Distilled Lifelong Self-Adaptation for Configurable Systems

Yulong Ye[1,2], Tao Chen[1,2*], Miqing Li[2]
[1] IDEAS Lab, University of Birmingham, United Kingdom
[2] School of Computer Science, University of Birmingham, United Kingdom
yxy382@student.bham.ac.uk, t.chen@bham.ac.uk, m.li.8@bham.ac.uk

***Abstract*—Modern configurable systems provide tremendous opportunities for engineering future intelligent software systems. A key difficulty thereof is how to effectively self-adapt the configuration of a running system such that its performance (e.g., runtime and throughput) can be optimized under time-varying workloads. This unfortunately remains unaddressed in existing approaches as they either overlook the available past knowledge or rely on static exploitation of past knowledge without reasoning the usefulness of information when planning for self-adaptation. In this paper, we tackle this challenging problem by proposing `DLiSA`, a framework that self-adapts configurable systems. `DLiSA` comes with two properties: firstly, it supports lifelong planning, and thereby the planning process runs continuously throughout the lifetime of the system, allowing dynamic exploitation of the accumulated knowledge for rapid adaptation. Secondly, the planning for a newly emerged workload is boosted via distilled knowledge seeding, in which the knowledge is dynamically purified such that only useful past configurations are seeded when necessary, mitigating misleading information.**

Extensive experiments suggest that the proposed `DLiSA` significantly outperforms state-of-the-art approaches, demonstrating a performance improvement of up to 229% and a resource acceleration of up to 2.22× on generating promising adaptation configurations. All data and sources can be found at our repository: https://github.com/ideas-labo/dlisa.

***Index Terms*—Self-adaptive systems, search-based software engineering, dynamic optimization, configuration tuning**

I. Introduction

Software systems are often highly configurable [1]–[5]. However, their operation environment is often confronted with dynamic and uncertain conditions that change over time [6], [7], which is crucial to their performance (e.g., runtime [8]). Taking the H2 database system as an example, its real-time workloads are known to highly fluctuate, prompting the system to dynamically adjust its configuration options to accommodate such changes [9].

To mitigate this, one promising way is to engineer self-adaptive configurable systems—a specific type of self-adaptive systems that, when the workload changes, self-adapt their configurations to meet different performance needs [10]–[13]. The critical challenge of self-adaptation lies in planning [14]–[16], i.e., how to identify the most effective configuration (a.k.a. adaptation plan) amidst constantly changing workload at runtime. Recently, Search-Based Software Engineering (SBSE) has been considered a promising direction for solving this challenge, which tries to iteratively search for and refine configurations to locate the optimal one by using tailored search algorithms [17]–[20]. The inherent search and optimization properties of SBSE make it well-suited to addressing the complexities of huge configuration spaces encountered in adapting configurable systems. Importantly, SBSE exhibits highly extensible potentials for complementing other approaches, such as control theoretical [21]–[23] and learning-based methods [24]–[29], to achieve integrated schemes, thereby providing a comprehensive solution for runtime planning.

Beyond the exponentially growing search space and the non-linear interaction among configuration options, the ever-changing workload further intensifies the planning difficulties when self-adapting configurable systems. Particularly, the landscape of the search space may shift dramatically across different workloads, suggesting that a configuration optimized for one workload may become suboptimal or even perform poorly in another [11]. This dynamic nature requires planning to not only search for an optimal configuration under a newly emerged workload but should be doing so rapidly.

A promising resolution to that end is to reuse "past knowledge", i.e., configurations that were optimized under the previous workloads, for the planning to start working with under the current workload [20]. However, unfortunately, existing works often assume a stationary adaptation, which restarts the planning process from scratch following each workload change or at a fixed frequency. Such methods may be inefficient, as they fail to fully utilize historical search experiences, resulting in repetitive effort and a waste of valuable information that could inform more effective adaptation planning [17], [18], [30]. Indeed, certain approaches have followed a dynamic adaptation [19], [20], [31] that exploit configurations found previously to speedup the planning (i.e., seeding). This, while running continuously, can still generate negative outcomes as the way how knowledge is exploited follows a static strategy: all (or randomly selected) configurations from the most recent past workload are used while any of those from earlier workloads are discarded. That said, the idea seems intuitive—the latest workload that has been changed may provide more useful information for the current newly emerged one while those older workloads may often be less useful. Yet, since the order of workload arriving in the system is uncertain, there is

*Corresponding author.

no guarantee that the configurations from the latest workload are all promising for the current one nor those from earlier workloads are completely irrelevant, as what has been implied in prior work [9] and observed from our study in Section-II.C.

To fill the above gap, in this paper, we propose a framework, dubbed DLiSA, to self-adapt configurable systems at runtime based on the MAPE-K loop [32]. DLiSA comes with a combination of two properties that makes it distinctive: (1) **lifelong planning**, where we leverage an evolutionary algorithm in the planning that runs continuously throughout the lifetime of the system while providing the foundation for seeding; and (2) **distilled knowledge seeding**—a truly dynamic knowledge exploitation strategy such that it not only seeds past configurations when there is evidence that they can be beneficial but also extracts the most useful ones to seed from all historical workloads, hence mitigating the misleading noises while keeping the most useful information. In this way, DLiSA ensures that the exploitation of past knowledge is neither static nor completely abandoned, which fits with the characteristics of configurable systems.

In a nutshell, our main contributions are as follows:

- We show, by examples of configurable systems' landscapes, the key characteristic faced by self-adapting configuration at runtime under changing workloads.
- We develop a ranked workload similarity analysis to excavate correlations and patterns of past workloads, helping to extrapolate the traits of the new workload for more informed adaptation planning.
- We propose a weighted configuration seeding that distills the past knowledge, seeding only the most useful configurations and mitigating the misleading ones.
- DLiSA is experimentally evaluated against four state-of-the-art approaches on nine real-world systems with different performance objectives, scales, and complexity, including 6–13 time-varying workloads. This leads to a total of 93 cases to investigate.

Experimental results encouragingly demonstrate that DLiSA exhibits significant improvements in both efficacy (up to 2.29×) and efficiency (up to 2.22×).

The rest of this paper is organized as follows. Section II introduces the background and motivation. Section III provides the details of our proposed DLiSA. Section IV presents our experiment methodology, followed by the experimental results in Section V. Section VI discusses the most noticeable aspects of DLiSA. Threats to validity, related work, and conclusion are presented in Sections VII, VIII, and IX, respectively.

II. Background and Motivation

In this section, we discuss the preliminaries and main motivation of this work.

A. Self-Adaptive Configurable Systems

In this work, we focus on self-adaptive configurable systems. According to a well-known taxonomy [10], the self-adaptive configurable systems differ from the other concepts as follows:

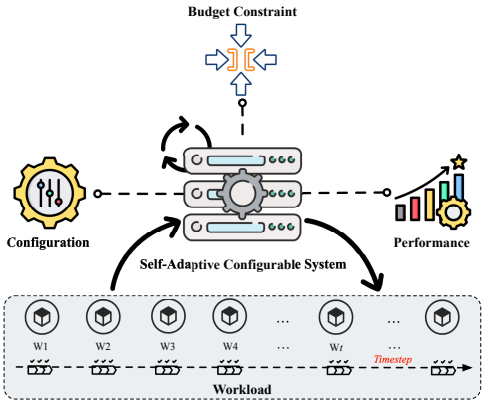


Fig. 1: Self-adaptation planning for configurable systems.

- **Self-Adaptive Systems:** These systems adapt to changes by modifying their behaviors, which could be any system's states (including structure and parameters) at runtime [10].
- **Self-Reconfigurable Systems:** A special type of self-adaptive systems that primarily alter their structure/architecture (including parameters) to adapt [33].
- **Self-Adaptive Configurable Systems:** Unlike others, these systems specifically adapt by adjusting configuration parameters [34].

Clearly, the self-adaptive configurable system is a type of self-adaptive/self-reconfigurable systems that primarily adjust configuration parameters at runtime to optimize their performance [10], [33], [34], focusing at the intersection between *self-optimized* and *self-configured* systems [10], which have been frequently studied in prior work involving dynamic workloads [11]–[13].

B. Problem Formalization

Without loss of generality, self-adaptation planning for a given configurable system involves the following key concepts, as shown in Figure 1.

- **System:** A configurable system with configurable options that can be adjusted at runtime.
- **Workload:** The time-varying and uncertain receiving jobs for which the system handles. The concrete instance can vary. For example, the workloads refer to different types and volumes of queries that emerged for database system H2; for file compressors such as KANZI, this becomes the incoming files to be compressed, which could be of diverse formats and sizes.
- **Configuration:** An instance of variability for a configurable system, formed by a set of values for the configurable options. In this work, we consider both the configurations (and options) that require system rebooting and those that do not.
- **Performance:** The metric(s) that evaluates the behavior of the system, such as runtime (i.e., the time taken by the system to process a given workload) and throughput.
- **Budget constraint:** The budget of cost allowed for self-adaptation planning under the workload for a particular

timestep. While the definition of budget varies, in this work, we use the number of system measurements during planning as the budget, which means that we can only measure a certain number of configurations as our constraint. The measurement is chosen because: (1) it is independent of the implementation, such as language and hardware; (2) it eliminates the interference of clock time caused by the running system to be adapted, when it is run at the same machine as the planning process; (3) it has been widely used in existing work [2], [35].

When a single performance objective is of concern, the goal of planning when self-adapting a configurable system S is, for each timestep t in which the system handles a workload over the time horizon, to identify a configuration that optimizes the specific performance attribute, e.g., minimizing runtime or maximizing throughput, of the target system, subject to a budget constraint for planning. Formally, this can be defined as:

$$\begin{aligned} &\arg\min f_t(\mathbf{x}) \text{ or } \arg\max f_t(\mathbf{x}), \\ &\text{s.t. } r_t \leq R_t, \end{aligned} \tag{1}$$

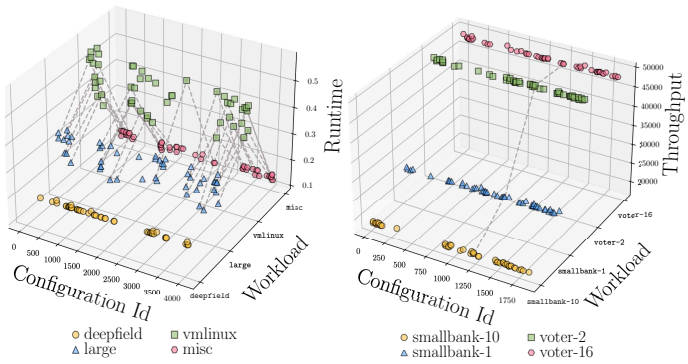
where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a configuration with the values of n options (e.g., x_n) in search space $\boldsymbol{\mathcal{X}}$. f_t represents the performance attribute of the target system. r_t and R_t respectively denote the cost consumption and the budget allowed for planning at timestep t.

C. Motivation and Challenges

It is well-known that, when self-adapting configurable systems, the configurations produced under one workload might be useful to the other workloads [9], [12], [27]. However, it remains unclear how to explicitly extract the key knowledge and whether there exists irrelevant or even misleading information, i.e., noises. To uncover these underlying issues, we analyze the datasets collected from commonly used configurable systems and their workload from prior studies [9], [36]–[38]. The goal is to investigate what are the similarities and discrepancies between the configuration landscapes of different workloads. Figure 2 shows the top 50 performing configurations for two systems under different workloads and we observe the following patterns (similar observations exist in other systems):

- There could be a strong overlap of the promising configurations across workloads (the connected points). That said, a promising configuration under a workload could also be promising under the others. For example, the points connected by dashed lines for workloads `large`, `vmlinux`, and `misc` of KANZI in Figure 2a.
- It is also possible that the promising configurations for each individual workload differ significantly. For instance, the points under workloads of H2 in Figure 2b rarely overlap with each other. The same phenomenon occurs even for the workloads of the same system, e.g., workload `deepfield` against the others for KANZI.

The above leads to a key characteristic for configurable systems, which motivates our work:



(a) KANZI (b) H2

Fig. 2: An illustration of similarities and discrepancies in top 50 performing configurations across workloads. The same configurations are connected by dashed lines.

Key Characteristic

Top-performing configurations between workloads can be very similar or very discrepant, depending on both the systems (e.g., KANZI *and* H2*) and the workloads within a single system (e.g., between workload* `deepfield` *and the others for* KANZI*).*

Since the order of workloads arriving at a system is uncertain, the above suggests that "seeding"[1] promising configurations optimized for the past workloads to the planning under the current workload can be beneficial, as long as we can:

- **Challenge 1:** extract the most useful configurations discovered previously (the configurations that are promising across workloads), if any, while doing so without injecting misleading information (the configurations that are "good" under the past workloads only);
- **Challenge 2:** and detect when it is generally more harmful to seed than simply restart planning.

Nevertheless, existing approaches have failed to explicitly handle the above characteristics and challenges of configurable systems when running under changing workloads: on one hand, *stationary adaptation* approaches (e.g., `FEMOSAA` [30]) restarts a new search/planning from scratch with each workload change, but clearly, according to the above characteristic, this would waste the valuable knowledge from the past workload instances that could have been exploited [39]. On the other hand, the *dynamic adaptation* approaches (e.g., `Seed-EA` [20]) rely on a static assumption for the knowledge exploitation strategy: all configurations accumulated to the most recent past workload are useful for seeding, while those from previous ones are discarded. Besides, they always trigger seeding even when the benefits are unjustified. Because the seeds retain the planning state, indeed, the dynamic approaches

[1]Seeding is a mechanism that benefits the planning for the current workload by reusing configurations optimized under the past workloads [20], [31].

TABLE I: Comparing DLiSA against other approaches.

Approach	Knowledge Exploitation	Seeding	Workloads	Configurations
FEMOSAA [30]	N/A	N/A	N/A	N/A
Seed-EA [20]	Static	Always	Most recent past	All
D-SOGA [40]	Static	Always	Most recent past	Random
LiDOS [11]	Static	Always	Most recent past	All
DLiSA	Dynamic	On-demand	All historical	Distilled

can also be executed in a "lifelong" manner where the planning process runs continuously and adapts to the changes in the workload, but they may pick up potentially misleading information (optimal configurations found in the most recent past workload but are no longer promising) or missing useful hints (generally promising configurations found under earlier workloads), hindering the planning in the current workload.

The above, therefore, are the key challenges and limitations we address in this paper. Table I summarizes the novelty of DLiSA against the properties of state-of-the-art approaches.

III. The DLiSA Framework

To tackle the current limitation and handle the key characteristic/challenges discussed in Section II-C, we propose DLiSA—a distilled lifelong planning framework for self-adapting configurable systems with time-varying workloads.

DLiSA comes with two unique properties:

- **Lifelong planning:** The planning runs continuously and adapts to workload changes—a typical case of dynamic optimization [41]—in which the state optimized across different workloads can be preserved. This provides the foundation for addressing **Challenge 1**.
- **Distilled knowledge seeding:** The knowledge of seeding is dynamically distilled, i.e., DLiSA extracts the most useful configurations from all past workloads to seed into the current planning process; or triggers randomly-initialized planning from scratch when the overall distilled knowledge is deemed not sufficiently useful. This tackles both **Challenge 1** and **Challenge 2**.

Next, we will articulate DLiSA's designs in great detail.

A. Architecture Overview

We design DLiSA using the typical MAPE-K architecture [32], as shown in Figure 3 and **Algorithm 1**. In a nutshell, MAPE-K distinguishes two sub-systems—the managed system refers to the configurable systems that should be managed; and the managing system governs the self-adaptation, i.e., DLiSA. Once a workload change has been detected (e.g., a new incoming job), the *Monitor* informs the *Analyzer* to analyze the current and past status, which then triggers *Planner* for reasoning about the best self-adaptation plan (configuration), subject to a given budget constraint. Finally, the best-optimized configuration is set to the managed system via *Executor*. The *Knowledge* refers to the preserved data that can be used by any phases in the MAPE loop. In this work, the knowledge we retain is the workloads experienced by the systems and all the corresponding configurations that were measured/discovered in the planning previously (line 7).

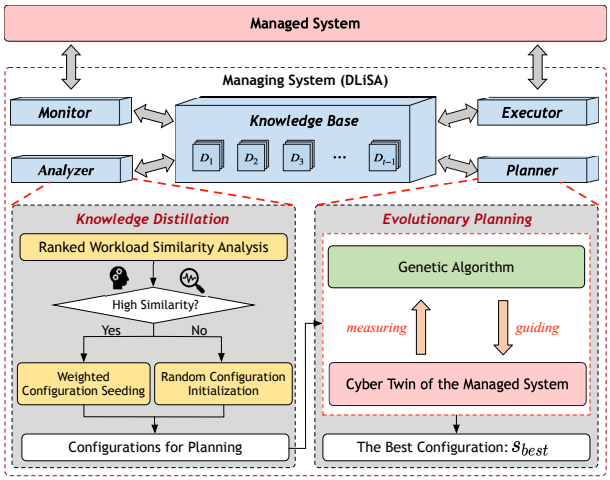


Fig. 3: DLiSA architecture for configurable systems.

Algorithm 1: DLiSA Framework

Input: Cyber-Twin of the managed system $\mathcal{S}$; the budge constraint R_t; threshold for triggering seeding α; population size N.
Declare: The best configuration at current workload s_{best}; the set of configurations preserved at workload t, $\mathbf{D}_t$. Seeded set of configurations $\mathbf{P}$, and the *knowledge base* $\mathcal{K}$.

```
1  while the managed system is running do
2      t = 0
3      if workload change then
4          t = t + 1
5          P = KNOWLEDGEDISTILLATION(K, α, N)
6          D_t, s_best ← EVOLUTIONARYPLANNING(S, K, P, R_t)
7          K = K ∪ D_t
8          SENDFORADAPTATION(s_best)
9      end
10 end
```

DLiSA specializes two key phases in MAPE-K (lines 5-6):

- **Analyzer:** The *Knowledge Distillation* component identifies whether seeding is beneficial, and if that is the case, extracts the most useful configurations preserved previously to seed the planning under the current workload, realizing **distilled knowledge seeding** (see Section III-B).
- **Planner:** Here, we leverage *Evolutionary Planning* component to evolve the configurations into better ones in the search space based on the given seeds, if any. We adopt a population-based optimizer where a set of the most promising configurations found is preserved and the best one is used for self-adaptation under a workload (see Section III-C). In particular, the search-based planning is conducted on a sandbox, which is often a Cyber-Twin or a surrogate system model, that allows expedited measurement of the configurations while emulating the behavior of the managed system under the given workload [42], [43]. Through those seeds, the planning status for the previous workloads can be kept, hence rendering the entire process as **lifelong planning** over the time horizon.

B. Knowledge Distillation

As shown in Figure 3, with the dynamic exploitation strategy realized by knowledge distillation, DLiSA seeks to distill

Algorithm 2: KNOWLEDGEDISTILLATION

Input: The knowledge base $\mathcal{K}$; threshold for triggering seeding α; initial/population size N
Output: The set of initial configuration for planning $\mathbf{P}$

/* Ranked workload similarity analysis */
1 **for** t = 1 to SIZE($\mathcal{K}$)−1 **do**
2 $\quad \mathbf{D}_t^{t+1} \leftarrow$ common configurations from those evaluated in the planning of two adjacent workloads $\mathbf{D}_t$ and $\mathbf{D}_{t+1}$ from $\mathcal{K}$
3 $\quad S_t^{t+1} \leftarrow$ calculate the similarity between workloads t and $t+1$ by (2) and (3)
4 $\quad S_{sum} = S_{sum} + S_t^{t+1}$
5 **end**
6 S_{ave} = AVERAGING(S_{sum})
/* Weighted configuration seeding */
7 **if** $S_{ave} \geq \alpha$ *and* SIZE($\mathcal{K}$)> 0 **then**
8 $\quad \mathbf{C} \leftarrow$ the $N/2$ best configurations under each workload
9 $\quad$ **foreach** $\forall c \in \boldsymbol{C}$ **do**
$\quad\quad$ /* Get quality weight for c by (4)-(6) */
10 $\quad\quad \mathbf{C}_w \leftarrow \langle c, w_c \rangle \leftarrow w_c = w_{c,r} + w_{c,t}$
11 $\quad$ **end**
12 $\quad \mathbf{P} \leftarrow$ stochastically pick N configurations from $\mathbf{C}_w$ using w_c
$\quad$ /* Only happens with one past workload */
13 $\quad$ **if** $|\boldsymbol{P}| \neq N$ **then**
14 $\quad\quad \mathbf{P} \leftarrow$ randomly initialize configurations till $|\mathbf{P}| = N$
15 $\quad$ **end**
16 **else**
17 $\quad \mathbf{P} \leftarrow$ randomly initialize configurations till $|\mathbf{P}| = N$
18 **end**
19 **return P**

the configurations optimized for all past workloads in two steps during planning: firstly, it selects representative configurations evaluated to assess the overall similarity amongst their performance across the workloads using a ranked similarity metric at the *workload level*. A high value of the metric represents a higher likelihood of the seeding being useful. Next, if there is convincing evidence that seeding can be beneficial for planning, we probabilistically extract N most useful configurations amongst those preserved for seeding using a quality weight at the *configuration level*; otherwise, a random initialization process is used instead. An algorithmic illustration has also been shown in **Algorithm 2**.

Ranked Workload Similarity Analysis (When to seed?): For the trigger of seeding, the idea is that, if the majority of those configurations that were discovered under the past workloads are "similarly good", then it is likely that there is a strong chance for certain promising configurations optimized previously to be equally good under the current, newly emerged workload, i.e., the seeding should be beneficial.

As a result, we propose a ranked workload similarity analysis at the workload level using all the common configurations searched across workloads (including those that were ruled out). In particular, we quantify the similarity level between two workloads by using a pairwise ranking loss [44]. The rationale is that while the concrete performance of a configuration may fluctuate with workload changes, the relative rankings can remain indicative of similarity while being scale-free.

We do so via the following steps (lines 1-6):

1) For every pair of adjacent workloads (e.g., t and $t+1$), retrieve all the evaluated configurations $\mathbf{D}_t$ and $\mathbf{D}_{t+1}$.
2) Identify their common configurations $\mathbf{D}_t^{t+1}$ (line 2).
3) Compute the ranking loss by quantifying the number of misranked pairs in $\mathbf{D}_t^{t+1}$ (line 3):

$$\mathcal{L}(\mathbf{D}_t^{t+1}) = \sum_{j=1}^{N_t^{t+1}} \sum_{k=1}^{N_t^{t+1}} \mathbf{1}((f_t(\mathbf{x}_j) < f_t(\mathbf{x}_k)) \oplus (f_{t+1}(\mathbf{x}_j) < f_{t+1}(\mathbf{x}_k))), \tag{2}$$

whereby $\oplus$ is the exclusive-or operator; N_t^{t+1} is the number of configurations evaluated in both workloads t and $t+1$ (i.e., the size of $\mathbf{D}_t^{t+1}$). The ranking loss $\mathcal{L}(\mathbf{D}_t^{t+1})$ represents the number of misranked pairs of configurations between adjacent workloads at timesteps t and $t+1$, reflecting the discrepancy among them.
4) Assess the similarity between t and $t+1$ (S_t^{t+1}) using the percentage of the order-preserving pairs, as follows:

$$S_t^{t+1} = 1 - \frac{\mathcal{L}(\mathbf{D}_t^{t+1})}{N_{pairs}}, \tag{3}$$

where N_{pairs} is the number of configuration combination in $\mathbf{D}_t^{t+1}$ (line 3).
5) Calculate the average similarity score for all pairs of adjacent workloads, i.e., S_{sav} (line 6).

The seeding is said to be beneficial and should be triggered only if $S_{sav} \geq \alpha$, where α is a given threshold. Note that, if no common configurations are found between a pair of adjacent workloads, we set their similarity S_t^{t+1} with a random value that is less than α, serving as a reasonable guess when no reliable information can be extracted.

Weighted Configuration Seeding (What to seed?): When DLiSA determines that the seeding is necessary, we need to further select the most useful configurations for seeding the current workload. As observed in Section II-C, there could be a strong overlap of good configurations across different workloads, yet sometimes, the promising ones for different workloads can be highly discrepant. Our idea here is to design a weighting scheme, such that it can discriminate the past configurations based on the likelihood of them being promising under the current workload. To this end, we design a two-stage weighted seeding that operates at the configuration level, considering only the good configurations preserved. As such, we say a past configuration is useful for seeding if (1) it is good within its own workload (line 8) while (2) being robust and timely across all past workloads (lines 9-12).

The first stage—the local stage weighting—focuses on selecting the best configurations locally (based on the performance objective) under each workload. This is because those configurations that perform badly in a workload would be less meaningful for seeding. To that end, we filter the preserved configurations at each workload by 50%, i.e., only $\frac{N}{2}$ configurations are considered where N is the number of configurations to be seeded in the end (line 8).

In the second stage, we seek to globally weight the configurations across all the past workloads. The hypotheses are:

- preserved configurations that have demonstrated robustness in many past workloads (as they were not ruled out) are likely to perform well in the new workload;

- since planning under a later workload might have evolved by integrating previously accumulated knowledge, configurations preserved in such a later workload are likely to exhibit good performance in the new workload.

Therefore, we use quality weight to sort the previously selected configurations from the first stage and it has two components: a robustness weight and a timeliness weight (lines 9-12). Specifically, a robustness weight is allocated to each configuration based on its recurrence across multiple past workloads (line 10). Configurations that appear in a larger number of workloads receive higher robustness weights, reflecting their robustness for being preferred frequently and the likelihood of successful performance:

$$w_{c,r} = \frac{O_c}{H}, \tag{4}$$

where O_c is the count of past workloads in which the configuration c is preserved and H denotes the total number of past workloads. In contrast, the timeliness weight of a configuration is calculated based on the chronological occurrence of the latest workload where the configuration is preserved (line 10). Configurations associated with more recent workloads are presumed to have integrated prior knowledge and are thus given higher timeliness weights:

$$w_{c,t} = \frac{S_c}{H}, \tag{5}$$

where S_c is the sequential number of the latest workload that the configuration c is associated with, indicating the most recent (largest) order in which the configuration appears across the past workloads.

Since both $w_{c,r}$ and $w_{c,t}$ range between 0 and 1, the quality weight of configuration c is then computed as:

$$w_c = w_{c,r} + w_{c,t} \tag{6}$$

In the end, we stochastically select N configurations according to w_c for seeding, where a greater value of w_c stands a higher probability of being favored. This, compared with selecting them deterministically, still retains a low possibility of selecting "less useful" configurations for seeding, hence maintaining diversity to escape from the local optima.

Notably, when there is exactly one previous workload, only the first stage would work as the number of configurations to be chosen ($\frac{N}{2}$) is smaller than the number required (N). The remaining $\frac{N}{2}$ configurations are then randomly generated. Of course, there will be no seeding under the very first workload.

C. Evolutionary Planning

As mentioned, DLiSA works the best with using evolutionary algorithms for planning because (1) they are based on population which fits well with the seeding—it caters to a set of configurations instead of one; (2) they have been widely studied in SBSE/self-adaptation [43], [45]. In this work, we employ Genetic Algorithm (GA) [46] for seeded planning. In a nutshell, GA works by iteratively reproducing from promising configurations via crossover and mutation, as evaluated on the Cyber-Twin (see Section IV-C), to evolve into even better ones. We adopt elitist-based GA where only the top-performing configurations are preserved in each iteration. Since GA has also been used for tuning configurable systems, interested readers can refer to prior work for a more detailed elaboration [11], [19], [47].

TABLE II: Subject system characteristics. The details of their workloads can be found in our repository: https://github.com/ideas-labo/dlisa.

System	Lang.	Domain	Perf.	Version	#O	#C	#W
JUMP3R [49]	Java	Audio Encoder	Runtime	1.0.4	16	4196	6
KANZI [50]	Java	File Compressor	Runtime	1.9	24	4112	9
DCONVERT [51]	Java	Image Scaling	Runtime	1.0.0-α7	18	6764	12
H2 [52]	Java	Database	Throughput	1.4.200	16	1954	8
BATLIK [53]	Java	SVG Rasterizer	Runtime	1.14	10	1919	11
XZ [54]	C/C++	File Compressor	Runtime	5.2.0	33	1999	13
LRZIP [55]	C/C++	File Compressor	Runtime	0.651	11	190	13
X264 [56]	C/C++	Video Encoder	Runtime	baee400...	25	3113	9
Z3 [57]	C/C++	SMT Solver	Runtime	4.8.14	12	1011	12

#O: No. of options; **#C:** No. of configurations; **#W:** No. of workloads tested.

IV. Experimental Setup

We experimentally assess the performance of DLiSA by unraveling the following research questions (RQs):

- **RQ1:** How effective is DLiSA against state-of-the-art approaches?
- **RQ2:** How efficient is DLiSA compared with others?
- **RQ3:** What benefits do ranked workload similarity analysis and weighted configuration seeding each provide?
- **RQ4:** How does α affect DLiSA's performance?

All experiments are run in a Python environment on MacOS with a quad-core 1.4 GHz CPU and 8GB RAM.

A. Subject Systems, Workloads, and Configurations

1) Systems: We follow all the systems from a prior empirical study [9] as our subjects, which investigates a range of widely studied configurable systems [36]–[38]. Our selection aligns with this study to ensure consistency and comprehensiveness. Specifically, these systems are carefully chosen to span a variety of application domains, performance objectives, and programming languages, including both Java and C/C++, thereby providing a solid foundation for our investigation into systems with diverse characteristics, as shown in Table II. More details on how to use these systems for conducting experiments can be found in [9], [48].

2) Workloads: The workloads we studied are diverse and domain-specific. For example, for SMT solver Z3, the workload can be different SMT instances that are of diverse complexity, e.g., QF_RDL_orb08 and QF_UF_PEQ018; for database system H2, the workloads are requests with different rates and types (e.g., read-only or read-write), such as tpcc-2 and ycsb-2400. In this study, we use the same various workloads as in [9], which range from 6 to 13 depending on the systems (denoted as W1, W2,..., W13). Self-adaptations are triggered as those different workloads arrive at the system

in a certain order. Here, we randomly shuffle the order of all workloads to arrive at a system and test self-adaptation therein.

3) Configurations: Each system in our study features a distinct configuration space, covering different option types (e.g., integers, boolean, and enumerates) and dimensions.

Overall, these diverse systems and workloads provide a robust foundation for assessing the efficacy and efficiency of our approach across different contexts and configurations.

B. Compared Adaptation Approaches

For the comparative analysis in our study, we compare DLiSA with the following state-of-the-art approaches:

- **FEMOSAA**[2] **(Stationary Adaptation)** [30]: The approach responds to workload changes by triggering a new search from scratch with randomly initialized configurations.
- **Seed-EA (Dynamic Adaptation)** [20]: By using an evolutionary algorithm, the approach seeds all the configurations preserved from the most recent past workload for planning under the new workload.
- **D-SOGA (Mixed Adaptation):** As a single-objective variant of D-NSGA-II [40], this approach retains 80% randomly chosen configurations from the most recent past workload with 20% new randomly initialized configurations to preserve diversity when the workload change.
- **LiDOS (Dynamic Adaptation)** [11]: The approach transforms single-objective problems into multi-objective ones via an auxiliary objective, thereby leveraging non-dominance relations to retain local optimal configurations under the most recent past workload to be seeded for the new workload.

C. Component and Parameter Settings

For a fair comparison across all approaches, the parameters of all stochastic search algorithms in the planning are standardized, where binary tournament is employed for mating selection, together with the boundary mutation and single-point crossover. The mutation and crossover rates are set at 0.1 and 0.9, respectively, with a population size of 20, which is widely used in prior works [5], [30]. For DLiSA, we set its only parameter $\alpha = 0.3$, unless otherwise stated, as this tends to be the generally best setting (see Section V-D).

In this study, we use Cyber-Twin to mimic the behaviors of the managed systems, which aims to expedite configuration evaluation in planning by using less time/resources without interfering with the managed system. There are different ways to create such a Cyber-Twin [58], e.g., (1) building a data driven surrogate model; (2) using existing benchmarks; or (3) creating a low-cost simulator/replica. Here, we chose existing benchmarks from [9] as the Cyber-Twin for all the compared adaptation approaches in experiments, which is straightforward and easy to implement, providing reliable performance data for accurate configuration measurements [11], [35]. Particularly, the budget constraint is 80 measurements (i.e., $R_t = 80$), which is sufficient for the approaches to converge while allowing them to timely self-adapt the system. The achieved performance at the end of planning is recorded. For the purpose of a controlled experiment, the workload would change shortly after the best configuration from the previous planning has been used to adapt the system.

In the search/optimization procedure of the planning under a particular workload, for all approaches, we do not explore duplicate configurations, i.e., only the newly evaluated/measured configurations from the Cyber-Twin would consume the budget. When the planning finds invalid configurations, we give a purposely worsened performance to those configurations, hence they would naturally be ruled out during the search/optimization process.

To obtain a statistically sound comparison, all experiments in this study are run 100 times independently. In particular, to test the self-adaptation of the systems, each of the runs follows a randomly shuffled order of the workloads, allowing us to alleviate the bias introduced by a specific occurrence sequence of the workloads. The performance results under each workload are used but it might appear at a different position across the repeated runs.

D. Statistical Validation

We employed different statistical validation for examining the 100 runs of results.

1) Pairwise Comparisons: For this, we use the following:

- **Non-parametric test:** We use Wilcoxon rank-sum test—a common test widely used in SBSE for its strong statistical power on pairwise comparisons [59]. The significance level over 100 runs is set at 0.05 and $p < 0.05$ indicates significant performance differences in the comparison.
- **Effect size:** In addition to statistical significance, we use $\hat{A}_{12}$ to measure the effect size [60]. Particularly, $\hat{A}_{12} \leq 0.44$ or $\hat{A}_{12} \geq 0.56$ suggests a non-trivial effect.

Thus, we say a comparison is statistically significant only if it has $\hat{A}_{12} \geq 0.56$ (or $\hat{A}_{12} \leq 0.44$) and $p < 0.05$.

2) Three or More Comparisons: We leverage the Scott-Knott test [61] to compare multiple approaches. In a nutshell, it first ranks the approaches based on the mean performance scores and then iteratively partitions this ordered list into statistically distinct subgroups. These subgroups are determined by maximizing the inter-group mean square difference Δ and their effect sizes. For example, for three approaches A, B, and C, the Scott-Knott test may yield two groups: {A, B} with rank 1 and {C} with rank 2, meaning that A and B are statistically similar but they are both significantly better than C.

V. Results and Analysis

A. RQ1: Effectiveness

1) Method: To answer **RQ1**, we compare DLiSA with four state-of-the-art approaches discussed in Section IV-B. We aggregate and scrutinize the best-performing configurations from 100 independent runs (each with randomly ordered workloads) under every workload, across a total of 93 cases (9 systems and each with 6 to 13 workloads). We also use the

[2]We use the single-objective version and pair it with GA.

TABLE III: The Mean and Standard deviation (Std) of performance objectives between DLiSA and other state-of-the-art approaches for all cases over 100 runs. For each case, green cells mean DLiSA has the best mean performance; or red cells otherwise. The one(s) with the best rank (r) from the Scott-Knott test is highlighted in bold.

Workload	Approach	LRZIP		XZ		Z3		DCONVERT		BATLIK		KANZI		X264		H2		JUMP3R	
		r	Mean (Std)	r	Mean (Std)	r	Mean (Std)	r	Mean (Std)	r	Mean (Std)	r	Mean (Std)	r	Mean (Std)	r	Mean (Std)	r	Mean (Std)
W1	FEMOSAA	2	3.214 (0.127)	3	4.931 (1.445)	**1**	**5.881 (0.131)**	4	1.952 (0.147)	3	0.953 (0.043)	3	1.830 (1.330)	3	1.038 (0.299)	3	25641.195 (1612.807)	2	2.979 (1.030)
	Seed-EA	2	3.134 (0.034)	2	4.636 (1.427)	2	5.898 (0.317)	3	1.867 (0.124)	2	0.912 (0.031)	2	1.346 (1.286)	2	0.937 (0.190)	2	26395.457 (1332.090)	2	2.636 (0.836)
	D-SOGA	**1**	**3.132 (0.015)**	**1**	**4.486 (1.251)**	2	5.951 (0.953)	**1**	**1.844 (0.093)**	2	0.912 (0.024)	2	1.252 (1.078)	2	0.933 (0.189)	2	26485.351 (1090.741)	2	2.666 (0.848)
	LiDOS	2	3.136 (0.032)	2	4.924 (1.986)	3	5.993 (0.628)	4	1.892 (0.135)	2	0.916 (0.025)	2	1.386 (1.252)	2	0.965 (0.195)	2	26287.284 (1466.595)	2	2.665 (0.789)
	DLiSA	2	3.135 (0.035)	**1**	**3.813 (0.849)**	**1**	**5.856 (0.011)**	2	1.849 (0.105)	**1**	**0.907 (0.014)**	**1**	**0.986 (0.866)**	**1**	**0.890 (0.140)**	**1**	**26721.450 (705.601)**	**1**	**2.573 (0.828)**
W2	FEMOSAA	2	0.031 (0.002)	2	0.014 (0.005)	**1**	**1.77 (0.089)**	2	1.186 (0.071)	3	1.38 (0.037)	3	0.166 (0.048)	3	3.954 (0.842)	3	18273.524 (1026.898)	3	1.093 (0.314)
	Seed-EA	2	0.030 (0.000)	2	0.013 (0.005)	4	2.463 (0.668)	2	1.12 (0.061)	2	1.340 (0.020)	2	0.143 (0.041)	2	3.775 (0.806)	2	18573.352 (1625.003)	2	0.915 (0.279)
	D-SOGA	2	0.030 (0.000)	2	0.013 (0.005)	**1**	**2.051 (0.491)**	2	1.118 (0.056)	3	1.341 (0.019)	2	0.141 (0.033)	**1**	**3.751 (0.819)**	**1**	**18847.953 (865.318)**	2	0.944 (0.264)
	LiDOS	2	0.030 (0.000)	3	0.015 (0.006)	3	2.375 (0.616)	2	1.126 (0.062)	3	1.345 (0.028)	2	0.147 (0.045)	2	3.927 (0.845)	2	18488.896 (1651.846)	2	0.915 (0.232)
	DLiSA	**1**	**0.030 (0.000)**	**1**	**0.011 (0.003)**	2	2.254 (0.608)	**1**	**1.115 (0.049)**	**1**	**1.338 (0.019)**	**1**	**0.131 (0.032)**	**1**	**3.590 (0.567)**	**1**	**18972.982 (758.262)**	**1**	**0.846 (0.197)**
W3	FEMOSAA	3	3.335 (0.037)	2	4.804 (1.461)	2	0.629 (0.999)	3	0.384 (0.007)	2	4.326 (0.153)	**1**	**0.293 (0.12)**	3	1.534 (0.421)	3	908.110 (46.396)	3	1.629 (0.514)
	Seed-EA	**1**	**3.304 (0.013)**	**1**	**4.558 (1.412)**	2	0.417 (0.802)	2	0.375 (0.007)	2	4.197 (0.040)	3	0.458 (0.821)	2	1.374 (0.327)	**1**	**943.275 (53.605)**	2	1.375 (0.421)
	D-SOGA	3	3.311 (0.023)	**1**	**4.543 (1.394)**	**1**	**0.352 (0.611)**	3	0.376 (0.007)	2	4.199 (0.042)	3	0.322 (0.168)	2	1.400 (0.322)	**1**	**946.655 (43.526)**	3	1.441 (0.414)
	LiDOS	2	3.309 (0.019)	**1**	**4.641 (1.438)**	2	0.429 (0.875)	3	0.377 (0.008)	2	4.206 (0.056)	3	0.659 (0.706)	2	1.422 (0.348)	2	938.221 (54.261)	2	1.431 (0.372)
	DLiSA	2	3.305 (0.014)	**1**	**3.835 (0.966)**	2	0.364 (0.660)	**1**	**0.375 (0.008)**	**1**	**4.196 (0.056)**	2	0.308 (0.129)	**1**	**1.286 (0.248)**	**1**	**948.344 (38.602)**	**1**	**1.309 (0.368)**
W4	FEMOSAA	3	7.268 (0.227)	**1**	**13.679 (3.906)**	**1**	**2.375 (0.245)**	3	1.657 (0.081)	3	1.233 (0.030)	4	2.625 (1.845)	3	1.770 (0.438)	3	963.053 (92.521)	3	0.741 (0.157)
	Seed-EA	2	7.165 (0.088)	**1**	**13.133 (3.830)**	2	2.422 (0.274)	3	1.608 (0.071)	**1**	**1.191 (0.022)**	2	1.851 (1.789)	**1**	**1.702 (0.430)**	2	1012.475 (108.587)	2	0.685 (0.145)
	D-SOGA	2	7.170 (0.078)	**1**	**13.132 (4.012)**	3	2.468 (0.428)	**1**	**1.603 (0.063)**	3	1.195 (0.023)	2	1.598 (1.403)	**1**	**1.652 (0.347)**	2	1015.531 (86.929)	2	0.697 (0.136)
	LiDOS	3	7.171 (0.039)	2	14.171 (8.349)	2	2.384 (0.254)	3	1.622 (0.08)	3	1.200 (0.021)	3	1.963 (1.883)	2	1.757 (0.413)	2	1006.740 (104.145)	2	0.679 (0.105)
	DLiSA	**1**	**7.159 (0.032)**	**1**	**11.102 (2.730)**	**1**	**2.324 (0.150)**	2	1.605 (0.067)	2	1.193 (0.026)	**1**	**1.173 (0.697)**	**1**	**1.586 (0.236)**	**1**	**1032.006 (45.261)**	**1**	**0.642 (0.076)**
W5	FEMOSAA	3	33.581 (0.386)	2	14.266 (3.910)	2	3.339 (0.655)	2	0.522 (0.021)	3	2.492 (0.066)	3	1.884 (1.057)	3	4.073 (2.727)	2	46866.246 (3439.525)	3	1.446 (0.699)
	Seed-EA	**1**	**33.395 (0.016)**	**1**	**13.657 (4.411)**	2	3.180 (0.330)	**1**	**0.502 (0.015)**	2	2.409 (0.040)	2	1.281 (1.076)	2	3.414 (0.729)	**1**	**47332.765 (3793.023)**	2	1.136 (0.417)
	D-SOGA	2	33.397 (0.017)	**1**	**13.815 (3.875)**	2	3.172 (0.223)	2	0.503 (0.015)	2	2.413 (0.042)	2	1.182 (0.994)	2	3.438 (0.795)	**1**	**47491.315 (3459.523)**	2	1.210 (0.553)
	LiDOS	3	33.424 (0.148)	**1**	**14.018 (4.555)**	2	3.195 (0.342)	2	0.505 (0.016)	3	2.420 (0.041)	2	1.362 (1.152)	2	3.530 (0.808)	2	47021.762 (4273.182)	2	1.179 (0.44)
	DLiSA	2	33.421 (0.150)	**1**	**11.702 (3.297)**	**1**	**3.150 (0.111)**	2	0.503 (0.019)	**1**	**2.404 (0.036)**	**1**	**0.938 (0.604)**	**1**	**3.222 (0.514)**	**1**	**47835.194 (2491.758)**	**1**	**1.045 (0.246)**
W6	FEMOSAA	4	0.978 (0.012)	3	2.172 (0.634)	2	1.406 (0.228)	2	0.391 (0.013)	3	3.323 (0.157)	4	0.687 (0.448)	3	0.111 (0.018)	3	47199.317 (2383.084)	4	0.319 (0.045)
	Seed-EA	2	0.971 (0.003)	**1**	**1.926 (0.513)**	2	1.330 (0.135)	**1**	**0.375 (0.011)**	2	3.158 (0.059)	3	0.528 (0.406)	2	0.104 (0.014)	3	47446.104 (3798.389)	2	0.304 (0.028)
	D-SOGA	**1**	**0.97 (0.002)**	2	2.077 (0.696)	2	1.327 (0.137)	2	0.376 (0.012)	3	3.158 (0.049)	2	0.519 (0.363)	2	0.103 (0.012)	2	47844.701 (2876.854)	2	0.309 (0.033)
	LiDOS	3	0.972 (0.006)	**1**	**2.053 (0.724)**	2	1.337 (0.158)	2	0.376 (0.01)	3	3.170 (0.049)	3	0.581 (0.440)	2	0.105 (0.015)	3	47119.407 (4400.199)	3	0.310 (0.030)
	DLiSA	3	0.971 (0.003)	**1**	**1.638 (0.375)**	**1**	**1.322 (0.130)**	2	0.376 (0.011)	**1**	**3.152 (0.042)**	**1**	**0.433 (0.263)**	**1**	**0.100 (0.013)**	**1**	**48335.083 (488.968)**	**1**	**0.298 (0.018)**
W7	FEMOSAA	3	0.198 (0.005)	3	0.215 (0.023)	**1**	**0.278 (0.212)**	2	19.015 (3.692)	2	1.170 (0.036)	3	0.243 (0.106)	3	0.679 (0.206)	2	18499.348 (1805.899)		
	Seed-EA	**1**	**0.192 (0.004)**	**1**	**0.205 (0.023)**	**1**	**0.320 (0.504)**	2	17.485 (2.864)	**1**	**1.136 (0.016)**	2	0.210 (0.108)	2	0.606 (0.154)	**1**	**19979.516 (1662.321)**		
	D-SOGA	2	0.192 (0.004)	2	0.206 (0.019)	**1**	**0.265 (0.194)**	**1**	**17.269 (2.668)**	2	1.136 (0.013)	2	0.205 (0.104)	2	0.606 (0.161)	**1**	**19952.252 (1608.535)**		N/A
	LiDOS	3	0.193 (0.005)	2	0.214 (0.031)	**1**	**0.269 (0.253)**	2	18.333 (3.433)	2	1.139 (0.022)	2	0.212 (0.116)	2	0.632 (0.173)	**1**	**19878.876 (1701.235)**		
	DLiSA	3	0.192 (0.004)	**1**	**0.196 (0.015)**	**1**	**0.292 (0.458)**	**1**	**17.366 (2.734)**	2	1.137 (0.016)	**1**	**0.177 (0.078)**	**1**	**0.572 (0.110)**	**1**	**20037.040 (1584.735)**		
W8	FEMOSAA	3	10.966 (0.092)	**1**	**29.552 (8.366)**	3	8.751 (0.016)	3	1.057 (0.033)	3	7.249 (0.251)	4	5.386 (4.813)	3	0.162 (0.086)	2	26235.578 (2089.717)		
	Seed-EA	3	10.910 (0.043)	**1**	**28.616 (8.807)**	3	8.747 (0.015)	**1**	**1.030 (0.026)**	2	7.079 (0.112)	2	3.869 (4.276)	2	0.142 (0.029)	**1**	**27973.092 (2149.536)**		
	D-SOGA	2	10.909 (0.039)	**1**	**28.969 (8.508)**	3	8.806 (0.590)	2	1.032 (0.026)	3	7.084 (0.105)	2	3.227 (2.843)	2	0.143 (0.025)	**1**	**28147.533 (1915.831)**		N/A
	LiDOS	3	10.919 (0.048)	2	29.977 (9.191)	**1**	**8.746 (0.005)**	3	1.039 (0.029)	3	7.095 (0.093)	3	4.002 (4.353)	2	0.146 (0.032)	**1**	**27926.014 (2064.011)**		
	DLiSA	**1**	**10.907 (0.020)**	**1**	**23.789 (5.998)**	2	8.746 (0.005)	3	1.032 (0.027)	**1**	**7.076 (0.077)**	**1**	**2.347 (2.228)**	**1**	**0.133 (0.019)**	**1**	**28129.890 (1669.565)**		
W9	FEMOSAA	4	9.492 (0.478)	**1**	**27.128 (7.707)**	3	3.184 (0.005)	3	0.491 (0.015)	3	1.068 (0.018)	3	1.092 (0.695)	3	0.261 (0.050)				
	Seed-EA	**1**	**9.180 (0.282)**	**1**	**25.640 (7.097)**	2	3.181 (0.003)	2	0.474 (0.015)	**1**	**1.047 (0.01)**	2	0.852 (0.615)	**1**	**0.249 (0.038)**				
	D-SOGA	3	9.279 (0.321)	**1**	**26.330 (7.248)**	**1**	**3.181 (0.003)**	3	0.474 (0.014)	2	1.049 (0.012)	2	0.813 (0.583)	2	0.249 (0.039)		N/A		N/A
	LiDOS	4	9.358 (0.296)	2	27.838 (16.49)	3	3.182 (0.004)	3	0.477 (0.016)	2	1.049 (0.012)	2	0.912 (0.745)	2	0.253 (0.040)				
	DLiSA	2	9.197 (0.314)	**1**	**21.324 (5.188)**	3	3.181 (0.003)	**1**	**0.473 (0.014)**	2	1.051 (0.014)	**1**	**0.709 (0.585)**	**1**	**0.240 (0.031)**				
W10	FEMOSAA	3	5.595 (0.381)	**1**	**13.081 (3.964)**	**1**	**6.795 (0.232)**	2	1.441 (0.011)	3	1.145 (0.037)								
	Seed-EA	2	5.358 (0.233)	**1**	**12.952 (4.033)**	**1**	**6.822 (0.233)**	**1**	**1.439 (0.009)**	**1**	**1.116 (0.015)**								
	D-SOGA	3	5.429 (0.299)	**1**	**12.878 (3.446)**	**1**	**6.840 (0.241)**	**1**	**1.44 (0.008)**	3	1.117 (0.017)		N/A		N/A		N/A		N/A
	LiDOS	2	5.398 (0.276)	2	13.191 (4.421)	**1**	**6.845 (0.265)**	2	1.441 (0.009)	3	1.120 (0.018)								
	DLiSA	**1**	**5.358 (0.228)**	**1**	**10.605 (2.606)**	**1**	**6.816 (0.236)**	**1**	**1.438 (0.009)**	2	1.117 (0.017)								
W11	FEMOSAA	2	2.126 (0.047)	3	3.388 (0.990)	**1**	**8.248 (1.047)**	3	1.464 (0.02)	3	1.693 (0.053)								
	Seed-EA	2	2.092 (0.035)	2	3.254 (0.913)	2	8.410 (1.719)	**1**	**1.442 (0.018)**	**1**	**1.627 (0.035)**								
	D-SOGA	2	2.094 (0.029)	**1**	**3.190 (0.943)**	**1**	**8.232 (1.282)**	2	1.446 (0.019)	3	1.630 (0.032)		N/A		N/A		N/A		N/A
	LiDOS	2	2.093 (0.029)	2	3.311 (0.962)	**1**	**8.327 (1.378)**	3	1.447 (0.021)	3	1.636 (0.041)								
	DLiSA	**1**	**2.089 (0.022)**	**1**	**2.804 (0.775)**	**1**	**7.948 (0.654)**	2	1.444 (0.019)	2	1.628 (0.038)								
W12	FEMOSAA	3	3.547 (0.121)	3	7.159 (2.183)	3	3.989 (0.344)	2	0.49 (0.009)										
	Seed-EA	**1**	**3.474 (0.059)**	2	6.327 (1.822)	2	3.890 (0.063)	**1**	**0.487 (0.009)**										
	D-SOGA	2	3.488 (0.079)	2	6.541 (1.967)	2	3.898 (0.150)	2	0.487 (0.01)		N/A		N/A		N/A		N/A		N/A
	LiDOS	3	3.501 (0.154)	2	6.567 (2.086)	2	3.897 (0.148)	2	0.489 (0.01)										
	DLiSA	2	3.477 (0.065)	**1**	**5.341 (1.318)**	**1**	**3.878 (0.009)**	**1**	**0.487 (0.007)**										
W13	FEMOSAA	3	2.546 (0.024)	2	3.621 (0.990)														
	Seed-EA	2	2.529 (0.018)	**1**	**3.440 (1.002)**														
	D-SOGA	**1**	**2.527 (0.016)**	**1**	**3.554 (0.992)**		N/A		N/A		N/A		N/A		N/A		N/A		N/A
	LiDOS	3	2.534 (0.019)	**1**	**3.612 (1.169)**														
	DLiSA	3	2.530 (0.018)	**1**	**2.939 (0.721)**														
Summary (r = 1)	FEMOSAA	**0**		**4**		**6**		**0**		**0**		**1**		**0**		**0**		**0**	
	Seed-EA	**5**		**9**		**2**		**6**		**5**		**0**		**2**		**4**		**0**	
	D-SOGA	**3**		**9**		**6**		**4**		**0**		**0**		**2**		**5**		**0**	
	LiDOS	**0**		**4**		**4**		**0**		**0**		**0**		**0**		**2**		**0**	
	DLiSA	**5**		**13**		**8**		**6**		**6**		**8**		**9**		**8**		**6**	

Scott-Knott test [61] for our analysis. All other settings are the same as discussed in Section IV-C.

2) Result: The experimental results are summarized in Table III. As can be seen, overall, DLiSA demonstrates superior performance, ranking first in 69 out of 93 cases, while FEMOSAA, Seed-EA, D-SOGA, and LiDOS achieve the best ranks in 11, 33, 29, and 10 out of 93 cases, respectively. Notably, within the 69 cases where DLiSA achieved first rank, it also realized the best performance values in 65 cases, highlighting the efficacy and robustness of DLiSA in self-adaptation. In particular, DLiSA achieves up to 2.29× enhancement compared with its counterparts (W8 of the KANZI).

The above efficacy of DLiSA lies in its knowledge distillation for seeding, tailored to the characteristics of changing workloads in configurable systems. This empowers DLiSA to judiciously and dynamically distill the most useful configurations and ignore the misleading ones to enhance planning or engage a conservative stance in case the past configurations are generally useless. Other approaches, in contrast, either ignore the valuable past knowledge or leverage it without catering to the noise, due to the stationary setting and the static knowledge exploitation strategy.

However, there are also some edge cases where other approaches are competitive to DLiSA. For instance, in LRZIP, DCONVERT, and BATLIK systems, the relatively high similarity across different workloads suggests that consistently effective configuration in one workload tends to perform well in others. This consistency favors the Seed-EA approach, which simply seeds all the configurations preserved in preceded planning without specific responses to workload changes. An interesting

TABLE IV: Comparing resource efficiency of DLiSA with respect to the state-of-the-art approaches. Detailed results can be found in our repository: https://github.com/ideas-labo/dlisa.

System	FEMOSAA				Seed-EA				D-SOGA				LiDOS			
	$s>1$	$s=1$	$0<s<1$	$s=$ N/A	$s>1$	$s=1$	$0<s<1$	$s=$ N/A	$s>1$	$s=1$	$0<s<1$	$s=$ N/A	$s>1$	$s=1$	$0<s<1$	$s=$ N/A
LRZIP	**13**	0	0	0	2	2	1	**8**	**6**	2	0	5	**10**	3	0	0
XZ	**13**	0	0	0	**13**	0	0	0	**13**	0	0	0	**13**	0	0	0
Z3	**8**	1	0	3	**8**	3	0	1	**6**	2	0	4	**7**	3	0	2
DCONVERT	**12**	0	0	0	1	**6**	0	5	1	**6**	0	5	**11**	0	0	1
BATLIK	**11**	0	0	0	2	4	0	**5**	3	**6**	0	2	**6**	4	0	1
KANZI	**8**	0	0	1	**9**	0	0	0	**9**	0	0	0	**9**	0	0	0
X264	**9**	0	0	0	**9**	0	0	0	**9**	0	0	0	**9**	0	0	0
H2	**8**	0	0	0	**7**	1	0	0	**5**	2	0	1	**8**	0	0	0
JUMP3R	**6**	0	0	0	**6**	0	0	0	**6**	0	0	0	**6**	0	0	0
Total	**88**	1	0	4	**57**	16	1	19	**58**	18	0	17	**79**	10	0	4
Range of s	$s \in [1, 2.16]$				$s \in (0, 2.22]$				$s \in [1, 2.05]$				$s \in [1, 2.05]$			

observation arises within Z3 system, in which D−SOGA exhibits relatively superior performance. This could be attributed to a possible moderate similarity across workloads, which allows the combination of historical insights and random configurations in D−SOGA to thrive.

Based on the above analysis, we can conclude that:

> ***RQ1:*** *DLiSA is effective as it is generally ranked better (in the statistical sense) than state-of-the-art in 74% cases (69 out of 93) with significant performance improvements of up to 2.29×.*

B. RQ2: Efficiency

1) Method: To evaluate the resource efficiency in **RQ2**, for each case out of the 93, we employ the following procedure:

- A baseline, b, is identified for each counterpart approach, representing the smallest number of measurements necessary for it to reach its best performance, denoted as T, averaging over 100 runs.
- For DLiSA, find the smallest number of measurements, denote as m, at which the average result of the performance (over 100 runs) is equivalent to or better than T.
- The speedup of DLiSA over a counterpart is reported as $s = \frac{b}{m}$, which is a common metric used in [5], [62].

If DLiSA is efficient, then we expect $s > 1$; $0 < s < 1$ and $s = 1$ means DLiSA has worse efficiency and they are equally efficient, respectively. We use $s =$ N/A to denote the case where DLiSA cannot achieve the T reached by its counterpart. All other settings are the same as **RQ1**.

2) Result: The results are depicted in Table IV, clearly, DLiSA consistently outperforms or equalling its counterparts in the majority of cases. Specifically, compared with FEMOSAA, DLiSA attained superior speedup in 88 cases (up to 2.16×) with equal efficiency in 1 case. When contrasted to Seed−EA, DLiSA excelled in 57 cases with a maximum of 2.22× speedup and matched in 16. For the comparisons with D−SOGA and LiDOS, DLiSA maintained a remarkable speedup within 58 and 79 (up to 2.05×) out of 93 cases, respectively. These observations illustrate DLiSA's ability to deliver robust performance in utilizing resources for self-adapting configurable systems. Therefore, we say:

> ***RQ2:*** *DLiSA is considerably more efficient than state-of-the-art approaches in the majority of the cases, achieving up to* $2.22\times$ *speedup.*

TABLE V: Comparing DLiSA against its two variants over 100 runs; "+", "=", and "−" respectively indicate DLiSA performing significantly better than, similarly to, or worse than the variants. Detailed results can be found in our repository: https://github.com/ideas-labo/dlisa.

System	DLiSA vs DLiSA-I	DLiSA vs DLiSA-II
LRZIP	5+/8=/0−	5+/8=/0−
XZ	13+/0=/0−	5+/8=/0−
Z3	0+/12=/0−	3+/8=/1−
DCONVERT	2+/10=/0−	5+/7=/0−
BATLIK	5+/6=/0−	10+/1=/0−
KANZI	8+/1=/0−	6+/3=/0−
X264	9+/0=/0−	0+/9=/0−
H2	2+/6=/0−	2+/6=/0−
JUMP3R	6+/0=/0−	3+/3=/0−
Total	**50+/43=/0−**	**39+/53=/1−**

C. RQ3: Ablation Analysis

1) Method: To understand which parts in the knowledge distillation of DLiSA work, in **RQ3**, we design two variants to compare with the original DLiSA over the 93 cases:

- **DLiSA-I:** We replace the weighted configuration seeding with a random seeding of preserved past configurations.
- **DLiSA-II:** We disable the ranked workload similarity analysis but randomly trigger the seeding of planning.

Since there are only pairwise comparisons, we leverage the Wilcoxon rank-sum test and $\hat{A}_{12}$ effect size across 100 runs.

2) Result: The results are summarized in Table V. Clearly, we see that DLiSA exhibits a remarkable improvement over its variants from the 93 cases: DLiSA wins DLiSA-I in 50 cases with 43 ties, reflecting the effectiveness of weighted configuration seeding. Against DLiSA-II, DLiSA wins in 39 cases; draws in 53 cases; and loses only in one case, indicating the usefulness of the workload similarity analysis. These results prove the benefit of seeding only when needed and the positive implication of considering the most useful configurations while excluding the misleading ones—all of which are specifically designed in our knowledge distillation according to the key characteristic of configurable systems under changing workloads discussed in Section II-C.

In light of these observations, we can conclude that:

> ***RQ3:*** *Each individual parts in the knowledge distillation in DLiSA contribute significantly to its superiority.*

D. RQ4: Sensitivity to α

1) Method: The parameter α determines the likelihood of triggering seeding, in **RQ4**, we examine the sensitivity of

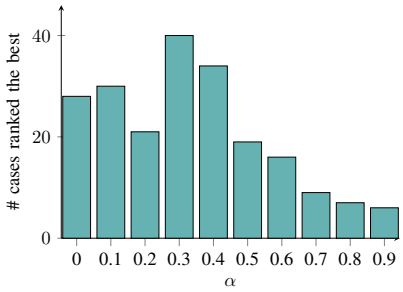
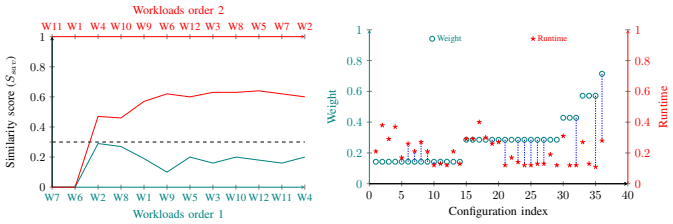


Fig. 4: The sensitivity of DLiSA to different α values.



(a) workload similarity (Z3) (b) configuration weights (KANZI)

Fig. 5: Examples illustrating the workload similarity and seeded configurations with respect to the weights. In (b), the dotted lines highlight the selected ones for seeding.

DLiSA to α by comparing the cases of $\alpha \in \{0, 0.1, ..., 0.9\}$. Again, we use the Scott-Knott to compare the results against different α values over 100 runs for all cases.

2) Result: As illustrated in Figure 4, DLiSA exhibits obviously superior performance when the α is set to 0.3 in terms of Scott-Knott ranks. We see that neither too small nor too large α is optimal. This is because, in the former case, there would be too many unnecessary seeding, making it difficult to eliminate the misleading information even with the weighted configuration seeding (e.g., $\alpha = 0$ means seeding constantly). In the latter case, it is simply due to the fact that it becomes rather difficult to trigger seeding, but instead rely merely on random initialization and hence waste the valuable accumulated knowledge. Clearly, the degradation of having larger α is more serious than setting it small, since not being able to seed is more influential than seeding misleading noises on the systems/workloads considered. Overall, we show that:

> ***RQ4:*** *Setting α to 0.3 yields the most effective performance for DLiSA, as it reaches a better balance between the benefit of seeding and seeding misleading information.*

VI. Discussion

A. How Workload Similarity Analysis Helps?

To understand why the ranked workload similarity analysis can help, Figure 5a shows the changing similarity scores on two exampled orders of the time-varying workloads for Z3. As can be seen, the similarity scores differ depending on the sequence of the emergent workloads—in some cases, they are higher than the threshold $\alpha = 0.3$ while in some other cases, they are lower. Such a discrepancy reflects the likelihood of seeding being beneficial: in the former cases, the seeding is constantly triggered because configurations found via the planning are sufficiently similar; while in the latter cases, randomly initialized configurations are used instead as seeding would likely be more harmful due to the misleading information caused by rather different landscapes between the workloads. In this way, DLiSA retains robust adaptability to diverse and changing workloads on configurable systems.

B. Why Weighted Configuration Seeding Work?

To demonstrate how configurations are weighted for knowledge distillation, Figure 5b visualizes the seeds extraction process for self-adaptation planning under workload artificl on the KANZI. The weights of distilled configurations selected for seeding and their performance under the current workload are connected by dashed lines, in which we see that configurations with higher weights are often selected, and they generally yield excellent performance (i.e., smaller runtime) than most of the remaining ones. Since we select the configurations stochastically based on the weights, we see that a small set of those with lower weights is also selected. Those lower weighted configurations, albeit slightly worse than the others on performance, help to prevent the selection of too many similar configurations for seeding. The above is what makes the weighted configuration effective in DLiSA.

C. What Are the Implications of DLiSA?

Lifelong self-adaptation (or self-evolving systems), as highlighted in prior work [11], [43], is an emerging paradigm for ensuring that systems can evolve autonomously under unanticipated changes. This study works along this direction from the perspective of seeding under changing workloads. It shows that the evolutionary planning that runs continuously for self-adaptation is beneficial. We have demonstrated the effectiveness of the two key contributions designed in DLiSA—*ranked workload similarity analysis* and *weighted configuration seeding*—in achieving lifelong self-adaptation for configurable systems: DLiSA considerably enhances system performance by identifying and leveraging useful historical knowledge while alleviating the impact of misleading information. These results hold substantial implications for the field of Software Engineering, as they support the development of more resilient and dependable software systems that make use of existing useful knowledge while filtering out useless knowledge at varying workloads. As such, we anticipate that our results will further advance the existing research in engineering self-adaptive configurable systems.

VII. Threats to Validity

Our investigation acknowledges the potential threats to **internal validity** associated with the parameter α, which we have set to 0.3. This choice is grounded in empirical evidence from our experiments in **RQ4**, where $\alpha = 0.3$ is a "rule-of-thumb" that yields generally favorable outcomes. We admit that the best value for α may differ on a system-by-system

basis and that exploring different settings for α might enhance the robustness of our results. For all experiments, we also use statistical tests and effect sizes to mitigate this threat.

Threats regarding **external validity** may arise from the specific configurable systems and workloads selected for our study. To mitigate potential biases, we included nine systems in our study. These systems span different domains, scales, and performance objectives, and we tested them across 93 unique workloads, following benchmarks set by prior studies [9]. This diverse selection aims to enhance the generalizability of our results, though expanding the range of systems examined could further deepen our insights.

VIII. Related Work

Here, we discuss the related work in light of DLiSA.

A. Stationary Adaptation for Configurable Systems

Stationary adaptation in self-adaptive configurable systems has been the cornerstone of strategies aiming at tuning the configurations under time-varying workloads [63]–[66]. Traditional approaches, such as those proposed by Chen *et al.* [30] and similar frameworks [14], [17] primarily focus on planning from scratch when changes in workloads are detected or at regular intervals, neglecting the accumulation of valuable historical insights. While this simplifies the optimization process, it may lead to repetitive effort and ineffective planning.

In contrast, our proposed DLiSA framework adopts a more holistic view that goes beyond the stationary paradigm. By harnessing the wealth of information available from past search experiences, DLiSA aims to construct a lifelong planning trajectory for the system. In this way, DLiSA sidesteps the inefficiencies of stationary planning, fostering a more intelligent optimization process that leverages historical information to facilitate future adaption planning.

B. Dynamic Adaptation for Configurable Systems

Dynamic adaptation is characterized by algorithms designed for planning continuously and adapting in real time. For example, Ramirez *et al.* [19] propose PLATO, a framework for adaptation planning using SOGA, which can automatically achieve adaptation planning by detecting the changes in fitness. Chen *et al.* [31] and Kinneer *et al.* [20] also use the concept of seeding to expedite the planning. This process, inspired by the principles of natural evolution and state preservation, is aimed at seamlessly self-adapting the workload changes, thereby achieving "lifelong optimization".

Although existing methods are dynamic in nature, they diverge from the true definition of dynamic optimization in the work [67]. This deviation stems from the static knowledge exploitation strategy, where all (or randomly chosen) configurations from the most recent past workload are seeded for the current one, while those from earlier workloads are simply discarded. In contrast, DLiSA introduces distilled knowledge seeding, a truly dynamic knowledge exploitation strategy, designed to navigate planning under time-varying workloads. Rather than naively assuming that only the configurations from the most recent past workload are useful, it proactively extracts the useful configurations for seeding from all past workloads while doing so only when it is necessary.

C. Control Theoretical Configuration Adaptation

Control theory has been recognized as an effective solution for the planning of configurable systems [21]–[23]. Among others, Maggio *et al.* [22] employ Kalman filters to refine and update the state values of the controller model, which are central to model predictive control schemes. Shevtsov and Weyns [23] expand on this by incorporating the simplex optimization method, which targets global optima in system states, thereby enhancing the precision of control mechanisms. The application of control theory in self-adaptation planning is promising, however, it faces significant challenges due to the complex, non-linear dynamics of real systems that can only be prescribed with advanced domain knowledge [68].

D. Configuration Performance Learning

Configuration performance learning for configurable systems is a distinct research trajectory that focuses on modeling the correlation between configuration and performance [69]. Several methods have been used, such as support-vector machines [70], decision trees [71], neural network [24], [72], [73], and ensemble learning [28], [73], [74], with the goal of crafting a function that accurately encapsulates the correlation between adaptation options and the performance of the target system. In contrast, DLiSA emphasizes optimization to self-adapting the changes of systems—a complementary aspect to performance learning [75].

IX. Conclusion

This paper proposes DLiSA, a distilled lifelong planning framework with ranked workload similarity analysis and weighted configuration seeding components for self-adapting configurable systems. The goal is to dynamically determine when it is generally more promising to seed with historical knowledge (***when to seed?***) and extract what knowledge should be redirected for planning without injecting misleading information (***what to seed?***). Empirical studies conducted on nine real-world configurable systems, spanning various domains and encompassing a total of 93 workloads, demonstrate that compared with state-of-the-art approaches, DLiSA is:

- **more effective,** as it achieves considerably better adaptation planning than its four competitors with up to 2.29×.
- **more efficient,** as it exhibits up to 2.22× speedup on producing promising configurations.

This work sheds light on the importance of automatically leveraging distilled past knowledge to self-adapt configurable systems in planning for future workloads. Looking ahead, we aim to delve into landscape analysis methods to better handle workload evolution and explore feedback mechanisms for more precise identification of beneficial planning information.

Acknowledgement

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

REFERENCES

[1] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "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*, 2015, pp. 307–319.

[2] T. Chen and M. Li, "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*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 453–465. [Online]. Available: https://doi.org/10.1145/3468264.3468555

[3] ——, "Do performance aspirations matter for guiding software configuration tuning? an empirical investigation under dual performance objectives," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 68:1–68:41, 2023. [Online]. Available: https://doi.org/10.1145/3571853

[4] ——, "Adapting multi-objectivized software configuration tuning," *Proceedings of ACM Software Engineering*, vol. 1, no. FSE, pp. 539–561, 2024. [Online]. Available: https://doi.org/10.1145/3643751

[5] P. Chen, T. Chen, and M. Li, "MMO: meta multi-objectivization for software configuration tuning," *IEEE Transactions on Software Engineering*, vol. 50, no. 6, pp. 1478–1504, 2024. [Online]. Available: https://doi.org/10.1109/TSE.2024.3388910

[6] D. Weyns, R. Calinescu, R. Mirandola, K. Tei, M. Acosta, A. Bennaceur, N. Boltz, T. Bures, J. Camara, A. Diaconescu *et al.*, "Towards a research agenda for understanding and managing uncertainty in self-adaptive systems," *ACM SIGSOFT Software Engineering Notes*, vol. 48, no. 4, pp. 20–36, 2023.

[7] J. García-Galán, L. Pasquale, G. Grispos, and B. Nuseibeh, "Towards adaptive compliance," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2016, pp. 108–114.

[8] L. Lesoil, M. Acher, X. Tërnava, A. Blouin, and J.-M. Jézéquel, "The interplay of compile-time and run-time options for performance prediction," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A*, 2021, pp. 100–111.

[9] C. S. Mühlbauer, F. Sattler, and N. Siegmund, "Analyzing the impact of workloads on modeling the performance of configurable software systems," in *Proceedings of the International Conference on Software Engineering (ICSE), IEEE*, 2023.

[10] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM transactions on autonomous and adaptive systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[11] T. Chen, "Lifelong dynamic optimization for self-adaptive systems: fact or fiction?" in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 78–89.

[12] ——, "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*, B. R. Schmerl, M. Maggio, and J. Cámara, Eds. ACM/IEEE, 2022, pp. 84–90. [Online]. Available: https://doi.org/10.1145/3524844.3528060

[13] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer learning for improving model predictions in highly configurable software," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017, pp. 31–41.

[14] T. Chen, R. Bahsoon, and X. Yao, "Synergizing domain expertise with self-awareness in software systems: A patternized architecture guideline," *Proceedings of the IEEE*, vol. 108, no. 7, pp. 1094–1126, 2020.

[15] T. Chen, R. Bahsoon, S. Wang, and X. Yao, "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*, 2018, pp. 48–55.

[16] T. Chen, M. Li, K. Li, and K. Deb, "Search-based software engineering for self-adaptive systems: Survey, disappointments, suggestions and opportunities," *arXiv preprint arXiv:2001.08236*, 2020.

[17] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 7–16.

[18] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, pp. 785–831, 2018.

[19] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley, "Applying genetic algorithms to decision making in autonomic computing systems," in *Proceedings of the 6th international conference on Autonomic computing*, 2009, pp. 97–106.

[20] C. Kinneer, D. Garlan, and C. L. Goues, "Information reuse and stochastic search: Managing uncertainty in self-* systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 15, no. 1, pp. 1–36, 2021.

[21] A. Filieri, H. Hoffmann, and M. Maggio, "Automated multi-objective control for self-adaptive software design," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 13–24.

[22] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Automated control of multiple software goals using multiple actuators," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 373–384.

[23] S. Shevtsov and D. Weyns, "Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 229–241.

[24] T. Chen and R. Bahsoon, "Self-adaptive and sensitivity-aware qos modeling for the cloud," in *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2013, pp. 43–52.

[25] H. Ha and H. Zhang, "Deepperf: Performance prediction for configurable software with deep sparse neural network," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1095–1106.

[26] T. Chen, R. Bahsoon, and X. Yao, "Online qos modeling in the cloud: A hybrid and adaptive multi-learners approach," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 327–336.

[27] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 497–508.

[28] T. Chen and R. Bahsoon, "Self-adaptive and online qos modeling for cloud-based software services," *IEEE Transactions on Software Engineering*, vol. 43, no. 5, pp. 453–475, 2016.

[29] Q. Zhang, I. Stefanakos, J. Camara Moreno, and R. Calinescu, "Towards lifelong social robot navigation in dynamic environments," 2023.

[30] T. Chen, K. Li, R. Bahsoon, and X. Yao, "Femosaa: Feature-guided and knee-driven multi-objective optimization for self-adaptive software," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 2, pp. 1–50, 2018.

[31] T. Chen, M. Li, and X. Yao, "On the effects of seeding strategies: a case for search-based multi-objective service composition," in *Proceedings of the genetic and evolutionary computation conference*, 2018, pp. 1419–1426.

[32] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[33] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[34] S. Wang, H. Hoffmann, and S. Lu, "Agilectrl: a self-adaptive framework for configuration tuning," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 459–471.

[35] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using FLASH," *IEEE Trans. Software Eng.*, vol. 46, no. 7, pp. 794–811, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2018.2870895

[36] J. Alves Pereira, M. Acher, H. Martin, and J.-M. Jézéquel, "Sampling effect on performance prediction of configurable systems: A case study," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, 2020, pp. 277–288.

[37] M. Velez, P. Jamshidi, F. Sattler, N. Siegmund, S. Apel, and C. Kästner, "Configcrusher: Towards white-box performance analysis for configurable systems," *Automated Software Engineering*, vol. 27, pp. 265–300, 2020.

[38] M. Weber, S. Apel, and N. Siegmund, "White-box performance-influence models: A profiling and learning approach," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1059–1071.

[39] R. Liu and L. Tahvildari, "Towards an uncertainty-aware adaptive decision engine for self-protecting software: an pomdp-based approach," *arXiv preprint arXiv:2308.02134*, 2023.

[40] K. Deb, U. B. Rao N, and S. Karthik, "Dynamic multi-objective optimization and decision-making using modified nsga-ii: A case study on hydro-thermal power scheduling," in *International conference on evolutionary multi-criterion optimization*. Springer, 2007, pp. 803–817.

[41] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 6, pp. 1–24, 2012. [Online]. Available: https://doi.org/10.1016/j.swevo.2012.05.001

[42] J. Ahlgren, K. Bojarczuk, S. Drossopoulou, I. Dvortsova, J. George, N. Gucevska, M. Harman, M. Lomeli, S. M. M. Lucas, E. Meijer, S. Omohundro, R. Rojas, S. Sapora, and N. Zhou, "Facebook's cyber-cyber and cyber-physical digital twins," in *EASE 2021: Evaluation and Assessment in Software Engineering, Trondheim, Norway, June 21-24, 2021*, R. Chitchyan, J. Li, B. Weber, and T. Yue, Eds. ACM, 2021, pp. 1–9. [Online]. Available: https://doi.org/10.1145/3463274.3463275

[43] D. Weyns, T. Bäck, R. Vidal, X. Yao, and A. N. Belbachir, "The vision of self-evolving computing systems," *J. Integr. Des. Process. Sci.*, vol. 26, no. 3-4, pp. 351–367, 2022. [Online]. Available: https://doi.org/10.3233/JID-220003

[44] Y. Li, Y. Shen, J. Jiang, J. Gao, C. Zhang, and B. Cui, "Mfes-hb: Efficient hyperband with multi-fidelity quality measurements," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8491–8500.

[45] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, 2012. [Online]. Available: https://doi.org/10.1145/2379776.2379787

[46] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.

[47] K. M. Bowers, E. M. Fredericks, and B. H. C. Cheng, "Automated optimization of weighted non-functional objectives in self-adaptive systems," in *Search-Based Software Engineering - 10th International Symposium, SSBSE 2018, Montpellier, France, September 8-9, 2018, Proceedings*, ser. Lecture Notes in Computer Science, T. E. Colanzi and P. McMinn, Eds., vol. 11036. Springer, 2018, pp. 182–197. [Online]. Available: https://doi.org/10.1007/978-3-319-99241-9_9

[48] "Experimental guideline," https://zenodo.org/records/7504284, retrieved on December 14, 2023.

[49] "The audio encoder: Jump3r," https://sourceforge.net/projects/jump3r/, retrieved on December 14, 2023.

[50] "The file compressor: Kanzi," https://github.com/flanglet/kanzi, retrieved on December 14, 2023.

[51] "The density image converter: Dconvert," https://github.com/patrickfav/density-converter, retrieved on December 14, 2023.

[52] "The database: H2," https://github.com/h2database/h2database, retrieved on December 14, 2023.

[53] "The svg rasterizer: Batlik," https://github.com/apache/xmlgraphics-batik, retrieved on December 14, 2023.

[54] "The file compressor: Xz," https://github.com/xz-mirror/xz, retrieved on December 14, 2023.

[55] "The file compressor: Lrzip," https://github.com/ckolivas/lrzip, retrieved on December 14, 2023.

[56] "The video encoder: X264," https://github.com/mirror/x264, retrieved on December 14, 2023.

[57] "The smt solver: Z3," https://github.com/Z3Prover/z3, retrieved on December 14, 2023.

[58] R. Eramo, F. Bordeleau, B. Combemale, M. van Den Brand, M. Wimmer, and A. Wortmann, "Conceptualizing digital twins," *IEEE Software*, vol. 39, no. 2, pp. 39–46, 2021.

[59] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd international conference on software engineering*, 2011, pp. 1–10.

[60] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[61] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Transactions on software engineering*, vol. 39, no. 4, pp. 537–551, 2012.

[62] Y. Gao, Y. Zhu, H. Zhang, H. Lin, and M. Yang, "Resource-guided configuration space reduction for deep learning models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 175–187.

[63] S. Kumar, T. Chen, R. Bahsoon, and R. Buyya, "Datesso: self-adapting service composition with debt-aware two levels constraint reasoning," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 96–107.

[64] S. Kumar, R. Bahsoon, T. Chen, K. Li, and R. Buyya, "Multi-tenant cloud service composition using evolutionary optimization," in *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*. IEEE, 2018, pp. 972–979.

[65] K. Li, Z. Xiang, T. Chen, and K. C. Tan, "Bilo-cpdp: Bi-level programming for automated model discovery in cross-project defect prediction," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 573–584.

[66] T. Chen and R. Bahsoon, "Self-adaptive trade-off decision making for autoscaling cloud-based services," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 618–632, 2015.

[67] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[68] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What does control theory bring to systems research?" *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 62–69, 2009.

[69] J. Gong and T. Chen, "Deep configuration performance learning: A systematic survey and taxonomy," *ACM Transactions on Software Engineering and Methodology*, 2024.

[70] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2013, pp. 11–20.

[71] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Faster discovery of faster system configurations with spectral learning," *Automated Software Engineering*, vol. 25, pp. 247–277, 2018.

[72] J. Gong and T. Chen, "Predicting configuration performance in multiple environments with sequential meta-learning," *Proceedings of ACM Software Engineering*, vol. 1, no. FSE, pp. 359–382, 2024. [Online]. Available: https://doi.org/10.1145/3643743

[73] ——, "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*, S. Chandra, K. Blincoe, and P. Tonella, Eds. ACM, 2023, pp. 858–870. [Online]. Available: https://doi.org/10.1145/3611643.3616334

[74] J. Gong, T. Chen, and R. Bahsoon, "Dividable configuration performance learning." *IEEE Transactions on Software Engineering*, 2024.

[75] P. Chen, J. Gong, and T. Chen, "Accuracy can lie: On the impact of surrogate model in configuration tuning," *IEEE Transactions on Software Engineering*, 2025.