		Radial basis function neural network	1	[128]
Recurrent neural network	10	Long short-term memory	9	[158], [175], [85], [41], [116], [108], [100], [167], [192]
		Gated recurrent unit	1	[98]
Convolutional neural network	9	Standard CNN	8	[186], [112], [15], [22], [54], [196], [167], [103]
		Residual neural network	1	[18]
Graph neural networks	9	Standard GNN	7	[44], [53], [19], [153], [146], [106], [174]
		Graph hyper network	1	[7]
		DAG-transformer	1	[184]
Adversarial learning	3	Generative adversarial network	3	[149], [105], [12]
Deep meta-learning	2	Model-agnostic meta-learning	1	[169]
		Sequential meta-learning	1	[58]
Deep reinforcement learning	2	Q-learning network	2	[182], [191]
Transformer	1	Transformer	1	[177]

Actionable Suggestion 5: *Future researchers should avoid relying heavily on human efforts for feature selection, but explore automatic approaches to improve efficiency and generalizability.*
Actionable Suggestion 6: *It would be highly beneficial to address sample sparsity by handling the sparse distribution of samples in the configuration landscape, instead of solely focusing on feature sparsity.*

5.2 Choosing Deep Learning Model

5.2.1 Deep Learning Models. It is not hard to anticipate that there will be a variety of models applied to deep configuration performance learning, as shown in Table 12. Specifically, the majority of primary studies (69 out of 99) employ a **Feedforward Neural Network (FNN)**. Among these, the most prevalent approach is Multilayer Perceptron (MLP) in 60 studies. MLP is a fundamental neural network consisting of interconnected neurons organized in multiple layers. These neurons apply activation functions to the weighted sum of their inputs, introducing non-linear transformations to the data. Ha and Zhang [64] propose the utilization of regularization techniques in regularized Deep Neural Networks (rDNN), such that additional penalties are introduced during training to remove insignificant features, control feature sparsity of the model and thereby prevent overfitting. Gong and Chen [57, 58] combines rDNN with the divide-and-learn framework to further address the sample sparsity issues. In addition, Kernel Extreme Learning Machines (KELM) are also used in two studies, aiming at simplifying the training process of neural networks, which randomly initializes the weights between the input and hidden layers, then applies a kernel function to obtain the transformed feature representation, and analytically determines the weights between the hidden and output layers based on the transformed features. A most recent work [38] employs a Hierarchical Interaction Neural Network (HINN) that leverages a hierarchical structure for performance learning, where lower-level layers typically capture low-level options, and higher-level layers learn to combine these low-level options into more complex representations. Further, dynamic neural networks and radial basis function neural networks are each utilized in one study.

However, FNNs cannot handle sequential or temporal data effectively in many real-world tasks, such as natural language processing and time series prediction. To mitigate this, **Recurrent Neural Networks (RNN)** are specifically designed to maintain an internal memory or hidden state that persists across time steps, allowing them to capture information from past inputs. Among our review scope, RNNs have been employed in 10 works. For example, Long Short-Term Memory (LSTM) is a dedicated version of the recurrent neural network with internal memory cells, which allows them to selectively retain or forget information based on the input and previous context by using specialized units called gates, and therefore overcome the limitations of traditional RNNs in capturing and remembering long-term dependencies in sequential data. For example, Tang et al. [158] employ LSTM to tune the performance of file systems like Ext4, F2FS, and PMFS, where the operations and workloads are dynamically changing over time and it is crucial to apply time-related deep learning models to model the temporal data.

On the other hand, **Convolutional Neural Networks (CNN)** have been studied in nine research studies to process grid-like structured data, such as images and videos, which can not be handled by FNNs. The hierarchical feature learning capability of the convolutional layers allows CNNs to automatically learn and extract meaningful representations from the input data. Notably, a particular variant of CNNs, known as Residual Networks (ResNets), has been exploited by Cengiz et al. [18], which incorporate novel architectural designs with residual connections to address the issue of vanishing gradients during training. As an example, Liu et al. [112] leverage the ability of CNN that can disclose the hidden and complex correlations among different options to predict the performance of high-performance computing systems like RISC-V.

In addition, **Graph Neural Networks (GNN)** and its variant DAG-transformer have also been used in nine studies, which utilize graph convolutions and message-passing techniques to capture complex relationships and dependencies within graph structures composed of nodes and edges. For example, Chai et al. [19] utilize GNN to predict the inference latency, energy, and memory footprint of DNN inference, because it can directly process arbitrary DNN TFlite graphs and has good generalizability.

Further, Generative Adversarial Networks (GAN) is an **adversarial learning** approach that is primarily designed for generative modeling tasks, aiming at learning the underlying distribution of the training data and generating new samples that resemble the training data. It consists of two neural networks: a generator network and a discriminator network, which are trained in an adversarial manner, with the generator trying to produce more realistic samples to deceive the discriminator, and the discriminator trying to become better at distinguishing real from generated samples. For instance, GAN is used by Bao et al. [12] to automatically generate configuration samples for different software systems like Kafka, Spark, and MySQL to save performance measurement costs.

Deep meta-learning, where meta-learning paradigms are used jointly with deep learners, has been explored in two studies. Specifically, Model-Agnostic Meta-Learning (MAML) [169] pre-trains a neural network's parameters using the known software environments to facilitate fast adaptation to new environments, allowing for efficient generalization and rapid adaptation. Most recently, a state-of-the-art sequential meta-learning model (SeMPL) is proposed by Gong and Chen [58], which builds a regularized deep neural network to learn a set of meta-environments in a specific sequence that can discriminate the influence of different environments and maximize the usage of meta-data.

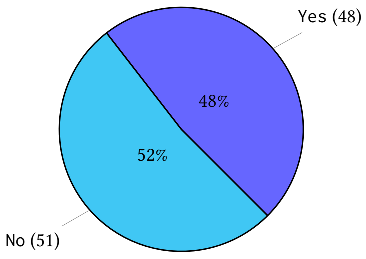


Fig. 8. Distribution of the reasons for choosing deep learning model.

On the other hand, **deep reinforcement learning** approaches like Q-learning networks are used in two studies to approximate the action-value function (Q-function), which represents the expected cumulative reward for taking a specific action from a given state and following a particular policy. As an example, Yin et al. [182] utilizes a Q-network to deal with the dynamic changes of workloads while predicting the configuration performance for multitier web systems like RUBiS.

Lastly, Wyzykowski et al. [177] employ **transformer** models, which are based on a self-attention mechanism that allows the model to weigh the importance of different input elements when making predictions, capturing the long-range dependencies in the sequential data of high-performance computing workloads more effectively compared to RNNs or CNNs.

5.2.2 Rational Behind the Choices. In addition to the statistics of deep learning models, our survey also investigates whether the studies have provided an explanation for their choice of specific learning models, which is captured in Figure 8. Among the 99 studies analyzed, it was found that only 48 of them have provided **explicit reasons** for the utilization of a specific learning model, accounting for approximately 48% of the total studies examined. On the other hand, a significant portion of the studies, 51 in total, do **not offer any justification** or rationale behind their choice of learning model.

Finding 5: *Various deep learning models have been applied for deep configuration performance learning, within which the most common one is the multilayer perception (a kind of FNN) from 60 out of 99 studies. However, it is worth noting that nearly 52% of the primary studies do not provide explicit justifications for their choice of deep learning model.*

5.2.3 Usage Scenario. Based on the above review results, we summarize the strengths, weakness, and best-suited scenario for each deep learning model in Table 13, where researchers and practitioners can discover the most suitable deep learning model under different situations.

5.2.4 Good Practice. Section 5.2 shows that the very basic version of DNN, i.e., MLP is the most commonly used learning model being employed by a total of 60 studies. This is evidence that a simple deep learning model is useful for learning software configurations and performance. This derives the exploitation of more complex DNN, for example, rDNN [57, 64] is a DNN armed with regularization to address the problem of feature sparsity, and ResNet [18]—a type of neural network that incorporates residual connections to overcome the challenges of vanishing gradient for configurations embed into high dimensional representation. This demonstrates the potential of the deep learning models by addressing specific challenges and sheds light on future studies on improving the existing models.

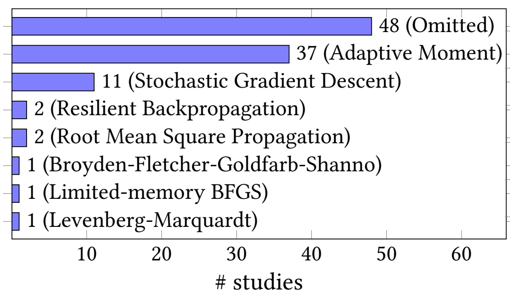
Table 13. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the deep learning models.

Category	Strength	Weakness	Best-Suited Scenario
FNN	(1) Simple architecture. (2) Scalable with more neurons and layers. (3) Flexible to be paired with various techniques [107].	Poor handling of sequential or temporal data [100].	Performance prediction with non-sequential configuration data.
RNN	(1) Effective with sequential data. (2) Captures temporal dependencies [85].	Computationally intensive due to continuous learning.	Continuous performance prediction with a time series of dynamic operations/workloads.
CNN	(1) Effective in capturing structured and hierarchical configuration data. (2) Robust to translations and distortions in input data [112].	Limited ability to model long-range dependencies.	Performance prediction for configuration data along with other data modalities, such as sequence data of unit codes and defect matrices of the configuration.
GNN	(1) Capable of handling graph-structured data. (2) Ability to capture relational dependencies between nodes [174].	High computational cost and complex implementation.	Software dependency analysis of configuration and performance; network topology-based predictions; relationship-driven performance factors.
Adversarial learning	(1) Ability to learn the underlying distribution of the training data. (2) Capable of generating new samples that resemble the training data [12].	Computationally expensive due to the need for adversarial training.	Configuration data synthesis when available data is limited.
Deep reinforcement learning	(1) Ability to learn from interactions with an environment. (2) Suitable for sequential decision-making problems [191].	Requires a large number of interactions with the environment to learn optimal policies.	Making sequential decisions for tasks like resource scheduling to maximize rewards in an environment.
Deep meta-learning	(1) Ability to learn from multiple related tasks. (2) Fast adaptation to new tasks with limited data [58].	Requires meta-data from a diverse set of tasks/environments.	Leveraging data on known environments for rapid modeling for unseen environments.
Transformer	(1) Effective in capturing contextual information. (2) Provides insights into the model's decision-making process [177].	High memory consumption due to the self-attention mechanism's quadratic complexity with respect to input length.	Performance modeling for context-rich configuration data.

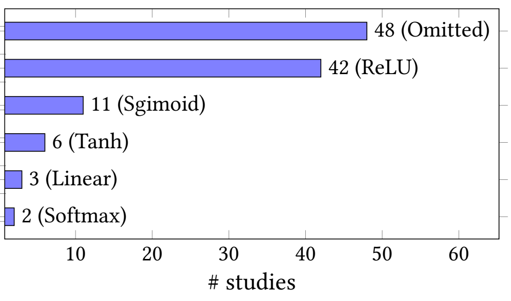
5.2.5 *Bad Smell.*

Recall from Section 5.2, there are 52% studies (51 out of 99) that provide no justification for their choices of deep learning models, which is a concern. It is essential for researchers to clarify their reasons for selecting specific models, as this enhances the understanding of the research motivations and contributions [111]. In the meantime, the lack of explicit explanations of model selection could be harmful to the community as researchers may miss out on important considerations or potential limitations.

The absence of justifications can be attributed to researchers assuming that the chosen models are widely accepted or commonly used, hence not realizing the need to provide explicit justifications. To address this issue, it is crucial for researchers to recognize the importance of providing clear explanations for their model selection, enhancing transparency, and promoting a deeper understanding within the research community.



(a) Optimization methods



(b) Activation functions

Fig. 9. Distribution of the loss function optimization techniques (one study might use items from multiple categories).

Actionable Suggestion 7: *Leverage domain knowledge in configuration performance learning to design dedicated DNNs like* `HINNPerf` *for handling hierarchical configuration data and* `DaL` *for addressing sparsity in configuration options and sample distribution.*
Actionable Suggestion 8: *Avoid omitting justifications for deep learning model choices and provide clear explanations to enhance research transparency and understanding.*

5.3 Optimizing Learning Loss

The effective training of deep learning models heavily relies on the choice of optimization methods that reduce the loss of training the activation functions, as the loss determines how the model's parameters are updated during training, while activation functions introduce non-linearity and enable complex representations within the neural network.

Figure 9a provides data on the optimization methods employed in the examined studies. A notable observation is that: a significant number of studies, 48 in total, **omitted** the specification of the loss optimization method employed [2, 44, 61, 81, 95, 110, 141]. Despite this omission, **Adaptive Moment Estimation (Adam)** optimizer emerges as the most prevalent optimization method, being utilized in 37 studies [38, 56, 57, 64, 161, 186], which is an adaptive optimization algorithm that maintains adaptive learning rates based on the first and second moments of the gradients. **Stochastic gradient descent (SGD)**, which is an iterative optimization algorithm that updates the model parameters by computing gradients on small randomly sampled subsets of the training data, stands as the second most frequently used method, appearing in 11 studies [8, 78, 123, 127, 132, 145]. **Resilient backpropagation (Rprop)**, which individually determines the appropriate step size for each parameter based on the sign of the gradient during training [26, 46], and **root mean square propagation (RMSprop)**, an extension of SGD that adapts the learning rate for each parameter based on the root mean square of recent gradients [140, 172], are both applied in two studies. Besides, all the rest methods are used by one study, i.e., Broyden-Fletcher-Goldfarb-Shanno (BFGS) [115], Limited-memory BFGS (L-BFGS) [127], and Levenverg-Marquardt [86]. Among others, when dealing with the high-dimensional, sparse, and non-linear configuration performance data of highly configurable software, Adam is a promising choice among others as it combines adaptive learning rates and momentum, efficiently handling sparse gradients and noisy data [88].

In Figure 9b, an overview of the activation functions employed in the surveyed studies is provided. Similar to the data of optimization methods, a significant number of studies, 48 out of 99, did not explicitly report the activation function employed. Despite this omission, **Rectified Linear Unit**

Table 14. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the deep learning model optimizer.

Category	Strength	Weakness	Best-Suited Scenario
Adam	Automatically adjusts learning rates.	Requires more memory due to storing past gradients.	In settings where quick convergence is desired.
SGD	Simple to implement and requires less memory.	Requires careful tuning of the learning rate.	In situations where simplicity and computational efficiency are key.
Rprop	Fast convergence by adapting step sizes.	Requires more memory for storing step sizes.	For batch training scenarios.
RMSprop	Adjusts learning rate based on recent gradient magnitudes.	Requires tuning of the decay hyperparameter.	In scenarios with noisy gradients.
Lavenberg-Marquardt	Combines gradient descent and Gauss-Newton methods.	Memory-intensive and computationally expensive.	Effective for small and medium-sized datasets.
BFGS	Uses second-order information for faster convergence.	Memory-intensive due to second-order derivatives.	For learning tasks with smooth gradients.
L-BFGS	Reduces memory usage by storing limited history.	Requires careful tuning of hyperparameters.	In scenarios where memory efficiency is important.

(ReLU) emerges as the most prevalent, utilized in 42 studies [8, 44, 115, 132, 196], which is known for its simplicity and effectiveness in handling non-linearities. **Sigmoid** activation function follows as the second most commonly used function appearing in 11 studies [26, 46, 47, 49, 186], and it can map the output of a deep learning model to a probability-like value between 0 and 1. In addition, the **hyperbolic tangent (tanh)** activation function, which is a smooth, symmetric, and nonlinear function that maps the input to a continuous range between -1 and 1 [52, 120, 161, 172], and **linear** activation functions are employed in six and three studies, respectively [61, 78, 95], while **softmax** is used in two studies [54, 61], which ensures the output values lie in the range of 0 to 1 and sum up to 1, representing valid probabilities. Notably, in the context of configuration performance learning, where it is often the case that only a subset of configuration options holds significant influence over performance, ReLU offers a natural solution by effectively zeroing out negative inputs, effectively focusing on the important configuration options [129].

Finding 6: *Despite that 48 of the primary studies omit the justifications of optimization and activation functions, we observe several different techniques, where the Adam optimizer and ReLU activation function stands out as the most common one, used in 37 and 42 studies, respectively.*

5.3.1 Usage Scenario. Based on our comprehensive review, we have identified the distinct advantages, limitations, and optimal usage scenarios for each model optimization method and activation function. These findings are concisely summarized in Table 14 and 15, providing researchers and practitioners with a valuable reference to select the most appropriate approach for training the deep learning performance model in their specific scenarios.

5.3.2 Good Practice. In configuration performance learning tasks, the best optimization method for reducing the loss in training differs depending on the systems and performance attribute. For example, tasks with sparse features may benefit from adaptive optimization methods like Adam, while SGD is often used in situations where computational resources are limited. Thus, given that the relationship between configurations and performance of highly configurable software is often sparse, it is encouraging to find that 24 studies address this problem by leveraging Adam optimizer.

Similarly, choosing the appropriate activation function can impact the DL model's performance, e.g., ReLU is effective in capturing complex, non-linear relationships. Hence, it is a promising

Table 15. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the activation functions.

Category	Strength	Weakness	Best-Suited Scenario
ReLU	(1) Simple and computationally efficient. (2) Zeroing out negative inputs, effectively focusing on the important configuration options.	Can suffer from the "dying ReLU" problem where neurons become inactive if they fall into the negative region and stop updating.	When some options are redundant and non-influential.
Sigmoid	Produces a smooth and continuous output, which can be interpreted as a probability.	Prone to the vanishing gradient problem.	When interpretability as probability is essential, such as interpreting the predicted reach rates as probabilities of events related to CDN performance.
Tanh	Stronger gradients than sigmoid, reducing vanishing gradient issue.	Computationally more expensive than ReLU.	For hidden layers in recurrent neural networks.
Linear	Does not suffer from vanishing gradient problem.	Limited representation power; can't capture non-linear relationships.	For simple networks where non-linearity is unnecessary.
Softmax	Ensures the output values lie in the range of 0 to 1 and sum up to 1, representing valid probabilities.	Sensitive to outliers and large input values.	Useful to generate probabilities for different performance states.

behavior to see that ReLU is used in 42 primary studies to handle the non-linearity specializing in the problem of configuration performance prediction.

5.3.3 Bad Smell. Surprisingly, nearly half of the studies (48 out of 99) omitted information on their choices of optimization and activation approaches. This omission poses a significant challenge to understanding and reproducing the deep learning models' results and findings [56, 57, 64]. Therefore, justification for these decisions should be provided in future works, as it allows practitioners to comprehend the rationale behind the decision-making.

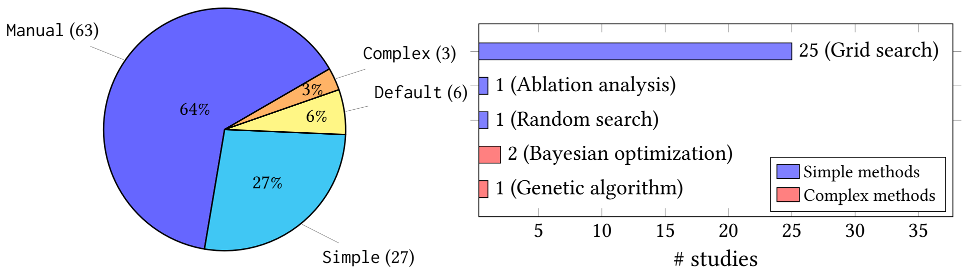
Actionable Suggestion 9: *Do not omit details on the choices of model optimization methods and activation functions in studies, instead, provide justifications to aid understanding and reproducibility.*

5.4 Tuning Hyperparameters

Hyperparameter tuning plays a critical role in the accuracy and generalization ability of deep learning models, especially for configuration performance learning, where the interactions between the configurations and performance are complex and can change largely and nonlinearly across software [75, 77]. However, determining the optimal values for hyperparameters is a challenging task as the hyperparameter space is often huge and the tuning is time-consuming.

Figure 10 presents an analysis of the hyperparameter tuning methods utilized in the examined studies. It is most worth noting that, a great portion of 63 studies, rely on **human experts** and domain knowledge to tune the hyperparameters. Among others, six primary studies simply rely on the **default** hyperparameter settings [42, 73, 105, 116, 145, 162], where models are trained using pre-defined configurations without any explicit tuning.

In addition, three **simple search methods** have been applied to tune the hyperparameters. For instance, grid search, which exhaustively tests over a predefined grid of values to find the best configuration, is the most widely used technique and is applied in 27 studies [38, 57, 64, 85, 146, 153]; ablation analysis [196], a way that gradually increases the number of layers and selects the hyperparameter setting with the highest AUC value, and random search [160], where the search path is randomly sampled from the hyperparameter space, are each employed by one study.



(a) Hyperparameter tuning methods. (b) Simple and complex tuning methods.

Fig. 10. Distribution of the hyperparameter tuning methods.

Table 16. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the hyperparameter tuning methods.

Category	**Strength**	**Weakness**	**Best-Suited Scenario**
Grid search	Exhaustive search covers all combinations.	Computationally expensive and time-consuming.	Suitable for limited hyperparameter sets.
Ablation analysis	Identifies the impact of each hyperparameter.	Not exhaustive; may miss optimal combinations.	For diagnosing model performance issues.
Random search	Reduces computational cost by sampling randomly.	May miss optimal settings due to randomness.	When computational efficiency is required.
Bayesian optimization	Utilizes past evaluations to guide search.	Sensitive to the choice of acquisition function.	In optimization tasks requiring a balance between exploration and exploitation.
Genetic algorithm	Can escape local optima through mutation and crossover.	Requires tuning of genetic algorithm parameters.	For complex hyperparameter spaces.

Then, more **complex hyperparameter tuning** methods, which leverage more complicated heuristics such as probabilistic models, evolutionary algorithms, or learning-based approaches to intelligently explore the hyperparameter space, are applied in three studies. For example, Bayesian optimization, a method that uses probability models to efficiently search for optimal hyperparameters by considering the performance of previously evaluated configurations, is employed in two studies [163, 186]. Additionally, genetic algorithm is explored by one study [54], which leverages the principles of natural selection to iteratively search for optimal hyperparameter configurations.

Finding 7: *Manual hyperparameter tuning is preferred by the majority of studies (63 out of 99). For the automated tuning method, grid search is the predominately popular one (25 studies).*

5.4.1 Usage Scenario. After conducting a thorough analysis, we have identified the unique benefits, constraints, and optimal application scenarios associated with each hyperparameter tuning technique. The results are presented in Table 16, serving as a practical resource for researchers to determine the most suitable approach for optimizing the hyperparameters of deep performance models.

5.4.2 Good Practice. With the complicated architectures of deep learning models, one challenge for hyperparameter tuning is the explosion of search space. As such, we are pleased to reveal that 33 primary studies have addressed this challenge by exploring different heuristic methods to guide the tuning of hyperparameters while training deep learning performance models. For

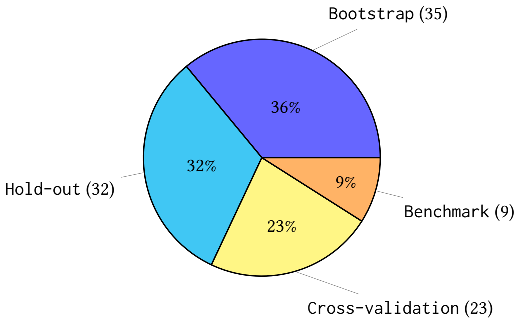


Fig. 11. Distribution of the model evaluation procedures (one study might use items from multiple categories).

instance, Ghamizi et al. [54] employ the idea of natural selection and evolution to find the best population of hyperparameter configurations, which is robust to noise and outliers as the overall fitness is based on the collective behavior of the population rather than individual solutions. This diverse range of heuristic methods reflects the researchers' efforts to explore different approaches for optimizing hyperparameters and enhancing model performance, which should be kept by future researchers.

5.4.3 Bad Smell. A great portion (63 out of 99) of papers choose to tune the hyperparameters manually, which is time-consuming and not generalizable. Besides, a subset of six studies relies solely on default hyperparameters that may not be optimal for all scenarios. This bad smell can be attributed to several reasons. Firstly, researchers may opt for manual tuning due to familiarity or convenience, as it allows them to have direct control over the hyperparameter settings. Secondly, the lack of awareness or availability of automated tuning methods may contribute to the prevalence of manual tuning. Additionally, some studies may rely on default hyperparameters as a time-saving measure or due to limited resources. This bad smell may restrict the exploration of the optimal hyperparameter settings and hinder the potential of deep learning models [51, 180]. As such, it is important for researchers to consider automatic tuning methods in order to maximize the performance and robustness of deep learning models.

Actionable Suggestion 10: *Researchers should employ automatic and heuristic approaches for hyperparameter tuning to navigate the vast search space effectively.*

6 RQ3: HOW ARE THE DEEP CONFIGURATION PERFORMANCE MODELS EVALUATED?

To understand how to evaluate the deep learning models for configuration performance learning, in this section, we summarize the procedure, metrics, statistical validation, and subject software systems that are commonly used in comparing the models.

6.1 Following Evaluation Procedures and Metrics

In Figure 11, we provide a summary of the evaluation procedure and metrics used. Among others, the most prevalent evaluation procedures is **bootstrap**, utilized by 35 out of the 99 studies [86, 108, 110, 145, 149, 192], which is a resampling technique that involves creating multiple datasets by randomly sampling from the original data, allowing for robust estimation of model performance. **Hold-out** evaluation, another widely used method, is employed in 32 studies [19, 42, 95, 122, 167, 184]. This approach involves splitting the dataset into training and testing sets based on a specific percentage.

Table 17. Distribution of the accuracy metrics (one study might use items from multiple categories).

Category	Total #	Example	# Studies	Reference
Relative error metrics	44	Mean absolute percentage error/Mean relative error	34	[17], [100], [87], [121], [57], [125], [85], [153], [2], [120], [118], [141], [123], [119], [112], [19], [167], [56], [81], [149], [127], [49], [134], [109], [38], [64], [53], [13], [58], [79], [189], [107], [174], [132]
		Relative error	5	[46], [78], [115], [7], [105]
		Mean percentage error	2	[47], [13]
		Symmetric MAPE	1	[26]
		Relative absolute error	1	[138]
		Root mean square relative error	1	[52]
		Relative percentage deviation	1	[106]
		Mean relative error percentage	1	[17]
Squared error metrics	27	Root mean squared error	16	[100], [145], [116], [167], [53], [162], [95], [117], [192], [96], [160], [42], [73], [41], [79], [177]
		Mean squared error	14	[17], [167], [86], [158], [144], [168], [18], [40], [162], [93], [41], [98], [103], [195]
Absolute error metrics	25	Mean absolute error	20	[172], [128], [110], [146], [39], [48], [8], [91], [122], [17], [161], [100], [145], [167], [18], [95], [41], [79], [177], [164]
		Normalized MAE	2	[80], [147]
		Absolute error	1	[168]
		Median absolute percentage error	1	[93]
		Median absolute error	1	[116]
Correlation metrics	11	Coefficient of determination (R^2)	10	[161], [145], [167], [18], [93], [162], [95], [73], [41], [183]
		Pearson correlation coefficient	1	[98]
Performance metrics	11	Performance improvemet	6	[182], [22], [108], [12], [45], [54]
		Optimal performance	5	[169], [43], [184], [191], [14]
Classification metrics	8	F-measure	5	[175], [186], [61], [163], [196]
		Mean average precision	1	[15]
		Positive predictive value	1	[140]
		Percentage of right predictions	1	[44]

Cross-validation, which involves iteratively partitioning the dataset into multiple subsets for training and testing, is used by 23 studies [78, 96, 118, 134, 141, 162]. Additionally, we find that nine studies opt to use the default evaluation pipeline provided by **benchmark** datasets, simplifying the evaluation process [43, 80, 87, 117, 182]. For instance, Kim et al. [87] run a subset of Linpack benchmarks on different hardware platforms to collect the power consumption data, representing a variety of industry-standard workloads, to build the models in the offline learning stage, while using the rest benchmarks for evaluating the online identification stage.

Table 17 presents the statistics and taxonomy of the accuracy metrics to evaluate the performance models. Among the identified metrics, **relative error metrics**, emerges as the most commonly used in 44 studies [38, 49, 53, 64, 109, 134]. Therein, Mean Absolute Percentage Error (MAPE) or Mean Relative Error (MRE), which measures the percentage difference between the predicted and actual values, is seen in 34 primary studies, being the most widely used metric. **Squared error metrics** follow as the second most frequently employed metric, appearing in 27 studies, e.g., Root Mean Square Error (RMSE) [41, 42, 73] and Mean Squared Error (MSE) [144, 158, 168] are utilized in 16 and 14 studies, respectively, providing insights into the model's ability to capture both small and large errors. Besides, **absolute error metrics** like Mean Absolute Error (MAE) [18, 41, 100, 145, 167] calculate the average absolute difference between the predicted and actual values, offering a straightforward measure of model accuracy, utilized in 25 reviewed studies. **Correlation metrics** like coefficient of determination (R^2)—a metric indicating the proportion of variance in the dependent variable explained by the model—is used in 11 studies [18, 93, 162].

Table 18. Summaries of the strengths, weaknesses, and best-suited usage scenarios for the model evaluation methods.

Category	Strength	Weakness	Best-Suited Scenario
Bootstrap	Accounts for sampling variability and provides robust performance estimates.	Prone to overfitting if the number of bootstrap samples is too high.	When the number of available samples is limited.
Hold-out	Simple and computationally efficient.	May not capture the full variability of model accuracy.	Quick evaluation of model performance.
Cross-validation	Provides a more stable estimate of model performance by averaging results.	Computationally expensive, especially with large datasets.	Evaluating the performance of models with high variance.
Benchmark	Allows for standardized evaluation.	Relies on the availability of well-defined benchmark datasets and metrics.	Assessing the performance of a new method against established baselines.

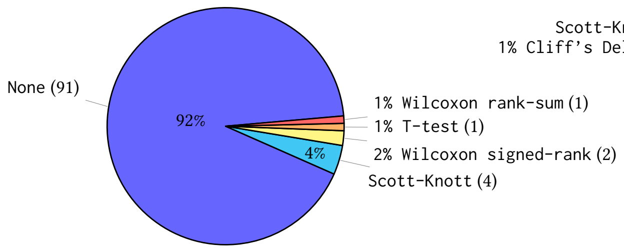
Additionally, **performance metrics** like performance improvement [12, 22, 45, 108, 182] and optimal performance [43, 169, 184, 191], which are computed using the performance resulting from a selected configuration, are used by six and five works, respectively. Notably, while these two metrics do not quantify the accuracy of deep configuration models directly, they are practical to evaluate how the deep configuration performance models can impact performance optimization, performance testing, and configuration tuning tasks. For instance, Chen et al. [22] measure the increment in the latency and energy consumption to examine the effectiveness of their model. Lastly, **classification metrics** such as F-measure are employed in eight studies, which is commonly used to examine the accuracy in classification tasks like predicting the software defect classes [175, 186, 196] or performance levels [61, 163].

Finding 8: *Bootstrap is the most prevalent evaluation procedure, which is utilized in 35 out of 99 studies. On the other hand, the most popular accuracy metric is MAPE/MRE in 33 studies.*

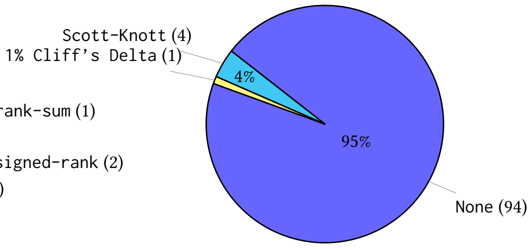
6.1.1 Usage Scenario. Based on our thorough review, we have identified the unique advantages, limitations, and optimal usage scenarios for each model evaluation method, as summarized in Table 18. By considering the strengths and weaknesses of each method, researchers and practitioners can make informed decisions to ensure accurate and reliable evaluations of their models, leading to improved understanding and advancement in the field of deep configuration performance learning.

6.1.2 Good Practice. A variety of metrics for accuracy has been applied is a rather positive sign for the community, as Mathews and Diamantopoulos [124] point out, no single measure gives an unambiguous indication of the modeling performance. More importantly, we have discovered that all the primary studies have leveraged some form of repeated evaluation procedure. This is crucial to ensure that the results were not produced due to chance or biased samples.

6.1.3 Bad Smell. We found that MAPE is the most widely used metric, due primarily to the fact that it is unit-free and easy to calculate. However, it is known that MAPE suffers the problem of being *asymmetric* on the error above and below the actual value [59], where the former receives a much greater penalty. As a result, such a strong bias tends to be problematic. To address this issue, researchers should be encouraged to explore alternative metrics that provide a more balanced and comprehensive evaluation of error, promoting a more accurate assessment of model performance.



(a) Statistical test methods



(b) Effect size test methods

Fig. 12. Distribution of the statistical validation methods.

Actionable Suggestion 11: *Implement repeated evaluation procedures in the experiments to ensure results and findings are reliable and not due to biased evaluation setup.*
Actionable Suggestion 12: *Researchers should consider using multiple metrics that can offer a more balanced evaluation of errors.*

6.2 Leveraging Statistical Validation

Since the deep learning models are stochastic in nature, evaluation in deep configuration performance learning is often conducted with repeated runs. Therefore, understanding their statistical significance is important. Indeed, when evaluating a deep configuration performance model, the results of two runs with exactly the same model settings and training/testing sample sizes could still be different, due to the stochastic loss optimization and the randomness in selecting the training and testing configurations, e.g., when evaluating with the compression tool Lrzip, two runs under the same conditions might resulted a MRE of 17.49% or 70.44% — a nearly 4× difference [57].

In Figure 12a, we survey the methods used for statistical validation. Surprisingly, the majority of the studies, 91 out of 99, do not utilize any specific statistical test methods to measure the significance of the results compared. Among the studies that do apply statistical tests, the **Scott-Knott Effect Size Difference (ESD)** test emerged as the most commonly used, appearing in three studies [56, 57, 196], which is a non-parametric test that groups data into distinct subset with significant difference. Additionally, the **Wilcoxon signed-rank** test was employed in two studies [26, 196], providing a non-parametric test for comparing two paired sets of data. Furthermore, the **t-test** [64] and the **Wilcoxon rank-sum** [38] test were each used in one study, offering comparisons for two independent data groups.

Similarly, Figure 12b reveals a phenomenon in the utilization of effect size tests for evaluating deep configuration performance learning, where the majority of the studies, 94 out of 99, do not employ any specific effect size tests. Among the rest studies, the **Scott-Knott ESD** test is used in three studies [56, 57, 196], which can identify significant differences on the effect size between groups in addition to its ability of assessing the statistical significance. Further, [196] utilized **Cliff's Delta**, a non-parametric effect size measure that quantifies the ordinal association between two variables.

Finding 9: *A significant portion of 92% and 95% studies omit the usage of statistical and effect size tests, respectively. Scott-Knott ESD test is used in 4 studies, standing out as the most common method for both cases.*

6.2.1 Good Practice. Almost all studies have conducted comparisons between their models with existing works, and a few of them have utilized statistical tests and effect size tests to ensure the statistical significance and effect size difference of the comparison. This practice is vital in deep learning research as it enhances the reliability and robustness of the findings.

6.2.2 Bad Smell. A notable concern arises from the fact that a significant majority of studies, around 92% (91 out of 99), did not conduct any statistical tests, and 95% of studies did not measure the effect size. This represents a significant problem as it introduces a potential source of randomness and uncertainty in the comparison results, limiting the ability to draw reliable conclusions and increasing the risk of causing severe external threats to validity. This highlights the need for greater awareness and adoption of statistical analysis in deep learning research.

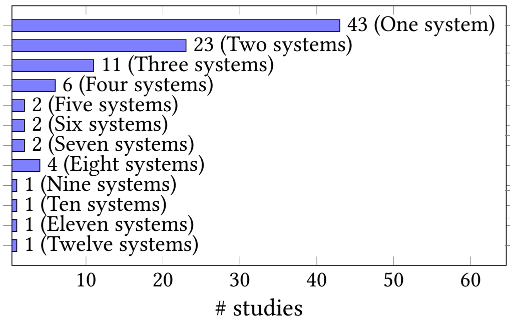
Actionable Suggestion 13: *Researchers should pay more attention to the importance of statistical and effect size tests to avoid external validity threats and to strengthen the reliability of their experiments.*

6.3 Evaluating Configurable Software Systems

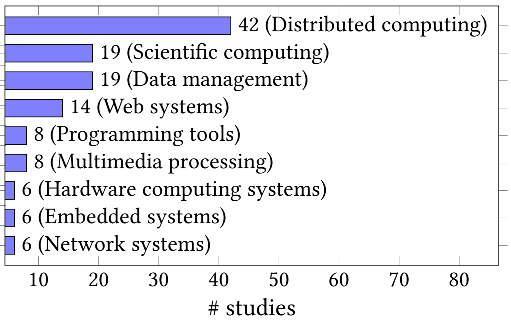
A crucial factor in evaluating deep configuration performance models is to ensure the evaluation covers a good range of subject systems. Figure 13a provides the distribution of the number of systems considered in the primary studies. Among others, 43 studies evaluate a **single software**, 23 studies evaluate **two software**, 11 studies assess **three systems**, six studies evaluate **four subject systems**, and the remaining 16 studies evaluate five or more systems.

The taxonomy of the domains of the subject software systems in the primary studies is shown in Figure 13b. Among these domains, 42 prominent studies build deep configuration performance models for **distributed computing** systems, which typically involve distributing computational tasks across multiple interconnected devices or nodes, enabling collaboration and parallel processing for improved computing performance, including domains like cloud computing [47, 80, 95, 123, 125, 127], big data processing [57, 85, 122, 134, 191, 195], and high-performance computing [78, 87, 109, 128, 146]. In learning configuration performance of distributed computing systems, the key challenges include capturing the dependencies and interactions between multiple interconnected components, the dynamic environments with nodes joining or leaving the system, network conditions changing, and workloads varying over time. To address these, researchers leverage domain knowledge in extracting diverse environmental features, constructing domain-specific data structures to capture the interconnection between the distributed components, and designing specialized deep learning model architectures. For example, Betting et al. [14] propose a contextual multi-armed bandit approach to improve resource recommendations by adapting with application requirements, hardware capabilities, and cost considerations, and Li et al. [106] utilize Graph Neural Network to identify and learn patterns of the interconnected nodes, enabling efficient performance modeling based on recognized patterns.

Scientific computing systems, used in 19 primary studies, play a crucial role in solving complex scientific and research problems by utilizing advanced algorithms and numerical methods, including domains like matrix multiplication [48], multi-grid solvers [38, 149], stencil-grid solvers [149], and the training of deep learning models [19, 52, 53, 108, 128, 153], enabling researchers and scientists to analyze and model intricate data sets efficiently. Modeling scientific computing software systems faces several key challenges: (1) scientific computing applications frequently leverage parallelism and hardware accelerators to accelerate computations, which requires considering related factors such as load balancing, communication speed, and hardware environments, and (2) scientific computing systems are often optimized for specialized mathematical operations or problem domains,



(a) # subject systems considered



(b) Domains of subject systems

Fig. 13. Distribution of the subject systems (one study might use items from multiple categories).

putting requirements on domain-specific knowledge and tailored modeling strategies. The primary studies address these challenges by building tailored configuration options that feature the unique demands of the scientific applications, e.g., Trümper et al. [164] encodes both static and dynamic features of parallel loop nests, and Wang et al. [174] design a subgraph-level performance prediction method that combines operator-level and graph-level features to model the application structure, enabling more effective configuration performance predictions.

Another 19 studies focus on the modeling of **data management** systems that facilitate the storage, organization, retrieval, and manipulation of data, including software systems that provide structured data storage and query capabilities [85, 122, 134, 191, 195], I/O libraries that facilitate efficient input/output operations with data [73], file systems that manage file storage and access [158], and compression tools that reduce data size for efficient storage and transmission [57]. The major challenges in modeling data management systems with deep learning models involve: (1) they often involve concurrent access to data and parallel processing, thereby, predicting the performance of these systems requires considering factors such as locking mechanisms, transaction management, and parallel query execution to accurately model system behavior under varying workloads, and (2) these systems often experience workload variability due to changing data volumes, query patterns, and access frequencies which requires adaptive modeling techniques that can adjust to varying workloads and resource demands. For instance, Zhang et al. [191] employ a reinforcement learning model (Q-learning network) to learn the interactions between the knobs and performance and predict the optimal knob settings through a try-and-error method.

Furthermore, **Web service** software like web server systems, involving the utilization of internet-based technologies to provide a platform for hosting websites, is explored in 14 studies [12, 57, 64, 128, 140, 182]. Notably, deep configuration performance prediction for Web service applications encounters challenges with fluctuating workloads based on user traffic, and uncertainty in web service environments, such as co-hosted services on the same server. To address these issues, multi-environment learning techniques like meta-learning have been utilized by Gong and Chen [58], which can discriminate the importance of each meta-environment and learn the environments in a tailored sequence, efficiently reusing the knowledge from the meta-environments.

Additionally, **Programming tools** and **multimedia processing** are each used in eight studies. Programming tools include Python programs [175] and compilers, which translate high-level programming languages into machine-readable instructions to allow the code to be executed by the computer [38, 44, 57, 64, 128, 149]. Multimedia processing systems include image processing [22, 38, 149], which focuses on techniques for enhancing, modifying, and analyzing images, as well as video encoding[38, 57, 64, 128], which involves the compression and encoding of video data to optimize storage and transmission while maintaining high-quality playback. Particularly, configuration performance learning of programming tools suffers from challenges such as domain-specific and language-specific features which require tailored feature construction for accurate predictions, and dependencies on external toolchains and libraries which can impact the performance of programming tools. A common practice among the primary studies is to leverage domain knowledge of the specific programming tool in feature selection, thereby modeling the interactions between the configuration options and software performance more efficiently [44]. Moreover, multimedia processing software often needs to deal with diverse workloads and types of input files, such as images, audio, or videos, which could significantly affect the performance behavior of the software. Thus, domain knowledge is required for designing workload-aware models, e.g., Chen et al. [22] propose `DeepPerform` to efficiently generate test samples to model the configuration performance and detect Input-Dependent Performance Bottlenecks for image processing applications.

Hardware computing systems, explored in six studies, comprise diverse domains such as CPU computing [161, 172], GPU applications that leverage graphics processing units for general-purpose computing [81], chip multi-processors [96], and multicore systems [39], all of which contribute to enhancing computational capabilities, efficiency, and parallel processing within computer systems [18]. Hardware computing systems face challenges in the complex architecture of GPUs, varying levels of parallelism in GPGPU applications, and the difficulty in tuning kernel and architectural parameters without a thorough examination of the design space. To mitigate these and enable more accurate performance predictions, one of the critical ways is to reduce the size of the configuration space by limiting parameter ranges, as performed by Jooya et al. [81].

Finally, six research works examine **embedded systems** and **network systems**, respectively. Especially, embedded computing includes mobile systems [160, 168], which are designed for smartphones and tablets, Internet of Things (IoT) communication systems that facilitate connectivity and data exchange among IoT devices [145], large-scale commercial systems used in areas such as e-commerce and finance [167], and operating systems [2]; and network systems refer to the diverse domains associated with the establishment, operation, and administration of computer networks, enabling efficient communication and data exchange between devices. This includes traditional network infrastructure [17, 41, 162], as well as emerging architectures like software-defined networking [144] and wireless sensor networks [8]. Embedded systems often have constraints such as a large number of connected devices, massive data exchange, high-speed network topology changes, and heterogeneous devices. Meanwhile, network systems like SDNs and WSNs usually operate in dynamic environments where network conditions can change rapidly, requiring adaptive performance models to account for fluctuations in traffic, topology, and interference. To bridge the gaps for both domains, several primary studies opt to leverage domain knowledge in incorporating environmental information and domain-specific features to increase the effectiveness in configuration performance modeling [8, 79, 145].

🔍 ***Finding 10:*** *Most commonly, the primary studies on deep configuration performance learning evaluate one configurable software system only (43 out of 99 studies). Furthermore, among the various domains explored, distributed computing emerges as the most popular domain for subject systems, utilized in 42 studies.*

6.3.1 Good Practice. Another positive observation is that 56 studies have considered multiple subject systems in their experiments. This is significant because evaluating the proposed model on software from different domains, scales, and performance indicators demonstrates its generalizability and robustness [55, 64, 93, 102, 147].

Table 19. Distribution of the model application purpose.

Category	# Studies	References
Prediction	65	[96], [95], [184], [125], [138], [47], [141], [100], [195], [122], [153], [19], [81], [40], [116], [86], [121], [115], [145], [160], [168], [41], [2], [73], [120], [80], [147], [172], [134], [39], [162], [42], [61], [146], [123], [91], [78], [128], [87], [110], [119], [56], [192], [52], [149], [93], [57], [46], [112], [161], [38], [64], [18], [53], [85], [144], [17], [7], [58], [183], [79], [189], [107], [174], [103]
Tuning	22	[49], [44], [45], [54], [105], [8], [108], [117], [12], [118], [169], [43], [48], [26], [191], [127], [158], [109], [15], [98], [177], [164]
Testing	6	[167], [140], [22], [196], [186], [175]
Scheduling	5	[182], [132], [13], [106], [14]
Development	1	[163]

Moreover, it is encouraging to see that the primary studies have examined a variety of domains with various challenges, and have leveraged domain-specific knowledge to address the challenges. This diversity in domain coverage enhances the credibility and applicability of the findings across various contexts, providing broader insights for developers and practitioners.

6.3.2 Bad Smell. It is worth noting that a significant proportion of studies (43% of 99) evaluate their models on only one software, which can be considered a potential threat to the validity of the results and findings of the study. Indeed, software systems of different characteristics can exhibit unique performance behavior due to factors such as architecture, complexity, and usage patterns. Therefore, it is important for future studies to evaluate their models on multiple software systems, spanning various domains, scales, and performance indicators, to ensure the robustness and learning ability of the proposed performance models.

Actionable Suggestion 14: *It is crucial for future research to include multiple software systems in their evaluations, covering a diverse range of domains, scales, and performance indicators.*
Actionable Suggestion 15: *Incorporate domain-specific knowledge into the model development process to tackle unique challenges of each domain, thereby enhancing the model's adaptability and performance.*

7 RQ4: HOW TO EXPLOIT THE DEEP CONFIGURATION PERFORMANCE MODEL?

The successful application of deep learning models trained on configuration data relies on considering various factors, including application purposes, adaptability to dynamic software running environments, and the availability of materials for reproducibility. In this section, we study those factors involved in the exploitation and dissemination of the deep configuration performance models.

7.1 Using the Model

A deep configuration performance model, once built, can be used in different stages of the software engineering practices. Table 19 shows that out of the 99 papers surveyed, 65 studies emphasize the domain of **general prediction** where the key has been ensuring the accuracy and reliability of the deep configuration performance model, without focusing on any particular domain in which the model can be used.

22 studies apply the deep learning models for **performance tuning**, or performance optimization, which aims at optimizing the performance by finding the optimal configuration. It involves tasks such as configuration auto-tuning [44, 45, 49, 49, 54], where software configurations are

automatically adjusted to fit performance requirements, and self-adaptive systems which dynamically adapt to varying conditions to achieve optimal performance [26]. In such a context, the deep configuration performance model directly serves as the surrogate for the tuning algorithm.

Performance testing tasks are considered by six studies. They involve assessing the system's behavior for specific configurations or under different environments to identify potential bottlenecks or performance issues. For example, configuration bug prediction seeks to identify potential bugs or issues that are caused by incorrect configuration, leading to performance degradation [186, 196]. Vulnerabilities detection focuses on identifying security vulnerabilities that could affect system performance or compromise its functionality [175].

Additionally, tasks on **scheduling** have been addressed by five primary studies, which seek to allocate the available resources, such as CPU, memory, or network bandwidth, in an optimal manner to maximize system performance [132, 182]. The unique property is that there are often low-level metrics, such as the CPU cycle or cache rate, that are considered together with the performance attributes. Notably, while both resource scheduling and performance tuning involve optimization processes, they mainly differ in the following points:

- Performance tuning aims to directly optimize the software configuration for enhancing the performance, while resource scheduling focuses on selecting the most appropriate resources for a given sequence of workload/task [14]. As a result, the latter involves an ordinal constraint while the former typically does not have such.
- Performance optimization can refer to the process of improving system performance at runtime (although offline scenarios are also possible) [24], while resource scheduling mainly occurs before the execution of the software.

Meanwhile, one study applies performance models for **development** [163], involving designing and implementing the best set of configurations for developing the architecture of deep learning models or software.

Finding 11: *The applications of deep configuration performance models are mainly for fulfilling five purposes, in which the most common one is the general prediction for performance analysis, as evidenced by 65 studies.*

7.1.1 *Good Practice.*

The best practice in the model applications is their widespread adoption across various domains and for different tasks. This diversity is critical as it showcases the potential of performance models in addressing diverse challenges, and provides practical insights for researchers in different fields [137].

7.1.2 *Bad Smell.*

A potential concern is that 65 studies only consider pure prediction tasks, while these studies contribute valuable insights into performance prediction, it raises the question of whether these models can effectively be applied to other tasks beyond prediction, in other words, the applicability and generalizability of performance models across different domains and tasks remain unclear, especially when the usefulness of the model is merely measured by accuracy metrics.

Actionable Suggestion 16: *Future studies should explore the applicability of performance models beyond prediction to ensure their relevance across a broader range of tasks.*

7.2 Considering Multiple Runtime Environments

Configurable software systems would inevitably run under multiple, and potentially changing environments. For example, a database system may experience both read-heavy and write-heavy workload [75]. Similarly, the hardware between the testing and production infrastructure might be drastically different [16, 99], especially during the modern DevOps era. It is therefore natural to understand how such a fact has been taken into account for deep configuration performance learning.

As in Figure 14, our survey results indicate that, out of the 99 papers surveyed, 64 studies do not explicitly consider the challenges posed by multiple and dynamic software running environments in their application and design of performance models [54, 57, 64, 80, 121, 144]. In other words, they rely on using the data from a **single** environment to build the configuration performance models and predict therein. This suggests that a considerable portion of the research in deep learning-based configuration performance learning has focused primarily on static or controlled settings, without accounting for the real-world complexities of dynamic, multiple environments. Nevertheless, among the 35 studies that acknowledge and consider dynamic environments in their application of performance models, 23 studies exclusively focus on exploring the impact of different **workloads** on configuration performance [39, 40, 85, 122, 158, 182]. Additionally, eight studies concentrate solely on investigating the influence of different **hardware** settings [19, 49, 61], and one study specifically examines the impact of different **software versions** [147]. Lastly, three studies research the implications of **mixed environment** types, i.e., one considers three workloads combined with two hardware settings [123], another considers 12 workloads combined with two hardware settings [45], and a recent work comprehensively addresses a total of 16 hardware, 26 workloads and 13 versions for nine software systems [58].

Finding 12: *More than half of the 99 primary studies (65%) do not consider multiple and dynamic environments. Among others, workload stands out as the most commonly considered factor (23%).*

7.2.1 Good Practice. The most encouraging trend is that 35 out of 99 studies have actively considered multiple and dynamic environments, demonstrating the robustness of models. Acknowledging that software systems run under multiple and potentially varying environments is an important property as the behavior of software could be completely different across the environments [25, 126, 136]. By accounting for dynamic workloads, versions, and hardware in real-world scenarios, these studies contribute to a more comprehensive understanding of how performance models can be applied in practical settings.

7.2.2 Bad Smell. Another bad smell is that 65% of 99 primary studies have solely focused on datasets from a single environment without considering dynamic factors that can impact model performance, which means there is a risk of overestimating the model's capabilities or encountering unexpected issues when applied in real-world scenarios [126]. Therefore, it is essential for future research to address this limitation by actively incorporating dynamic environments into their evaluation processes.

Actionable Suggestion 17: *Account for dynamic workloads, versions, and hardware to gain a comprehensive understanding of model performance in practical settings.*

7.3 Publishing Source Code and Data

The availability of public repositories plays an essential role in promoting replicability, transparency, and development in research. This is particularly crucial for practical research fields such as configuration performance learning. The results of our survey in Figure 15 reveal that a significant number of studies do not provide an open-access repository containing the necessary resources for

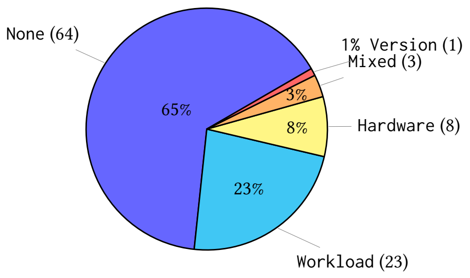


Fig. 14. Distribution of the environment types.

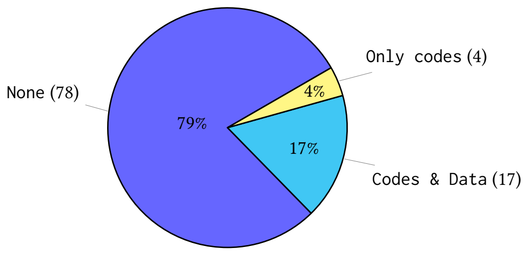


Fig. 15. Distribution of the availability of artifacts.

replication. In particular, out of the 99 papers surveyed, a total of 78 studies **do not make their code, data, or experiment results publicly available** [140, 167, 175, 186, 196]. This indicates a lack of emphasis on open science practices and hinders the ability of other researchers and practitioners to validate and build upon the findings. On the other hand, 21 studies offer an open-access repository, of which 17 provide **both source code and datasets** [12, 42, 54, 134, 140], and four of them **only provide source codes** [22, 169, 175, 191].

Finding 13: *Only 21% (21 out of 99) of the primary studies provide public repositories. An even lower proportion of them, i.e., 17%, make both source code and datasets publicly available.*

7.3.1 Good Practice. 21 out of the total studies have embraced the principles of open science by providing a public repository that includes source codes and datasets. This is a good practice as it allows researchers to replicate or reproduce the experiment results, build further research based on the existing knowledge, and collaborate with each other [111].

7.3.2 Bad Smell. It is concerning to note that a significant number of studies (79%) still missed the opportunity to promote open science by not providing public repositories. This is a limitation as it restricts the replicability, reproducibility, and development of the community, which could be addressed by future studies. Further, among the 21 studies that offer an open-access repository, only 17 of them provide both source codes and datasets.

One possible explanation for this bad smell is that there is a lack of a universal standard, guidelines, or best practices for using and sharing source codes and data, which makes it expensive to construct and maintain a public repository. Additionally, a lack of awareness or understanding regarding the benefits associated with open science may also contribute to the limited availability of public repositories.

Actionable Suggestion 18: *Future studies should aim to establish and follow universal standards for sharing resources and raise awareness about the benefits of open science.*

8 FUTURE RESEARCH OPPORTUNITIES

Our survey has revealed several key knowledge gaps and promising directions that are worth further investigation in future studies, which we elaborate on below.

8.1 Model-Based Sampling for Deep Configuration Performance Learning

From RQ1 (Section 4.3), we see that while several sampling strategies have been employed in the studies reviewed, it is evident that a significant majority of them rely on random sampling, which highlights a significant opportunity for future research to explore and develop more effective sampling strategies. Random sampling, while straightforward, may not always yield the most informative or representative samples within the configuration space [76, 83, 148]. By exploring alternative sampling techniques that select samples based on specific criteria or heuristics, researchers can potentially improve the efficiency and effectiveness of the models [136]. This is especially important for deep learning configuration performance since it is known that the deep learning model is sensitive to the quality of training data [76, 83, 136, 148].

Among others, addressing this gap by investigating model-based sampling methods that can provide an explicit acquisition function to reliably and accurately guide the sampling process is a promising future direction. For instance, Sequential Model-Based Optimization (SMBO) [169], such as Bayesian Optimization, effectively explores the sample space by quantifying the uncertainty of the samples to guide the search towards potentially optimal regions [4, 71, 154]. In particular, using a surrogate model, such as a Gaussian process, to sequentially select new samples based on the current model's predictions and uncertainty estimates, would enable efficient exploration of the sample space. However, to the best of our knowledge, there has been little work that investigates this thread of research.

8.2 Explainable Deep Configuration Performance Learning.

Through RQ2 (Section 5.2), we reveal that various types of deep learning models have been used to learn configuration performance, however, almost all of the proposed deep learning models lack explainability. Unlike analytical performance models, which enable researchers to analyze the importance of options and the interactions between them [165], deep learning models learn the configuration performance in a black-box manner, which could be harmful to the reliability of the results.

In contrast, certain statistical machine learning performance models are naturally interpretable. For instance, `SPLConqueror` [150] propose SPLConqueror leverages a linear model to learn configuration performance, enabling one to quantify the influence of each configuration; Gong and Chen [56, 57], Guo et al. [62] have examined Random Forest (RF) and Classification and Regression Tree (CART) for performance learning, highlighting the importance of options and the decision boundaries to cluster the configurations. Several studies have explored Bayesian theory to analyze the uncertainty of predictions that aid the explainability [3, 71, 163, 186].

Yet, addressing the above is challenging for deep learning model due to their nonlinear and high-dimensional representation nature. Luckily, explainable deep learning has been studied in various domains, such as pattern recognition tasks including medical diagnosis, face recognition, and self-driving cars [9], including those model-agnostic ones like Local Interpretable Model-agnostic Explanation (LIME) [143] and SHapley Additive exPlanations (SHAP) [114]. We can directly leverage the advance of the current explainable extension of deep learning, and specialize them in the context of configuration performance. However, achieving explainable deep configuration performance models faces several challenges, e.g.,

- How to quantify the internal elements of a deep learning model, e.g., activation function, with respect to their contributions according to the type and characteristics of the configuration options.
- How to equip a deep learning model with an intrinsic method that produces explanations on the configuration performance alongside its outputs.

- How to visualize the configuration options and their contributions toward the final prediction with a deep learning model.

Pushing the research toward addressing any of the above would be of immediate interest and benefit to the field.

8.3 Configuration Performance Modeling with Few-Shot Deep Learning

Deep neural networks for learning configuration performance often require a substantial amount of labeled data, which can be challenging and time-consuming to obtain, as the availability of labeled data is often limited due to factors such as the complexity of software systems, the cost of performance measurements, and the lack of expert knowledge.

A promising direction of most efficient deep learning approaches is to complement it with few-shot learning, which aims at training to quickly adapt and generalize to new tasks or classes with only a few labeled configuration samples or even a single configuration, e.g., siamese networks, memory-augmented networks, transfer learning, meta-learning, and augmentation strategies [82, 173]. For example, augmentation techniques like Generative Adversarial Networks have been used in three studies to generate additional training samples to augment the available configuration dataset [12, 105, 149], and siamese neural networks are able to differentiate between pairs of examples, making them effective for few-shot learning, which have been successfully used for image processing tasks [63].

Despite the above, few-shot deep learning still imposes several challenges for learning configuration data due to the complexity and specialty of configurable software systems. For example, the selected most representative data samples are difficult to quantify, obscuring effective generalizability in the few-shot deep learning model. Therefore, further research is needed to unlock the full potential of few-shot deep learning for configuration performance modeling by enhancing robustness, generalizability, and transferability.

8.4 Interactive Deep Configuration Performance Learning.

While deep learning models have demonstrated effectiveness in configuration performance learning, they come with inherent challenges, notably the substantial training overhead when compared to other machine learning models. The resource-intensive nature of training deep learning models poses limitations on the adaptability and reliability, especially when quick adaptation to newly measured data or real-time updates is required. This issue underscores the urgent need for innovative solutions that can efficiently update and optimize deep learning models, paving the way for more updated and reliable systems.

To address these challenges, interactive deep learning techniques have emerged as a powerful solution, which enables continuous learning and adaptation optionally with the support of humans, overcoming the limitations associated with the static nature of traditional deep learning models and fully benefiting from the domain expertise. By incorporating real-time updates and leveraging user feedback, interactive deep learning ensures that models remain accurate and up-to-date using data filtered by human knowledge. This adaptability is particularly crucial in applications where the underlying data distribution may change over time as the new measurements become available [26, 118]. Despite being a promising direction related to the human-in-the-loop techniques for configuration performance modeling, interactive updates of deep configuration performance models are currently under-explored.

Furthermore, an emerging technique in interactive deep learning is the use of pre-trained large language models (LLMs), which have demonstrated significant utility in numerous software engineering tasks, including code generation, bug detection, and requirement analysis. These models, trained on vast corpora of text, can provide powerful understanding and generation capabilities, making them potentially useful for software configuration performance learning. Potential usage scenarios for LLMs in this context include automatically generating optimal configuration settings based on system requirements and historical performance data, predicting the impact of configuration changes on system performance by reading the configuration setup files, and diagnosing performance bottlenecks by analyzing logs.

However, the adoption of pre-trained LLMs in this domain is prevented by several factors and challenges:

- Configuration performance learning is primarily based on relatively simple tubular data, while LLM is designed to handle complex modalities, such as texts or code. As such, how to better exploit the strengths of LLM is still unclear in the field. Of course, a potential way is to add extra modalities such as configuration code, documentation, and logs, but in that case, information fusion becomes a key challenge.
- There is a need to design sophisticated prompts to fine-tune these models to the specific problem of learning configuration performance which is not a straightforward task.
- Their generalizability to configurable software systems in diverse domains and environments such as database software vs. video encoding software and distributed computing vs. local computing, is still unknown.
- The computational overhead associated with deploying and obtaining inference from LLMs could be the barrier to adopting them in real-time performance prediction scenarios.

Nevertheless, during our search, we did find two studies that leveraged LLMs for configuration tuning or performance modeling. Firstly, by leveraging the natural language processing capabilities of Generative Pre-trained Transformer (GPT), Lao et al. [97] propose GPTune to analyze, filter, summarize, and check the consistency of the domain knowledge extracted from various sources, ultimately enhancing the tuning process and optimizing the DBMS configuration effectively. Yet, this study mainly concentrates on tuning the configuration and does not involve the process of modeling, and therefore lies outside the scope of this survey. Secondly, in the work by Nichols et al. [133], an LLM is fine-tuned on a curated dataset containing HPC and scientific codes, and is employed to predict the relative performance impact of changes made to the source code. Specifically, given two versions of a code, the model can analyze and predict which version is likely to perform better in terms of execution time. Nonetheless, they were filtered out in the stage when applying the exclusion criteria, since they had not been published in any peer-reviewed public venue by the end of our search procedures (till May 2024).

As such, addressing these challenges is non-trivial and requires further research, which has not yet been achieved for deep configuration performance learning during the period covered by our survey. However, we anticipate that those challenges will be tackled in the community and we foresee a future where more research about LLM adoption will gradually appear.

8.5 Deep Configuration Performance Learning under Multiple Dynamic Environments

From RQ4, it is evident that a majority, specifically 66%, do not address the dynamic and varying nature of software running environments, which may significantly limit the effectiveness and generalizability of deep configuration performance models. In particular, Mühlbauer et al. [126] systematically analyzes the influence of workloads on configurable software systems and discloses that varying the workload does not only affect the performance value but also affects the relationships between the configurations and performance.

This observation highlights the need to address this gap by developing deep learning methodologies that can effectively handle configuration data across multiple environments. Little work has tackled the above by using, e.g., multi-task learning [4, 135, 155, 179], which learns the common representations between multiple tasks simultaneously; transfer learning [55, 80, 93, 102, 147], which seek to leverage the common information gained from related environments, enabling models to adapt and generalize to the target environments; or meta-learning [20, 70, 166, 181, 193], which pretrain a set of model parameters via meta-training and can adapt quickly or unseen tasks or environments. Particularly, a recent work [58] proposes a sequential meta-learning framework, namely `SeMPL`, that trains a meta-model with multiple meta-environments in an optimal order such that the pre-trained model can generalize to new environments well. Yet, most of the existing work only assumes homogeneous configuration performance learning, i.e., for different environments, the configuration options need to remain the same. Further, those works do not cater to online learning [23, 28], where the data is continually used to update the model as it becomes available. As a result, deep learning models that can learn from multiple environments and are capable of self-updating at runtime are in high demand for configuration performance learning [24, 27, 30].

9 THREATS TO VALIDITY

When conducting a systematic literature review, it is important to consider potential threats to validity that could affect the reliability and generalizability of the findings. We now discuss the threats to the validity related to this work.

9.1 Sampling Bias

Sampling bias refers to the potential threats of the results due to the selection of studies. In this survey, sampling bias could arise from the selection of primary papers from indexing services or the inclusion and exclusion of certain papers. To mitigate this threat, we have codified specific rules, procedures and criteria in the search protocol, which is based on the guidance provided by Kitchenham et al. [90]. Specifically, the sampling of studies is done by a comprehensive automatic search via six popular indexes in software engineering to cover a wide range of papers, then the application of domain knowledge to remove the redundant and irrelevant studies, and filtering them with the carefully crafted inclusion and exclusion criteria to further remove potential bias, as summarized in Figure 2.

9.2 Internal Threats

Internal threats to validity relate to issues within the study design or data analysis that could affect the accuracy and reliability of the results. In this survey, potential internal threats could include inconsistencies in data extraction, subjectivity in data analysis, or biased interpretation of the findings. To address these threats, we have followed a systematic and rigorous approach to data extraction and analysis. By clearly defining research questions, it ensures that the data extraction process is focused and consistent. Additionally, we have limited this by employing multiple reviewers and by conducting three iterations of independent paper reviews among the authors. Error checks and investigations were also conducted to correct any issues found during the search. Any discrepancy in the results was discussed until an agreement can be reached.

9.3 External Threats

External threats to validity are usually related to the generalizability and applicability of the findings beyond the specific context of the survey. In this survey, external threats could arise from the limited time range from 2013 to 2023, or the limited scope of the papers using deep learning for performance modeling. To mitigate these threats, we have searched 948 studies from six indexing services, we clearly defined the inclusion and exclusion criteria for the selection of papers, based on which we extracted 85 prominent primary studies for detailed analysis. By including a diverse range of studies from different indexing services, we provide a broader perspective on deep configuration performance learning.

10 CONCLUSION

In this work, we present a comprehensive survey on the increasingly popular topic of deep configuration performance learning, covering 99 prominent works from 1,206 studies found on six indexing services. We provide detailed taxonomy, together with discussions on the advantages, disadvantages, and best-suited scenarios of the various techniques involved, according to the key phases in building a deep configuration performance model, i.e., preparation, modeling, evaluation, and applications. Our results also reveal several good practices, bad smells, and actionable suggestions of the existing studies in the field.

More importantly, we highlight promising research opportunities for this particular research field, namely:

- model-based sampling methods for deep configuration performance learning.
- explainable deep configuration performance model.
- interactively learned deep configuration performance model.
- deep few-shot learning for modeling configuration performance.
- deep configuration performance learning under multiple and dynamic environments.

By highlighting the identified trends and future directions of deep learning for configuration performance modeling—we hope to inspire sustainable growth on this particular topic.

ACKNOWLEDGMENTS

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

REFERENCES

[1] [n.d.]. Dividable Configuration Performance Learning, Author = Jingzhi Gong and Tao Chen and Rami Bahsoon, Journal = IEEE Transactions on Software Engineering, Year = 2024. ([n. d.]).

[2] Mathieu Acher, Hugo Martin, Luc Lesoil, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Eddine Khelladi, Olivier Barais, and Juliana Alves Pereira. 2022. Feature subset selection for learning huge configuration spaces: the case of linux kernel size. In *SPLC '22: 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume A*, Alexander Felfernig, Lidia Fuentes, Jane Cleland-Huang, Wesley K. G. Assunção, Andreas A. Falkner, Maider Azanza, Miguel Á. Rodríguez Luaces, Megha Bhushan, Laura Semini, Xavier Devroey, Cláudia Maria Lima Werner, Christoph Seidl, Viet-Man Le, and José Miguel Horcas (Eds.). ACM, 85–96. https://doi.org/10.1145/3546932.3546997

[3] M. Agarwal, D. Singhvi, P. Malakar, and S. Byna. 2019. Active Learning-based Automatic Tuning and Prediction of Parallel I/O Performance. In *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*. 20–29.

[4] 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

[5] Mokhtar Z. Alaya, Simon Bussy, Stéphane Gaïffas, and Agathe Guilloux. 2019. Binarsity: a penalization for one-hot encoded features in linear supervised learning. *J. Mach. Learn. Res.* 20 (2019), 118:1–118:34. http://jmlr.org/papers/v20/17-170.html

[6] Ethem Alpaydin. 2004. *Introduction to machine learning.*

[7] Kevin Assogba, Eduardo Lima, M. Mustafa Rafique, and Minseok Kwon. 2023. PredictDDL: Reusable Workload Performance Prediction for Distributed Deep Learning. In *IEEE International Conference on Cluster Computing, CLUSTER 2023, Santa Fe, NM, USA, October 31 - Nov. 3, 2023.* IEEE, 13–24. https://doi.org/10.1109/CLUSTER52292.2023.00009

[8] Muhammad Ateeq, Muhammad Khalil Afzal, Sheraz Anjum, and Byung-Seo Kim. 2022. Cognitive quality of service predictions in multi-node wireless sensor networks. *Comput. Commun.* 193 (2022), 155–167. https://doi.org/10.1016/j.comcom.2022.06.042

[9] Xiao Bai, Xiang Wang, Xianglong Liu, Qiang Liu, Jingkuan Song, Nicu Sebe, and Been Kim. 2021. Explainable deep learning for efficient and robust pattern recognition: A survey of recent developments. *Pattern Recognit.* 120 (2021), 108102. https://doi.org/10.1016/J.PATCOG.2021.108102

[10] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. 2004. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. Software Eng.* 30, 5 (2004), 295–310.

[11] Simonetta Balsamo, Vittoria De Nitto Persone, and Paola Inverardi. 2003. A review on queueing network models with finite capacity queues for software architectures performance prediction. *Perform. Evaluation* 51, 2/4 (2003), 269–288.

[12] Liang Bao, Xin Liu, Fangzheng Wang, and Baoyin Fang. 2019. ACTGAN: Automatic Configuration Tuning for Software Systems with Generative Adversarial Networks. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019.* IEEE, 465–476. https://doi.org/10.1109/ASE.2019.00051

[13] Jan-Harm Betting, Dimitrios Liakopoulos, Max Engelen, and Christos Strydis. 2023. Oikonomos: An Opportunistic, Deep-Learning, Resource-Recommendation System for Cloud HPC. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP).* IEEE, 188–196.

[14] Jan-Harm L. F. Betting, Chris I. De Zeeuw, and Christos Strydis. 2023. Oikonomos-II: A Reinforcement-Learning, Resource-Recommendation System for Cloud HPC. In *30th IEEE International Conference on High Performance Computing, Data, and Analytics, HiPC 2023, Goa, India, December 18-21, 2023.* IEEE, 266–276. https://doi.org/10.1109/HIPC58850.2023.00044

[15] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 1–23.

[16] 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. *CoRR* abs/1508.04752 (2015). arXiv:1508.04752 http://arxiv.org/abs/1508.04752

[17] Qi Cao, Man-On Pun, and Yi Chen. 2022. Deep Learning in Network-Level Performance Prediction Using Cross-Layer Information. *IEEE Trans. Netw. Sci. Eng.* 9, 4 (2022), 2364–2377. https://doi.org/10.1109/TNSE.2022.3163274

[18] Mehmet Cengiz, Matthew Forshaw, Amir Atapour-Abarghouei, and Andrew Stephen McGough. 2023. Predicting the Performance of a Computing System with Deep Networks. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering, ICPE 2023, Coimbra, Portugal, April 15-19, 2023*, Marco Vieira, Valeria Cardellini, Antinisca Di Marco, and Petr Tuma (Eds.). ACM, 91–98. https://doi.org/10.1145/3578244.3583731

[19] Yuji Chai, Devashree Tripathy, Chuteng Zhou, Dibakar Gope, Igor Fedorov, Ramon Matas, David Brooks, Gu-Yeon Wei, and Paul Whatmough. 2023. PerfSAGE: Generalized Inference Performance Predictor for Arbitrary Deep Learning Models on Edge Devices. *arXiv preprint arXiv:2301.10999* (2023).

[20] 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

[21] 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. https://doi.org/10.1109/TSE.2024.3388910

[22] Simin Chen, Mirazul Haque, Cong Liu, and Wei Yang. 2022. DeepPerform: An Efficient Approach for Performance Testing of Resource-Constrained Neural Networks. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022.* ACM, 31:1–31:13. https://doi.org/10.1145/3551349.3561158

[23] 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.* 157–168.

[24] 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. https://doi.org/10.1109/SANER53432.2022.00022

[25] 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

[26] 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.

[27] 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. https://doi.org/10.1109/TSC.2015.2499770

[28] Tao Chen, Rami Bahsoon, Shuo Wang, and Xin Yao. 2018. To Adapt or Not to Adapt?: Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018*, Katinka Wolter, William J. Knottenbelt, André van Hoorn, and Manoj Nambiar (Eds.). ACM, 48–55. https://doi.org/10.1145/3184407.3184413

[29] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Comput. Surv.* 51, 3 (2018), 61:1–61:40. https://doi.org/10.1145/3190507

[30] Tao Chen, Rami Bahsoon, and Xin Yao. 2020. Synergizing Domain Expertise With Self-Awareness in Software Systems: A Patternized Architecture Guideline. *Proc. IEEE* 108, 7 (2020), 1094–1126. https://doi.org/10.1109/JPROC.2020.2985293

[31] 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.

[32] 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

[33] 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. https://doi.org/10.1145/3571853

[34] 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. https://doi.org/10.1145/3514233

[35] 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

[36] Tao Chen, Miqing Li, and Xin Yao. 2018. On the effects of seeding strategies: a case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, Hernán E. Aguirre and Keiki Takadama (Eds.). ACM, 1419–1426. https://doi.org/10.1145/3205455.3205513

[37] Tao Chen, Miqing Li, and Xin Yao. 2019. Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition. *Inf. Softw. Technol.* 114 (2019), 155–175. https://doi.org/10.1016/J.INFSOF.2019.05.013

[38] 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. https://doi.org/10.1145/3528100

[39] Yuxia Cheng, Wenzhi Chen, Zonghui Wang, and Yang Xiang. 2017. Precise contention-aware performance prediction on virtualized multicore system. *J. Syst. Archit.* 72 (2017), 42–50.

[40] Soumia Chokri, Sohaib Baroud, Safa Belhaous, Mohamed Youssfi, and Mohammed Mestari. 2022. Performance prediction of parallel applications using artificial neuronal network and graph representation. *Concurrency and Computation: Practice and Experience* (2022), e7514.

[41] Murtaza Cicioglu and Ali Çalhan. 2022. Handover management in software-defined 5G small cell networks via long short-term memory. *Concurr. Comput. Pract. Exp.* 34, 10 (2022). https://doi.org/10.1002/cpe.6832

[42] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. 2015. Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. 145–156.

[43] Kuo-Teng Ding, Hui-Shan Chen, Yi-Lun Pan, Hung-Hsin Chen, Yuan-Ching Lin, and Shih-Hao Hung. 2021. Portable Fast Platform-Aware Neural Architecture Search for Edge/Mobile Computing AI Applications. *ICSEA 2021* (2021), 108.

[44] Vanderson Martins do Rosario, Anderson Faustino da Silva, André Felipe Zanella, Otávio Oliveira Napoli, and Edson Borin. 2023. Fast selection of compiler optimizations using performance prediction with graph neural networks. *Concurr. Comput. Pract. Exp.* 35, 17 (2023). https://doi.org/10.1002/cpe.6869

[45] Hui Dou, Yilun Wang, Yiwen Zhang, and Pengfei Chen. 2022. DeepCAT: A Cost-Efficient Online Configuration Auto-Tuning Approach for Big Data Frameworks. In *Proceedings of the 51st International Conference on Parallel Processing, ICPP 2022, Bordeaux, France, 29 August 2022 - 1 September 2022.* ACM, 67:1–67:11. https://doi.org/10.1145/3545008.3545018

[46] Guangyu Du, Hong He, and Fanxin Meng. 2013. Performance modeling based on artificial neural network in virtualized environments. *Sensors & Transducers* 153, 6 (2013), 37.

[47] Ta Nguyen Binh Duong, Jinghui Zhong, Wentong Cai, Zengxiang Li, and Suiping Zhou. 2016. RA2: Predicting Simulation Execution Time for Cloud-Based Design Space Explorations. In *20th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2016, London, United Kingdom, September 21-23, 2016.*

IEEE Computer Society, 120–127. https://doi.org/10.1109/DS-RT.2016.9

[48] Thomas L. Falch and Anne C. Elster. 2015. Machine Learning Based Auto-Tuning for Enhanced OpenCL Performance Portability. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015, Hyderabad, India, May 25-29, 2015*. IEEE Computer Society, 1231–1240. https://doi.org/10.1109/IPDPSW.2015.85

[49] Thomas L. Falch and Anne C. Elster. 2017. Machine learning-based auto-tuning for enhanced performance portability of OpenCL applications. *Concurrency and Computation: Practice and Experience* 29, 8 (2017).

[50] Jesus Flores-Contreras, Hector A. Duran-Limon, Arturo Chavoya, and Sergio H. Almanza-Ruiz. 2021. Performance prediction of parallel applications: a systematic literature review. *J. Supercomput.* 77, 4 (2021), 4014–4055. https://doi.org/10.1007/S11227-020-03417-5

[51] Markus Frank, Marcus Hilbrich, Sebastian Lehrig, and Steffen Becker. 2017. Parallelization, Modeling, and Performance Prediction in the Multi-/Many Core Area: A Systematic Literature Review. In *2017 IEEE 7th International Symposium on Cloud and Service Computing, SC^2 2017, Kanazawa, Japan, November 22-25, 2017*. IEEE, 48–55.

[52] Silvery Fu, Saurabh Gupta, Radhika Mittal, and Sylvia Ratnasamy. 2021. On the Use of ML for Blackbox System Performance Prediction. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, James Mickens and Renata Teixeira (Eds.). USENIX Association, 763–784. https://www.usenix.org/conference/nsdi21/presentation/fu

[53] Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2023. Runtime Performance Prediction for Deep Learning Models with Graph Neural Network. In *45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 368–380. https://doi.org/10.1109/ICSE-SEIP58684.2023.00039

[54] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2019. Automated search for configurations of convolutional neural network architectures. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*, Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi (Eds.). ACM, 21:1–21:12. https://doi.org/10.1145/3336294.3336306

[55] Devarshi Ghoshal, Kesheng Wu, Eric Pouyoul, and Erich Strohmaier. 2019. Analysis and Prediction of Data Transfer Throughput for Data-Intensive Workloads. In *2019 IEEE International Conference on Big Data (Big Data)*. 3648–3657.

[56] 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

[57] 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

[58] Jingzhi Gong and Tao Chen. 2024. Predicting Configuration Performance in Multiple Environments with Sequential Meta-Learning. *FSE'24: Proceedings of the ACM on Software Engineering (PACMSE)* 1, FSE (2024). https://doi.org/10.1145/3643743

[59] Paul Goodwin and Richard Lawton. 1999. On the asymmetry of the symmetric MAPE. *International journal of forecasting* 15, 4 (1999), 405–408.

[60] Vincenzo Grassi, Lorenzo Donatiello, and Salvatore Tucci. 1992. On the Optimal Checkpointing of Critical Tasks and Transaction-Oriented Systems. *IEEE Trans. Software Eng.* 18, 1 (1992), 72–77. https://doi.org/10.1109/32.120317

[61] Johannes Grohmann, Patrick K. Nicholson, Jesús Omana Iglesias, Samuel Kounev, and Diego Lugones. 2019. Monitorless: Predicting Performance Degradation in Cloud Applications with Machine Learning. In *Proceedings of the 20th International Middleware Conference, Middleware 2019, Davis, CA, USA, December 9-13, 2019*. ACM, 149–162. https://doi.org/10.1145/3361525.3361543

[62] 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).

[63] Yao Guo, Junyang Song, Yu Guo, Rongfang Bie, Libin Jiao, Anton Umek, and Anton Kos. 2022. RtSNN: Building a Real-time Siamese Neural Networks Model for Few-shot Action Recognition. In *8th International Conference on Big Data Computing and Communications, BigCom 2022, Xiamen, China, August 6-7, 2022*. IEEE, 243–250. https://doi.org/10.1109/BIGCOM57025.2022.00038

[64] 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.

[65] Huong Ha and Hongyu Zhang. 2019. Performance-Influence Model for Highly Configurable Software with Fourier Learning and Lasso Regression. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME*.

470–480.

[66] 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*. 23:1–23:10.

[67] Xue Han, Tingting Yu, and Gongjun Yan. 2023. A systematic mapping study of software performance research. *Softw. Pract. Exp.* 53, 5 (2023), 1249–1270. https://doi.org/10.1002/SPE.3185

[68] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. 2011. Starfish: A Self-tuning System for Big Data Analytics. In *5th Biennial Conference on Innovative Data Systems Research*. 261–272.

[69] Max Hort, Maria Kechagia, Federica Sarro, and Mark Harman. 2022. A Survey of Performance Optimization for Mobile Applications. *IEEE Trans. Software Eng.* 48, 8 (2022), 2879–2904. https://doi.org/10.1109/TSE.2021.3071193

[70] Timothy M. Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. 2022. Meta-Learning in Neural Networks: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 9 (2022), 5149–5169. https://doi.org/10.1109/TPAMI.2021.3079209

[71] Chin-Jung Hsu, Vivek Nair, Vincent W. Freeh, and Tim Menzies. 2018. Arrow: Low-Level Augmented Bayesian Optimization for Finding the Best Cloud VM. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS*. 660–670.

[72] Jianglin Huang, Yan-Fu Li, and Min Xie. 2015. An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Inf. Softw. Technol.* 67 (2015), 108–127. https://doi.org/10.1016/J.INFSOF.2015.07.004

[73] Florin Isaila, Prasanna Balaprakash, Stefan M. Wild, Dries Kimpe, Robert Latham, Robert B. Ross, and Paul D. Hovland. 2015. Collective I/O Tuning Using Analytical and Machine Learning Models. In *2015 IEEE International Conference on Cluster Computing, CLUSTER*. 128–137.

[74] Pooyan Jamshidi and Giuliano Casale. 2016. An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems. In *24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2016, London, United Kingdom, September 19-21, 2016*. IEEE Computer Society, 39–48.

[75] 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. https://doi.org/10.1109/ASE.2017.8115661

[76] 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*. 71–82.

[77] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. 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.

[78] Shuo Ji, Yinliang Zhao, and Yuxiang Li. 2022. Performance prediction for distributed graph computing. *Concurr. Comput. Pract. Exp.* 34, 12 (2022). https://doi.org/10.1002/cpe.5961

[79] Weiwei Jiang, Haoyu Han, Miao He, and Weixi Gu. 2024. ML-based pre-deployment SDN performance prediction with neural network boosting regression. *Expert Systems with Applications* 241 (2024), 122774.

[80] Andreas Johnsson, Farnaz Moradi, and Rolf Stadler. 2019. Performance Prediction in Dynamic Clouds using Transfer Learning. In *IFIP/IEEE International Symposium on Integrated Network Management,IM*. 242–250.

[81] Ali Jooya, Nikitas Dimopoulos, and Amirali Baniasadi. 2019. Multiobjective GPU design space exploration optimization. *Microprocess. Microsystems* 69 (2019), 198–210. https://doi.org/10.1016/j.micpro.2019.06.001

[82] Suvarna Kishorkumar Kadam and Vinay G. Vaidya. 2018. Review and Analysis of Zero, One and Few Shot Learning Approaches. In *Intelligent Systems Design and Applications - 18th International Conference on Intelligent Systems Design and Applications, ISDA 2018, Vellore, India, December 6-8, 2018, Volume 1 (Advances in Intelligent Systems and Computing, Vol. 940)*, Ajith Abraham, Aswani Kumar Cherukuri, Patricia Melin, and Niketa Gandhi (Eds.). Springer, 100–112. https://doi.org/10.1007/978-3-030-16657-1_10

[83] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-based sampling of software configuration spaces. In *the 41st International Conference on Software Engineering*. 1084–1094.

[84] Yong-Bin Kang, Shonali Krishnaswamy, Wudhichart Sawangphol, Lianli Gao, and Yuan-Fang Li. 2020. Understanding and improving ontology reasoning efficiency through learning and ranking. *Inf. Syst.* 87 (2020).

[85] Flora Karniavoura and Kostas Magoutis. 2017. A Measurement-Based Approach to Performance Prediction in NoSQL Systems. In *25th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and*

Telecommunication Systems, MASCOTS. 255–262.

[86] Byeong Soo Kim and Tag Gon Kim. 2017. Cooperation between data modeling and simulation modeling for performance analysis of Hadoop. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2017, Seattle, WA, USA, July 9-12, 2017.* IEEE, 1–7. https://doi.org/10.23919/SPECTS.2017.8046769

[87] Yeseong Kim, Pietro Mercati, Ankit More, Emily Shriver, and Tajana Rosing. 2017. P^4: Phase-based power/performance prediction of heterogeneous systems via neural networks. In *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD.* 683–690.

[88] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[89] Barbara Kitchenham and Stuart M. Charters. 2007. Guidelines for performing systematic literature reviews in software engineering.

[90] Barbara A. Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen G. Linkman. 2009. Systematic literature reviews in software engineering - A systematic literature review. *Inf. Softw. Technol.* 51, 1 (2009), 7–15.

[91] George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis, Spyridon V. Gogouvitis, and Theodora A. Varvarigou. 2014. Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms. *Future Gener. Comput. Syst.* 32 (2014), 27–40. https://doi.org/10.1016/j.future.2012.05.009

[92] 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

[93] Rajat Kumar, Amit Mankodi, Amit Bhatt, Bhaskar Chaudhury, and Aditya Amrutiya. 2020. Cross-Platform Performance Prediction with Transfer Learning using Machine Learning. In *11th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2020, Kharagpur, India, July 1-3, 2020.* IEEE, 1–7. https://doi.org/10.1109/ICCCNT49239.2020.9225281

[94] Satish Kumar, Tao Chen, Rami Bahsoon, and Rajkumar Buyya. 2020. DATESSO: self-adapting service composition with debt-aware two levels constraint reasoning. In *SEAMS '20: IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Seoul, Republic of Korea, 29 June - 3 July, 2020*, Shinichi Honiden, Elisabetta Di Nitto, and Radu Calinescu (Eds.). ACM, 96–107. https://doi.org/10.1145/3387939.3391604

[95] Indika Kumara, Mohamed Hameez Ariz, Mohan Baruwal Chhetri, Majid Mohammadi, Willem-Jan van den Heuvel, and Damian A. Tamburri. 2023. FOCloud: Feature Model Guided Performance Prediction and Explanation for Deployment Configurable Cloud Applications. *IEEE Trans. Serv. Comput.* 16, 1 (2023), 302–314. https://doi.org/10.1109/TSC.2022.3142853

[96] Shivam Kundan, Ourania Spantidi, and Iraklis Anagnostopoulos. 2021. Online frequency-based performance and power estimation for clustered multi-processor systems. *Comput. Electr. Eng.* 90 (2021), 106971. https://doi.org/10.1016/j.compeleceng.2021.106971

[97] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2023. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *CoRR* abs/2311.03157 (2023). https://doi.org/10.48550/ARXIV.2311.03157 arXiv:2311.03157

[98] Jieun Lee, Sangmin Seo, Jonghwan Choi, and Sanghyun Park. 2024. K2vTune: A workload-aware configuration tuning for RocksDB. *Inf. Process. Manag.* 61, 1 (2024), 103567. https://doi.org/10.1016/J.IPM.2023.103567

[99] 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

[100] Junnan Li, Zhihui Lu, Yu Tong, Jie Wu, Shalin Huang, Meikang Qiu, and Wei Du. 2019. A general AI-defined attention network for predicting CDN performance. *Future Gener. Comput. Syst.* 100 (2019), 759–769.

[101] 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. https://doi.org/10.1145/3324884.3416617

[102] 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. https://doi.org/10.1145/3377811.3380360

[103] Lingda Li, Santosh Pandey, Thomas Flynn, Hang Liu, Noel Wheeler, and Adolfy Hoisie. 2022. SimNet: Accurate and High-Performance Computer Architecture Simulation using Deep Learning. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2 (2022), 25:1–25:24. https://doi.org/10.1145/3530891

[104] 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

[105] Mingyu Li, Zhiqiang Liu, Xuanhua Shi, and Hai Jin. 2020. ATCS: Auto-Tuning Configurations of Big Data Frameworks Based on Generative Adversarial Nets. *IEEE Access* 8 (2020), 50485–50496. https://doi.org/10.1109/ACCESS.2020.2979812

[106] Wenzheng Li, Xiaoping Li, and Long Chen. 2024. Pattern learning for scheduling microservice workflow to cloud containers. *International Journal of Machine Learning and Cybernetics* (2024), 1–14.

[107] Yufei Li, Liang Bao, Kaipeng Huang, Chase Q. Wu, and Xinwei Li. 2024. RSFIN: A Rule Search-based Fuzzy Inference Network for performance prediction of configurable software systems. *J. Syst. Softw.* 209 (2024), 111913. https://doi.org/10.1016/J.JSS.2023.111913

[108] Yang Li, Zhenhua Han, Quanlu Zhang, Zhenhua Li, and Haisheng Tan. 2020. Automating Cloud Deployment for Deep Learning Inference of Real-time Online Services. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 1668–1677. https://doi.org/10.1109/INFOCOM41043.2020.9155267

[109] Yuhao Li and Benjamin C. Lee. 2022. Phronesis: Efficient Performance Modeling for High-dimensional Configuration Tuning. *ACM Trans. Archit. Code Optim.* 19, 4 (2022), 56:1–56:26. https://doi.org/10.1145/3546868

[110] Chieh-Jan Mike Liang, Zilin Fang, Yuqing Xie, Fan Yang, Zhao Lucis Li, Li Lyna Zhang, Mao Yang, and Lidong Zhou. 2023. On Modular Learning of Distributed Systems for Predicting End-to-End Latency. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, Mahesh Balakrishnan and Manya Ghobadi (Eds.). USENIX Association, 1081–1095. https://www.usenix.org/conference/nsdi23/presentation/liang-chieh-jan

[111] Chao Liu, Cuiyun Gao, Xin Xia, David Lo, John C. Grundy, and Xiaohu Yang. 2022. On the Reproducibility and Replicability of Deep Learning in Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 31, 1 (2022), 15:1–15:46. https://doi.org/10.1145/3477535

[112] Frank Liu, Narasinga Rao Miniskar, Dwaipayan Chakraborty, and Jeffrey S. Vetter. 2020. Deffe: a data-efficient framework for performance characterization in domain-specific computing. In *Proceedings of the 17th ACM International Conference on Computing Frontiers, CF 2020, Catania, Sicily, Italy, May 11-13, 2020*, Maurizio Palesi, Gianluca Palermo, Catherine Graves, and Eishi Arima (Eds.). ACM, 182–191. https://doi.org/10.1145/3387902.3392633

[113] Unai Lopez-Novoa, Alexander Mendiburu, and José Miguel-Alonso. 2015. A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing. *IEEE Trans. Parallel Distrib. Syst.* 26, 1 (2015), 272–281.

[114] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 4765–4774. https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[115] Thomas C. H. Lux, Layne T. Watson, Tyler H. Chang, Jon Bernard, Bo Li, Li Xu, Godmar Back, Ali Raza Butt, Kirk W. Cameron, and Yili Hong. 2018. Predictive modeling of I/O characteristics in high performance computing systems. In *Proceedings of the High Performance Computing Symposium, SpringSim (HPC) 2018, Baltimore, MD, USA, April 15-18, 2018*, Layne T. Watson, Masha Sosonkina, William I. Thacker, and Josef Weinbub (Eds.). ACM, 8:1–8:12. http://dl.acm.org/citation.cfm?id=3213077

[116] 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. https://doi.org/10.1145/3337821.3337922

[117] Ashraf Mahgoub, Paul Wood, Sachandhan Ganesh, Subrata Mitra, Wolfgang Gerlach, Travis Harrison, Folker Meyer, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. 2017. Rafiki: a middleware for parameter tuning of NoSQL datastores for dynamic metagenomics workloads. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, December 11 - 15, 2017*, K. R. Jayaram, Anshul Gandhi, Bettina Kemme, and Peter R. Pietzuch (Eds.). ACM, 28–40. https://doi.org/10.1145/3135974.3135991

[118] Hosein Mohammadi Makrani, Hossein Sayadi, Najmeh Nazari, Sai Manoj Pudukotai Dinakarrao, Avesta Sasan, Tinoosh Mohsenin, Setareh Rafatirad, and Houman Homayoun. 2021. Adaptive Performance Modeling of Data-intensive Workloads for Resource Provisioning in Virtualized Environment. *ACM Trans. Model. Perform. Evaluation Comput. Syst.* 5, 4 (2021), 18:1–18:24. https://doi.org/10.1145/3442696

[119] Preeti Malakar, Prasanna Balaprakash, Venkatram Vishwanath, Vitali A. Morozov, and Kalyan Kumaran. 2018. Benchmarking Machine Learning Methods for Performance Modeling of Scientific Applications. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, PMBS@SC 2018, Dallas, TX, USA, November 12, 2018*. IEEE, 33–44. https://doi.org/10.1109/PMBS.2018.8641686

[120] Muhammad Junaid Malik, Thomas Fahringer, and Radu Prodan. 2013. Execution time prediction for grid infrastructures based on runtime provenance data. In *Proceedings of WORKS 2013: 8th Workshop On Workflows in Support of Large-Scale Science, Held in conjunction with SC13, Denver, CO, USA, November 17, 2013*, Johan Montagnat and Ian J. Taylor (Eds.). ACM, 48–57. https://doi.org/10.1145/2534248.2534253

[121] Aniruddha Marathe, Rushil Anirudh, Nikhil Jain, Abhinav Bhatele, Jayaraman J. Thiagarajan, Bhavya Kailkhura, Jae-Seung Yeom, Barry Rountree, and Todd Gamblin. 2017. Performance modeling under resource constraints using deep transfer learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC*. 31:1–31:12.

[122] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (2019), 1733–1746. https://doi.org/10.14778/3342263.3342646

[123] Alexandre Maros, Fabricio Murai, Ana Paula Couto da Silva, Jussara M. Almeida, Marco Lattuada, Eugenio Gianniti, Marjan Hosseini, and Danilo Ardagna. 2019. Machine Learning for Performance Prediction of Spark Cloud Applications. In *12th IEEE International Conference on Cloud Computing, CLOUD*. 99–106.

[124] Brian P Mathews and Adamantios Diamantopoulos. 1994. Towards a taxonomy of forecast error measures a factor-comparative investigation of forecast error dimensions. *Journal of Forecasting* 13, 4 (1994), 409–416.

[125] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. 2022. Performance and Power Prediction for Concurrent Execution on GPUs. *ACM Trans. Archit. Code Optim.* 19, 3 (2022), 35:1–35:27. https://doi.org/10.1145/3522712

[126] 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.

[127] Rohyoung Myung and Sukyong Choi. 2021. Machine-Learning Based Memory Prediction Model for Data Parallel Workloads in Apache Spark. *Symmetry* 13, 4 (2021), 697.

[128] Farrukh Nadeem, Daniyal M. Alghazzawi, Abdulfattah S. Mashat, Khalid Faqeeh, and Abdullah Al-Malaise Al-Ghamdi. 2019. Using Machine Learning Ensemble Methods to Predict Execution Time of e-Science Workflows in Heterogeneous Distributed Systems. *IEEE Access* 7 (2019), 25138–25149. https://doi.org/10.1109/ACCESS.2019.2899985

[129] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, Johannes Fürnkranz and Thorsten Joachims (Eds.). Omnipress, 807–814. https://icml.cc/Conferences/2010/papers/432.pdf

[130] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding Faster Configurations using FLASH. *IEEE Transactions on Software Engineering* (2018).

[131] Manoj Nambiar, Ajay Kattepur, Gopal Bhaskaran, Rekha Singhal, and Subhasri Duttagupta. 2016. Model Driven Software Performance Engineering: Current Challenges and Way Ahead. *SIGMETRICS Perform. Evaluation Rev.* 43, 4 (2016), 53–62. https://doi.org/10.1145/2897356.2897363

[132] Daniel Nemirovsky, Tugberk Arkose, Nikola Markovic, Mario Nemirovsky, Osman S. Unsal, and Adrián Cristal. 2017. A Machine Learning Approach for Performance Prediction and Scheduling on Heterogeneous CPUs. In *29th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2017, Campinas, Brazil, October 17-20, 2017*. IEEE Computer Society, 121–128. https://doi.org/10.1109/SBAC-PAD.2017.23

[133] Daniel Nichols, Aniruddha Marathe, Harshitha Menon, Todd Gamblin, and Abhinav Bhatele. 2023. Modeling Parallel Programs using Large Language Models. *CoRR* abs/2306.17281 (2023). https://doi.org/10.48550/ARXIV.2306.17281 arXiv:2306.17281

[134] Abdelkader Ouared, Abdelhafid Chadli, and Mohamed Amine Daoud. 2022. DeepCM: Deep neural networks to improve accuracy prediction of database cost models. *Concurr. Comput. Pract. Exp.* 34, 10 (2022). https://doi.org/10.1002/cpe.6724

[135] Kewen Peng, Christian Kaltenecker, Norbert Siegmund, Sven Apel, and Tim Menzies. 2021. VEER: Disagreement-Free Multi-objective Configuration. *CoRR* abs/2106.02716 (2021). arXiv:2106.02716 https://arxiv.org/abs/2106.02716

[136] 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

[137] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning software configuration spaces: A systematic literature review. *J. Syst. Softw.* 182 (2021), 111044. https://doi.org/10.1016/J.JSS.2021.111044

[138] Thanh-Phuong Pham, Juan José Durillo, and Thomas Fahringer. 2020. Predicting Workflow Task Execution Time in the Cloud Using A Two-Stage Machine Learning Approach. *IEEE Trans. Cloud Comput.* 8, 1 (2020), 256–268. https://doi.org/10.1109/TCC.2017.2732344

[139] Sabri Pllana, Ivona Brandic, and Siegfried Benkner. 2007. Performance Modeling and Prediction of Parallel and Distributed Computing Systems: A Survey of the State of the Art. In *First International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2007), Vienna, Austria, 10-12 April 2007*. IEEE, 279–284.

[140] Ivan Porres, Tanwir Ahmad, Hergys Rexha, Sébastien Lafond, and Dragos Truscan. 2020. Automatic exploratory performance testing using a discriminator neural network. In *13th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2020, Porto, Portugal, October 24-28, 2020*. IEEE, 105–113. https://doi.org/10.1109/ICSTW50294.2020.00030

[141] Joy Rahman and Palden Lama. 2019. Predicting the End-to-End Tail Latency of Containerized Microservices in the Cloud. In *IEEE International Conference on Cloud Engineering, IC2E 2019, Prague, Czech Republic, June 24-27, 2019*. IEEE, 200–210. https://doi.org/10.1109/IC2E.2019.00034

[142] Research.com. 2024. Conference Rankings in Computer Science and Software Programming. https://research.com/conference-rankings/computer-science/software-programming. Accessed: 2024-06-15.

[143] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Model-Agnostic Interpretability of Machine Learning. *CoRR* abs/1606.05386 (2016). arXiv:1606.05386 http://arxiv.org/abs/1606.05386

[144] Ann Sabbeh, Yousif Al-Dunainawi, HS Al-Raweshidy, and Maysam F Abbod. 2016. Performance prediction of software defined network using an artificial neural network. In *2016 SAI Computing Conference (SAI)*. IEEE, 80–84.

[145] Omar Said and Amr Tolba. 2021. Accurate performance prediction of IoT communication systems for smart cities: An efficient deep learning based solution. *Sustainable Cities and Society* 69 (2021), 102830.

[146] Peter Peter Samoaa, Linus Aronsson, Antonio Longa, Philipp Leitner, and Morteza Haghir Chehreghani. 2023. A Unified Active Learning Framework for Annotating Graph Data with Application to Software Source Code Performance Prediction. *CoRR* abs/2304.13032 (2023). https://doi.org/10.48550/arXiv.2304.13032 arXiv:2304.13032

[147] Fernando García Sanz, Masoumeh Ebrahimi, and Andreas Johnsson. 2022. Exploring Approaches for Heterogeneous Transfer Learning in Dynamic Networks. In *2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*. IEEE, 1–9. https://doi.org/10.1109/NOMS54207.2022.9789761

[148] Atri 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*. 342–352.

[149] 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

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

[151] 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. https://doi.org/10.1007/s11219-011-9152-9

[152] Dalwinder Singh and Birmohan Singh. 2020. Investigating the impact of data normalization on classification performance. *Appl. Soft Comput.* 97, Part B (2020), 105524. https://doi.org/10.1016/J.ASOC.2019.105524

[153] Shikhar Singh, James Hegarty, Hugh Leather, and Benoit Steiner. 2022. A graph neural network-based performance model for deep learning applications. In *MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022*, Swarat Chaudhuri and Charles Sutton (Eds.). ACM, 11–20. https://doi.org/10.1145/3520312.3534863

[154] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*. 2960–2968.

[155] 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

[156] Szymon Stradowski and Lech Madeyski. 2023. Machine learning in software defect prediction: A business-driven systematic mapping study. *Inf. Softw. Technol.* 155 (2023), 107128. https://doi.org/10.1016/J.INFSOF.2022.107128

[157] Jingwei Sun, Guangzhong Sun, Shiyan Zhan, Jiepeng Zhang, and Yong Chen. 2020. Automated Performance Modeling of HPC Applications Using Machine Learning. *IEEE Trans. Computers* 69, 5 (2020), 749–763. https://doi.org/10.1109/TC.2020.2964767

[158] Yong Tang, Ronghua Lin, Dingding Li, Yuguo Li, and Deze Zeng. 2022. FSbrain: An intelligent I/O performance tuning system. *J. Syst. Archit.* 129 (2022), 102623. https://doi.org/10.1016/j.sysarc.2022.102623

[159] Paul Temple, Mathieu Acher, Gilles Perrouin, Battista Biggio, Jean-Marc Jézéquel, and Fabio Roli. 2019. Towards quality assurance of software product lines with adversarial configurations. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC*. 38:1–38:12.

[160] Jakob Thrane, Darko Zibar, and Henrik Lehrmann Christiansen. 2020. Model-Aided Deep Learning Method for Path Loss Prediction in Mobile Communication Systems at 2.6 GHz. *IEEE Access* 8 (2020), 7925–7936. https://doi.org/10.1109/ACCESS.2020.2964103

[161] Ashkan Tousi and Mikel Luján. 2022. Comparative Analysis of Machine Learning Models for Performance Prediction of the SPEC Benchmarks. *IEEE Access* 10 (2022), 11994–12011. https://doi.org/10.1109/ACCESS.2022.3142240

[162] Jelena Trajkovic, Sara Karimi, Samantha Hangsan, and Wenlu Zhang. 2022. Prediction Modeling for Application-Specific Communication Architecture Design of Optical NoC. *ACM Trans. Embed. Comput. Syst.* 21, 4 (2022), 35:1–35:29. https://doi.org/10.1145/3520241

[163] Ngoc Tran, Jean-Guy Schneider, Ingo Weber, and A. Kai Qin. 2020. Hyper-parameter optimization in classification: To-do or not-to-do. *Pattern Recognit.* 103 (2020), 107245. https://doi.org/10.1016/j.patcog.2020.107245

[164] Lukas Trümper, Tal Ben-Nun, Philipp Schaad, Alexandru Calotoiu, and Torsten Hoefler. 2023. Performance Embeddings: A Similarity-Based Transfer Tuning Approach to Performance Optimization. In *Proceedings of the 37th International Conference on Supercomputing, ICS 2023, Orlando, FL, USA, June 21-23, 2023*, Kyle A. Gallivan, Efstratios Gallopoulos, Dimitrios S. Nikolopoulos, and Ramón Beivide (Eds.). ACM, 50–62. https://doi.org/10.1145/3577193.3593714

[165] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. 2021. White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1072–1084. https://doi.org/10.1109/ICSE43902.2021.00100

[166] 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

[167] Arthur Vitui and Tse-Hsun (Peter) Chen. 2021. Correction to: MLASP: Machine learning assisted capacity planning. An industrial experience report. *Empir. Softw. Eng.* 26, 5 (2021), 109. https://doi.org/10.1007/s10664-021-10011-7

[168] Han Wang, Lingwei Xu, Ye Tao, and Xianpeng Wang. 2020. OP Performance Prediction for Complex Mobile Multiuser Networks Based on Extreme Learning Machine. *IEEE Access* 8 (2020), 14557–14564. https://doi.org/10.1109/ACCESS.2020.2966690

[169] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. 2021. Morphling: fast, near-optimal auto-configuration for cloud-native model serving. In *Proceedings of the ACM Symposium on Cloud Computing*. 639–653.

[170] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. 2023. Machine/Deep Learning for Software Engineering: A Systematic Literature Review. *IEEE Trans. Software Eng.* 49, 3 (2023), 1188–1231. https://doi.org/10.1109/TSE.2022.3173346

[171] Tiantian Wang, Mark Harman, Yue Jia, and Jens Krinke. 2013. Searching for better configurations: a rigorous approach to clone evaluation. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13*. 455–465.

[172] Yu Wang, Victor Lee, Gu-Yeon Wei, and David M. Brooks. 2019. Predicting New Workload or CPU Performance by Analyzing Public Datasets. *ACM Trans. Archit. Code Optim.* 15, 4 (2019), 53:1–53:21. https://doi.org/10.1145/3284127

[173] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. 2021. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Comput. Surv.* 53, 3 (2021), 63:1–63:34. https://doi.org/10.1145/3386252

[174] Zhenyi Wang, Pengfei Yang, Linwei Hu, Bowen Zhang, Chengmin Lin, Wenkai Lv, and Quan Wang. 2024. SLAPP: Subgraph-level attention-based performance prediction for deep learning models. *Neural Networks* 170 (2024), 285–297.

[175] Laura Wartschinski, Yannic Noller, Thomas Vogel, Timo Kehrer, and Lars Grunske. 2022. VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. *Inf. Softw. Technol.* 144 (2022), 106809. https://doi.org/10.1016/j.infsof.2021.106809

[176] Cody Watson, Nathan Cooper, David Nader-Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.* 31, 2 (2022), 32:1–32:58. https://doi.org/10.1145/3485275

[177] André Brasil Vieira Wyzykowski, Gabriel Mascarenhas Costa De Sousa, Breno Spinelli Coelho, Lucas Batista Santos, and Darlan Anschau. 2024. Optimizing Geophysical Workloads in High-Performance Computing: Leveraging Machine Learning and Transformer Models for Enhanced Parallelism and Processor Allocation. In *2024 Third International Conference on Distributed Computing and High Performance Computing (DCHPC)*. IEEE, 1–14.

[178] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE*. 307–319.

[179] 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

[180] Yanming Yang, Xin Xia, David Lo, and John C. Grundy. 2022. A Survey on Deep Learning for Software Engineering. *ACM Comput. Surv.* 54, 10s (2022), 206:1–206:73. https://doi.org/10.1145/3505243

[181] 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

[182] Xiaojing Yin, Jiwei Huang, Lei Liu, Wei He, and Lizhen Cui. 2023. An reinforcement learning approach for allocating software resources. *Concurr. Comput. Pract. Exp.* 35, 14 (2023). https://doi.org/10.1002/cpe.6349

[183] Dewi Yokelson, Marc Robert Joseph Charest, and Ying Wai Li. 2023. HPC Application Performance Prediction with Machine Learning on New Architectures. In *Proceedings of the 2023 on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy, PERMAVOST 2023, Orlando, FL, USA, June 20-30, 2023*, Connor Scully-Allison, Radita Liem, Ana Veroneze Solorzano, Ayesha Afzal, and Pouya Kousha (Eds.). ACM, 1–8. https://doi.org/10.1145/3588993.3597262

[184] Jixiang Yu, Ming Gao, Yuchan Li, Zehui Zhang, Wai Hung Ip, and Kai-Leung Yung. 2022. Workflow performance prediction based on graph structure aware deep attention neural network. *J. Ind. Inf. Integr.* 27 (2022), 100337. https://doi.org/10.1016/j.jii.2022.100337

[185] Tingting Yu, Wei Wen, Xue Han, and Jane Huffman Hayes. 2019. ConPredictor: Concurrency Defect Prediction in Real-World Applications. *IEEE Trans. Software Eng.* 45, 6 (2019), 558–575.

[186] Zuhaira Muhammad Zain, Sapiah Sakri, Nurul Halimatul Asmak Ismail, and Reza M Parizi. 2022. Software Defect Prediction Harnessing on Multi 1-Dimensional Convolutional Neural Network Structure. *Computers, Materials & Continua* 71, 1 (2022).

[187] Zuhaira Muhammad Zain, Sapiah Binti Sakri, and Nurul Halimatul Asmak Ismail. 2023. Application of Deep Learning in Software Defect Prediction: Systematic Literature Review and Meta-analysis. *Inf. Softw. Technol.* 158 (2023), 107175. https://doi.org/10.1016/J.INFSOF.2023.107175

[188] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. 2012. A qualitative study on performance bugs. In *9th IEEE Conference of Mining Software Repositories, MSR.* 199–208.

[189] Dongwen Zhang, Tongtong Wang, and Yang Zhang. 2023. Performance Prediction for Lock-based Programs. In *23rd IEEE International Conference on Software Quality, Reliability, and Security, QRS 2023 Companion, Chiang Mai, Thailand, October 22-26, 2023.* IEEE, 437–446. https://doi.org/10.1109/QRS-C60940.2023.00028

[190] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Inf. Softw. Technol.* 53, 6 (2011), 625–637. https://doi.org/10.1016/J.INFSOF.2010.12.010

[191] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 415–432. https://doi.org/10.1145/3299869.3300085

[192] Wei Zhang, Zhihui Lu, Ziyan Wu, Jie Wu, Huanying Zou, and Shalin Huang. 2018. Toy-IoT-Oriented data-driven CDN performance evaluation model with deep learning. *J. Syst. Archit.* 88 (2018), 13–22. https://doi.org/10.1016/j.sysarc.2018.05.005

[193] Yang Zhang, Fuli Feng, Chenxu Wang, Xiangnan He, Meng Wang, Yan Li, and Yongdong Zhang. 2020. How to Retrain Recommender System?: A Sequential Meta-Learning Method. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 1479–1488. https://doi.org/10.1145/3397271.3401167

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

[195] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query Performance Prediction for Concurrent Queries using Graph Embedding. *Proc. VLDB Endow.* 13, 9 (2020), 1416–1428. https://doi.org/10.14778/3397230.3397238

[196] Kun Zhu, Shi Ying, Nana Zhang, and Dandan Zhu. 2021. Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. *J. Syst. Softw.* 180 (2021), 111026. https://doi.org/10.1016/j.jss.2021.111026

[197] Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. 2018. How practitioners perceive automated bug report management techniques. *IEEE Transactions on Software Engineering* 46, 8 (2018), 836–862.