Abstract—Software configuration tuning is essential for optimizing a given performance objective (e.g., minimizing latency). Yet, due to the software’s intrinsically complex configuration landscape and expensive measurement, there has been a rather mild success, particularly in preventing the search from being trapped in local optima. To address this issue, in this paper we take a different perspective. Instead of focusing on improving the optimizer, we work on the level of optimization model and propose a meta multi-objectivization (MMO) model that considers an auxiliary performance objective (e.g., throughput in addition to latency). What makes this model distinct is that we do not optimize the auxiliary performance objective, but rather use it to make similarly-performing while different configurations less comparable (i.e. Pareto nondominated to each other), thus preventing the search from being trapped in local optima. Importantly, by designing a new normalization method, we show how to effectively use the MMO model without worrying about its weight—the only yet highly sensitive parameter that can affect its effectiveness. Experiments on 22 cases from 11 real-world software systems/environments confirm that our MMO model with the new normalization performs better than its state-of-the-art single-objective counterparts on 82% cases while achieving up to 2.09× speedup. For 68% of the cases, the new normalization also enables the MMO model to outperform the instance when using it with the normalization from our prior FSE work under pre-tuned best weights, saving a great amount of resources which would be otherwise necessary to find a good weight. We also demonstrate that the MMO model with the new normalization can consolidate recent model-based tuning tools on 68% of the cases with up to 1.22× speedup in general.

Index Terms—Configuration tuning, performance optimization, search-based software engineering, multi-objectivization.

I. INTRODUCTION

ANY software systems are highly configurable, such that the configuration options can be flexibly adjusted for performance, including database systems, machine learning systems, and cloud systems, to name a few. For example, APACHE STORM, a stream processing system, can be tuned by changing some key configuration options such as splitters. However, a daunting number of configuration options will inevitably introduce a high risk of inappropriate or even poor software configurations set by software engineers. It has been reported that 59% of the software performance issues worldwide are related to ill-suited configuration rather than code [43]. In 2017-2018, configuration-related performance issues costed at least 400,000 USD per hour for 50% of the software companies.

Indeed, adjusting the configurations will affect the outcomes of different performance attributes, such as latency, throughput, and CPU load [23], [24], [25], [26], [70], [80]. However, there are many cases wherein only the optimization of a single performance attribute is of interest, whose minimization/maximization serves as a sole performance objective in consideration. For example, in the finance sector, a millisecond decrease in the trade delay may boost a high-speed firm’s earnings by about 100 million USD per year [90]. Another example is related to the machine learning systems deployed by large organizations (e.g., GPT-4 [72]), or those in the health care domain [3], where the concern is mainly on the accuracy, while caring little about the overhead/resource incurred for training. This has been well-echoed from the literature on software configuration tuning, in the majority of which only a single performance attribute is considered at a time [7], [8], [60], [61], [62], [71], [95], [99].

Despite only a single performance attribute being of concern, such an optimization scenario is not easy to deal with for any optimizer that tunes the software configuration. This is because

1) The configurable systems involve a daunting number of configuration options with complex interactions, rendering a black-box to the software engineers [18], [21], [22], [97].

2) The measurement of each configuration through running the software system is often expensive [51], hence exhaustively exploring every configuration is unrealistic.

3) There is generally a high degree of sparsity in the configurable software systems [19], [20], [70], i.e., similar configurations can also have radically different performance.

The last characteristic poses a particular challenge to the automatic software configuration tuning in finding the optimal configuration (performance), because firstly different

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
While avoiding being trapped in local optima, such methods require proposing a meta multi-objectivization (MMO) model for the optimizer, i.e., the "external" part of an optimizer. This is achieved by focusing on the "internal" components of the optimizer. They work on the configuration options such as random search [8], [71], [99], hill climbing [62], [95], genetic algorithm [7], [78], [80], [84], and simulated annealing [37], [42]. To seek the global optimum (best performance of the concerned performance attribute) while avoiding being trapped in a local optimum, such methods focus on the "internal" components of the optimizer. They work on designing novel search operators (i.e., improving the search operators of various optimizers), increasing the neighbourhood size of randomly mutated configurations [71], or developing various search strategies (i.e., the way to balance exploration and exploitation, for example, restarting the search in hill climbing [95]). However, a common limitation of such single-objective optimizers is that the goal to find the global optimum is "less oriented" as there is no clear "incentive" to encourage them to traverse the wide search space and locating as many local optima as possible, thus finding the best one in a resource-efficient manner.

To better mitigate the local optima, in this paper and our prior FSE’21 work [27] (we call it FSE work thereafter), we tackle this software configuration tuning problem from a different perspective. In contrast to the effort made by the existing works on the development of the optimizer, we work on the optimization model, i.e., the "external" part of an optimizer. This is achieved by proposing a meta multi-objectivization (MMO) model for this single-objective problem, to help the search avoid being trapped in local optima and progressively explore the entire objective space.

In a nutshell, MMO seeks to optimize two meta-objectives, each of which has two components. The first component of both meta-objectives is the target performance objective (e.g., latency), whereby only those configurations that perform well on the target objective being in favor. The second component, which is related to the other given auxiliary performance objective (e.g., throughput), is a completely conflicting term for the two meta-objectives. The reason for this design is that we hope to keep the target performance objective as a primary term in the model to preserve the tendency towards its optimality, but at the same time, we want the configurations with different values on the auxiliary performance objective to be incomparable. We are not interested in minimizing/maximizing the auxiliary performance objective since we do not know which value of it can lead to the best result on the target performance objective, but we wish to keep a good amount of configurations with diverse values of the auxiliary performance objective in the search, thus not being trapped in local optima (we will elaborate on this in Section III). The contributions from both this work and the FSE work are:

- Unlike existing work for the software configuration tuning which puts effort on the "internal part" of the optimization (i.e., improving the search operators of various optimizers), we work on the "external part"—multi-objectivizing this single-objective optimization scenario.
- We present a meta multi-objectivization model, MMO, as opposed to the existing multi-objectivization model considered in other SBSE scenarios which directly optimizes the target and auxiliary objectives simultaneously (referred to as plain multi-objectivization or PMO). We show, analytically and experimentally, why MMO is more suitable than PMO for software configuration tuning.

However, MMO requires a weight parameter to aggregate the target objective component and the auxiliary objective component. It is a critical parameter to balance searching for a good target performance objective value and maintaining diverse auxiliary performance objective values, requiring fine-tuning from the software engineers for every configurable software/environment, as done in our FSE work [27]. This, if done inappropriately, could lead to poor outcomes, as we will show in Section III-E. Yet, since the measurement of configurations is often expensive, finding the best weight in a case-by-case manner is not always realistic, which is a major threat to the applicability of the MMO model.

Therefore, in this paper, we also tackle this unwelcome issue. We show why the weight can be a highly sensitive parameter in the MMO model and propose a way to make the model weight-free without compromising the result. This is achieved by presenting a new normalization method, which is simple, but works very well—it leads to results that are even better than those of the FSE work under its best-tuned weight [27] for the majority of the problems. To sum up, the unique contributions of this paper are:

- A sound and formal analysis of the principle behind MMO, derived from the perspective of geometric transformation.

Fig. 1. A projected landscape of the performance objective Throughput with respect to configuration options Splitters and Counters for STORM under the WordCount benchmark. o is the global optimum and * denotes the local optima of throughput that an optimizer needs to escape from.
in the performance objective space, that explains its intention and what role the parameter $w$ and the normalization play therein. This then enables us to formally reflect on the limitation posed by the MMO model design proposed in the FSE work.

- Drawing on insights from the analysis, we design a new normalization method as part of the MMO model, capturing the bounds of both performance objectives adaptively. This allows us to keep the strengths and characteristics of the MMO model while removing the weight (i.e., setting $w = 1$ for all cases).
- An extensive evaluation that expands to 11 systems/environments that are of very different domains. Since a system comes with two performance objectives, each of these is used in turn as the target performance objective, leading to 22 cases. Under these cases, we compare MMO model using the new normalization with the PMO model and four single-objective counterparts, as well as with the MMO model using the normalization from the FSE work.
- An investigation on how our MMO model with the new normalization can consolidate FLASH [70] and BOCA [17], which are state-of-the-art model-based tuning methods for software configuration tuning.

Our experiment results are encouraging: we show that the MMO model with the new normalization achieves better results over the best single-objective counterpart and PMO (on 18 and 20 out of 22 cases wherein 14 and 15 of them are considerably better, respectively), while being much more resource-efficient overall (with up to $2.09 \times$ speedup over the single-objective optimizers and use significantly less resource than that of the PMO). In contrast to using the MMO model with the normalization from FSE work under its best weight, the MMO model with the new normalization shows better results on 15 cases (7 of which are significant) and competitive resource efficiency. Notably, this is achieved without the need of setting the weight, which can be undesirable as in 13 out of 22 cases, it requires at least 50% of the search budget as the extra resource to identify the best weight. The MMO model with the new normalization can also consolidate the model-based tuning methods like FLASH and BOCA; with minimal code change, both can be improved for 15 out of 22 cases (with 12 or 13 cases of statistically significant improvement) while having a $1.22 \times$ and $1.06 \times$ speedup in general, respectively.

To promote the open-science practice, a GitHub repository that contains all source code and data in this work can be accessed at: https://github.com/ideas-labo/mmo.

The rest of this paper is organized as follows. Section II introduces some background information. Section III elaborates on the design of the MMO and PMO model, as well as why and how we design the new normalization. Section IV presents our experiment methodology, followed by a detailed discussion of the results in Section V. Section VI delineates how to apply MMO in practical software engineering scenarios. The threats to validity are discussed in Section VII. Sections VIII and IX analyze the related work and conclude the paper, respectively.

II. PRELIMINARIES

In this section, we describe the necessary background information and context for this work.

A. Software Configuration Tuning Problem

A configurable software system often comes with a set of critical configuration options such that the $i$th option is denoted as $x_i$, which can be either a binary or integer variable, where $n$ is the total number of options. The search space, $\mathcal{X}$, is the Cartesian product of the possible values for all the $x_i$. Formally, when only a single performance concern is of interest (such as latency, throughput, or accuracy), the goal of software configuration tuning is to achieve $^2$:

$$\arg \min f(x), \ x \in \mathcal{X} \quad (1)$$

where $x = (x_1, x_2, ..., x_n)$. This is a classic single-objective optimization model and the measurement of $f$ is entirely case-dependent according to the target software and the corresponding performance attribute; thus we make no assumption about its characteristics.

B. Multi-Objectivization

Multi-objectivization is the method of transforming a single-objective optimization problem into a multi-objective one, in order to make the search easier to find the global optimum. It can be realized by adding a new objective (or several objectives) to the original objective or replacing the original objective with a set of objectives. The motivation is that since in a complex problem landscape, the search may get trapped in local optima when considering the original objective (due to the total order relation between solutions with respect to that objective), considering multiple objectives may make similarly-performed solutions incomparable (i.e., Pareto nondominated to each other), thus helping the search jump out of local optima [55].

Two solutions being Pareto nondominated means that one is better than the other on some objective and worse on some other objective. Formally, for two solutions $x$ and $y$, we call $x$ and $y$ nondominated to each other if $x \neq y \land x \not\succ y$, where $\not\succ$ is the negation of “to Pareto dominate” ($\prec$), the superiority relation between solutions for multi-objective optimization. That is, considering a minimization problem with $m$ objectives, $x$ is said to (Pareto) dominate $y$ (denoted as $x \prec y$) if $f_j(x) \leq f_j(y)$ for $1 \leq j \leq m$ and there exists at least one objective $j$ on which $f_j(x) < f_j(y)$. Pareto dominance is a partial order relation, and thus there typically exist multiple optimal solutions in multi-objective optimization. For a solution set $X$, a solution $x \in X$ is called Pareto optimal to $X$ if there is no solution in $X$ that dominates $x$. When $X$ is the collection of all feasible solutions for a multi-objective problem, $x$ becomes an optimal solution to the problem, and the set of all Pareto optimal solutions of the problem is called its Pareto optimal set.

$^2$Without loss of generality, we assume the performance objective to be minimized.
Multi-objectivization is not uncommon in the modern optimization realm, particularly to the evolutionary computation community [16, 50, 55, 87, 88]. To tackle various challenging single-objective optimization problems, researchers put much effort in introducing/designing additional objectives, e.g., creating sub-problems (sub-objectives) of the original objective [55], converting the constraints into an additional objective [16], constructing similar adjustable objectives [50], considering one of the decision variables [87], or even adding a man-made less relevant objective function [88].

III. MULTI-OBJECTIVIZATION FOR SOFTWARE CONFIGURATION TUNING

In this section, we present the designs of the multi-objectivization models and how they are derived from the key properties in software configuration tuning.

A. Properties in Configuration Tuning

We observed that, in general, software configuration tuning bears the following properties.

Property 1: As shown in Fig. 1 and what has already been reported [28], [29], [51], [70], the configuration landscape of different performance objectives for most configurable software systems is rather rugged with numerous local optima at varying slopes. Therefore the tuning, once the search is trapped at a local optimum, would be difficult to progress. This is because all the surrounding configurations of a local optimum are significantly inferior to it, and the search focus would have no much drive to move away from that local optimum (if only the concerned performance attribute is used to guide the search). As a result, a good optimization model has some additional “tricks” to avoid comparing configurations solely based on a single performance attribute.

Property 2: A single measurement of configuration is often expensive. For example, Valov et al. [92] reported that sampling all values of 11 configuration options for x264 needs 1,536 hours. This means that the resource (search budget) in software configuration tuning is highly valuable, hence utilizing them efficiently is critical.

Property 3: The correlation between different performance attributes is often uncertain, as different configurations may have different effects on distinct attributes. We observed that the configurations may achieve extremely good or bad performance on one while having similarly good results on the other, as illustrated in Fig. 2. Taking the system STORM with ROLLINGSORT benchmark (denoted STORM/RS) from Fig. 2 (left) as an example, suppose that in a multi-threaded and multi-core environment with 100 successful messages, if a configuration A enables each of these messages to be processed at 30ms, then the latency and throughput are

\[
\frac{100 \times 30}{100} = 30 \text{ms \ and \ } \frac{100}{30} = 3.33 \text{ msg/s},
\]

respectively. In contrast, another configuration B may restrict the parallelism (e.g., lower spout_num), hence there could be 50 messages processed at 20ms each while the other 50 are handled at 40ms each (including 20ms queuing time due to reduced parallelism). Here, the latency remains at \(\frac{100 \times 40}{100} = 40\) ms but the throughput is changed to \(\frac{100}{40} = 2.5\) msg/s, which is a 25% drop. Therefore, we should neither presume a strict conflict nor a harmonic correlation between the performance attributes.

Clearly, a good optimization model for software configuration tuning needs to take the above properties into account.

B. Plain Multi-Objectivization (PMO) Model

A straightforward idea to perform multi-objectivization is to add an auxiliary objective to optimize, along with optimizing the target performance objective. This is what has been commonly used in SBSE scenarios (e.g., [1], [15], [33], [69]). That PMO model can be formulated as:

\[
\begin{align*}
\text{minimize} & \quad f_a(x) \\
& \quad f_t(x)
\end{align*}
\]

(2)

where \(f_t(x)\) denotes the target performance objective (i.e., the concerned one) and \(f_a(x)\) denotes the auxiliary performance objective.

Putting it in the context of software configuration tuning, the PMO model may cover Property 1, because the natural Pareto relation with respect to the two objectives ensures that the target performance objective is no longer a sole indicator to guide the search. However, it does not fit Property 2 as PMO additionally optimizes the auxiliary performance objective. As such, configurations that perform well on the auxiliary performance objective but poorly on the target performance objective are still regarded as optimal in PMO, despite being meaningless to the original problem. This can result in a significant waste of resources. In addition, PMO does not consider Property 3 as it often assumes conflicting correlation between the two objectives [33], [68], which is hard to assure in software configuration tuning.

Footnote 3The relief of peak CPU load could allow the process of each message faster.
C. Meta Multi-Objectivization (MMO) Model

Unlike PMO, our meta multi-objectivization (MMO) model creates two meta-objects based on the performance attributes. The aim is to drive the search towards the optimum of the target performance objective and at the same time, not to be trapped in local optima. In particular, we want to achieve two goals:

— **Goal 1:** optimizing the target performance objective still plays a primary role, thus no resource waste on, for example, optimizing the auxiliary one (this fits in **Property 2**);
— **Goal 2:** but those with different values of the auxiliary performance objective are more likely to be incomparable (i.e., Pareto nondominated), hence the search would not be trapped in local optima (this relates to **Properties 1** and 3).

Formally, the MMO model with two meta-objects $g_1(x)$ and $g_2(x)$ is constructed as:

\[
\begin{align*}
\text{minimize } & \quad \begin{cases} g_1(x) = f_1(x) + w f_a(x) \\ g_2(x) = f_1(x) - w f_a(x) \end{cases} \\
\end{align*}
\]

whereby each of the two meta-objects shares the same target performance objective $f_1(x)$, but differs (effectively being opposite) regarding the auxiliary performance objective $f_a(x)$. The auxiliary objective can be a readily available one and whose result is of no interest (e.g., throughput or CPU load, in addition to latency). The weight $w$ is a critical parameter that balances the target and auxiliary performance objectives.

1) **Formal Analysis of MMO:** Compared with the original space of $f_1$ and $f_a$, MMO essentially does two main geometric operations to transform the original space of the two performance objectives into a meta-objective space: (1) it scales (stretches or shrinks) the configurations along the $f_a$ axis by a factor of $w$; (2) it rotates the scaled configurations by $45^\circ$ clockwise and then dilates them on both $f_1$ and $f_a$ by a factor of $\sqrt{2}$. Geometrically, MMO in Equation 3 can be decomposed via the following transformation metrics in linear algebra:

\[
\begin{align*}
\begin{bmatrix} g_1(x) \\ g_2(x) \end{bmatrix} = \sqrt{2} \begin{bmatrix} \cos \frac{\pi}{4} & \sin \frac{\pi}{4} \\ -\sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{bmatrix} \begin{bmatrix} f_1(x) \\ f_a(x) \end{bmatrix} \end{align*}
\]

whereby $\sin \frac{\pi}{4} = \frac{\sqrt{2}}{2}$ and $\cos \frac{\pi}{4} = \frac{\sqrt{2}}{2}$, hence the rotation angle is $\frac{\pi}{4}$ (i.e., $45^\circ$ clockwise) and both axes are dilated by a factor of $\sqrt{2}$ thereafter to create a rotation matrix of $1$ and $-1$.

To better understand how the transformation works in MMO, suppose that there are four configurations $A$, $B$, $C$, and $D$, where $A$ is one of the nondominated configurations, as shown in Fig. 3(a). The areas around $A$ can be divided into eight regions every $45^\circ$, starting counterclockwise from the region where $B$ is located and they are marked as $r_0$, $r_1$, ..., $r_7$, respectively. Note that since $f_a$ and $f_1$ are to be minimized in the original space of PMO (the same for $g_1$ and $g_2$ in the MMO space), the configurations that are dominated by $A$ will be those in its first quadrant, i.e., in regions $r_0$ and $r_1$. Therefore, we can precisely define the dominance relations between configurations and $A$ with respect to the regions of $A$ via the following:

- Configurations in $r_0$ and $r_1$ of $A$ (including the boundaries) will be dominated by $A$.
- Likewise, configurations in $r_4$ and $r_5$ of $A$ (including the boundaries) will dominate $A$ (not applicable if $A$ is Pareto optimal).
- Finally, configurations in $r_2$, $r_3$, $r_6$, and $r_7$ will be non-dominated to $A$, including the boundaries except those adjacent to $r_0$, $r_1$, $r_4$, and $r_5$.

Following the rotation in MMO, the configurations in the regions with respect to $A$ will also be rotated with $A$, and hence they would be in different regions compared with where
they were before, which might cause shifts in their relative dominance relationship to \( A \)—the key that makes MMO works effectively in tuning software configuration. Assuming that \( w = 1 \), i.e., no scaling (Fig. 3(b)) and hence we can focus on discussing the impact of rotation in MMO, from Fig. 3(c) it is not difficult to see that, compared to \( A \), all the configurations will be moved clockwise to their adjacent regions after the rotation of \( 45^\circ \). Using the case of \( A \) as an example again, configuration \( D \) in \( r_2 \) will be moved to \( r_1 \); \( C \) in \( r_1 \) will be moved to \( r_0 \); \( B \) in \( r_0 \) will be moved to \( r_7 \). As such, we can generalize the following rule:

**MMO Rule**

A configuration in a region \( r_i \) will be in a new region \( r_j \) following the rotation in MMO, and they satisfy the condition below:

\[
j = (i + 7) \mod 8
\]

Finally, \( f_a \) and \( f_t \) of the configurations are dilated by \( \sqrt{2} \) times for normalizing the coefficients of the rotation matrix to 1, and hence such dilatation has no effect on the dominance relationships between configurations.

From the above, it is intuitive to understand how the MMO can change the dominance relationships in the space: by moving the configurations between the regions with respect to a particular configuration, it is possible to make them change from dominated (comparable) to nondominated (incomparable), e.g., from \( r_0 \) to \( r_7 \); or vice versa, e.g., from \( r_2 \) to \( r_1 \), thereby altering the number of incomparable configurations (in the sense of Pareto dominance) during the tuning. Such an amendment of the Pareto dominance relationships determines the effectiveness of MMO, as either too many or too few incomparable configurations throughout the tuning would be harmful since the former causes a loss of search direction while the latter increases the possibility of being trapped at local optima.

Once it is clear how rotation affects the configurations in the MMO space, we can now explain the role of \( w \) in the MMO model. Since the value of weight \( w \) determines the factor of scaling, it is not hard to imagine that increasing \( w \) stretches all configurations horizontally on \( f_a \); conversely, decreasing \( w \) shrinks the configurations horizontally over \( f_a \). Fig. 4(a), 4(b), and 4(c) give a concrete example using the same configurations as before. When changing from \( w = 1 \) to \( w = 2 \), the \( f_a \) for all configurations are stretched by 2 times, causing configuration \( C \) to move from region \( r_1 \) to \( r_0 \) before the rotation (Fig. 4(b)). This means that the relative positions of configurations \( A, B, C, \) and \( D \) will rotate from Fig. 4(b) to 4(c). As such, \( C \), which should be dominated by \( A \) when rotating under \( w = 1 \), now will become nondominated to \( A \) after the rotation. In contrast,
if we change \( w = 1 \) to \( w = 0.5 \) (Fig. 4(d), 4(e), and 4(f)), the configurations will be shrunk on \( f_a \) by 0.5 times, where configuration \( B \) will be moved from regions \( r_0 \) to \( r_1 \) before rotation (Fig. 4(e)). As a result, the relative positions of configurations \( A, B, C, \) and \( D \) will rotate from Fig. 4(e) to 4(f). In this case, \( B \), which should be nondominated to \( A \) when rotating under \( w = 1 \), will now be dominated by \( A \) after the rotation.

The above indicates a simple rule to understand the role of \( w \) in MMO: a larger \( w \) suggests a bigger stretch on \( f_a \), making more of the configurations become incomparable following the rotation, which encourages the exploration in the search space to find more diverse configurations. In the extreme case where \( w = \infty \), the differences between configurations on \( f_a \) become infinitely large, making the two meta-objectives linearly conflicted, hence all configurations are incomparable if they differ on \( f_a \) and render the tuning with no guidance. On the other hand, a smaller \( w \) means a bigger shrink on \( f_a \), thus more configurations become comparable after the rotation, putting more emphasis on optimizing \( f_t \), i.e., exploitation. In the extreme case when \( w = 0 \), \( f_a \) is completely ruled out, leaving the two meta-objectives identical, and as such all configurations are comparable provided that they differ on \( f_t \), thereby making the tuning more difficult to jump out of the local optima. In general, neither too large nor too small \( w \) is ideal; yet, how large (or small) is considered as too large (or too small) is really case-dependent.

While here we use a nondominated configuration \( A \) as the example, it is worth noting that the analysis discussed thereof is applicable to any configurations. As such, the dominance relationships between any pair of configurations can be potentially changed by the scaling of \( w \) and rotation introduced in MMO. This as a whole would affect the behaviors and focus (exploration vs. exploitation) of the configuration tuning process regardless of the underlying multi-objective optimizer.

2) The Characteristics of MMO: To better understand the characteristics of the MMO model derived from our analysis and how the aforementioned two goals can be achieved, Fig. 5 gives an example of STORM on how it distinguishes between different configurations, in comparison with the PMO model, where we assume that latency is the target performance objective \( f_t \) and throughput is the auxiliary performance objective \( f_a \). Suppose that there is a set of four configurations \( A, B, C \) and \( D \). Let us say if we want to select two from them based on their fitness (e.g., in order to put some better configurations into the next-generation population in a multi-objective optimizer, such as NSGA-II). For the PMO model (Fig. 5(a)) that minimizes latency and maximizes throughput, the configuration \( D \), which performs extremely poor on latency, will certainly be selected by any multi-objective optimizer, since it is Pareto optimal and also less crowded than the other Pareto optimal configuration \( A \) and \( B \). In contrast, for the MMO model (Fig. 5(b)) which minimizes the two meta-objectives, the two configurations that will be selected are \( A \) and \( C \) (since they are the only two Pareto optimal ones).

It is worth noting that for the single-objective optimization model (which only considers latency), the two chosen configurations will be \( A \) and \( B \). However, since \( C \) and \( A \) behave much more differently than \( B \) and \( A \) on the throughput, it is more likely that they are located in distant regions in the configuration landscape; thus preserving \( C \) rather than \( B \) (when \( A \) is preserved) is generally more likely to help the search to escape from the local optimum.

In the following, we provide several remarks to help further grasp the characteristics of the MMO model.

Remark 1: The global optimum of the original single-objective problem (i.e., the configuration with the best target performance objective) is Pareto optimal (e.g., the configuration \( A \) in the example of Fig. 5). This can be derived immediately by contradiction from Equation (3) or the analysis of rotation from Section III-C1.

Remark 2: A similar but more general observation is that a configuration will never be dominated by another that has a worse target performance objective. That is, if configuration \( x_1 \) has a better target performance objective than \( x_2 \) (i.e., \( f_t(x_1) < f_t(x_2) \)), then whatever their auxiliary performance objective values are, \( x_2 \) will not be better than \( x_1 \) on both \( g_1 \) and \( g_2 \); in...
The value of the auxiliary performance objective corresponds to the configuration $B$ versus $C$ in Fig. 5. Indeed, according to the analysis from Section III-C1, there is no way for $x_1$ to be moved to the region $r_0$ or $r_1$ of $x_2$ as $x_1$ can only be in the $r_4$ to $r_7$ of $x_2$ before the rotation.

**Remark 3:** The above two remarks apply to the target performance objective, but not to the auxiliary performance objective. This is a key difference from the PMO model, where both objectives hold these remarks. An example of the consequence is the configuration $D$ of Fig. 5, which is meaningless to the original problem, but treated as being optimal in PMO and not in MMO.

**Remark 4:** Our MMO model does not bias to a higher or lower value on the auxiliary performance objective, in contrast to PMO. Indeed, as in Property 3, we do not know for certain what value of the auxiliary performance objective corresponds to the best target performance objective.

**Remark 5:** Configurations with dissimilar auxiliary performance objective values tend to be incomparable (i.e., nondominated to each other) even if one is fairly inferior to the other on the target performance objective. For example, the configuration $C$ in Fig. 5, which has worse latency than $A$, is not dominated by $A$ as their throughput are rather different. In contrast, the configuration $B$, which even has better latency than $C$, is dominated by $A$, as they are similar on throughput. This enables the model to keep exploring diverse promising configurations during the search, thereby a higher chance to find the global optimum.

**Remark 6:** If two configurations have the same value of auxiliary performance objective $f_a$, then they are always subject to the dominance relation (i.e., either dominating or being dominated). This is because, if configuration $x_1$ has the same $f_a$ value as $x_2$, then $x_1$ can only be on the boundary between $r_1$ and $r_2$ (or $r_3$ and $r_4$) for $x_2$, meaning that after the rotation, it can only be on the boundary between $r_0$ and $r_1$ (or $r_4$ and $r_5$) for $x_2$, in which case their relationships are always nondominated (comparable) regardless the $f_t$ value.

**Remark 7:** If two configurations have the same value of the target performance objective $f_t$, then they are always nondominated to each other in the MMO model. This is because, if configuration $x_1$ has the same $f_t$ value as $x_2$, then $x_1$ can only be on the boundary between $r_0$ and $r_3$ (or $r_4$ and $r_5$) for $x_2$, meaning that after the rotation, it can only be on the boundary between $r_7$ and $r_6$ (or $r_2$ and $r_3$) for $x_2$, in which case their relationships are always nondominated (incomparable) regardless the $f_a$ value and its scaling.

From Remarks 1–5, we can see that the MMO model is capable of focusing on optimizing the target performance objective (Goal 1) while mitigating the search from being trapped in local optima (Goal 2). In the following sections, on the basis of Remarks 6 and 7, together with the analysis from Section III-C1, we will explain why and how the weight parameter $w$ in the MMO model can be removed by changing the normalization method for the model.

### D. Normalization for MMO Model in the FSE Work [27]

The above analysis assumes an ideal scenario, i.e., the target and auxiliary performance objectives are of similar scale. This is, however, unrealistic for practically tuning configuration. For example, measuring the difference between latency results often reaches the magnitude of 5 order while CPU load merely differs at the scale of a few percentages. The consequence is that the appropriate $w$ value, which enables a good balance between exploitation and exploitation for MMO, can be either very large or very small depending on the case, leading to high difficulty in setting the $w$.

Fig. 6 gives an example. As can be seen, from Fig. 6(a), $f_t$ has roughly $100\times$ greater scale than that of $f_a$, and thereby from the coordinates, the configurations will be shrunk along $f_a$ to the boundary between $r_1$ and $r_2$ for $A$. This means that, after the rotation in MMO, $A$ will dominate all other configurations and make them comparable, which is harmful. To mitigate such, one would need to give a rather large $w$ value, i.e., $w = 100$.

---

**Fig. 6.** An illustration of the impact of the scale discrepancy between performance objectives and the anticipated normalization effect in MMO. $A$, $B$, $C$, and $D$ are four configurations with the auxiliary and target performance objective values as $(2 \times 10^2, 4)$, $(4 \times 10^2, 5.5)$, $(3 \times 10^2, 8)$, respectively. $\bigcirc$ and $\bigodot$ denote the configurations that will still be/become nondominated and dominated, respectively, after the rotation in MMO; $\bullet$ is the reference nondominated configuration.
that enables a more reasonable incomparability among the configurations, i.e., some are nondominated while some others are dominated by $A$ (Fig. 6(b)). Yet, since we do not normally have a precise understanding as to what extent the scales between different performance objectives differ beforehand, one would need to examine a wide range of possible $w$ values.

To ease the above, in the FSE work [27], we obtain more commensurable $f_i(x)$ and $f_o(x)$ via the following normalization (we call it FSE normalization thereafter):

$$f(x) = \frac{f^o(x) - f^o_{\text{lower}}}{f^o_{\text{upper}} - f^o_{\text{lower}}}$$

where $f^o(x)$ denotes the original value of the configuration $x$ on the performance objective $f$, and $f^o_{\text{lower}}$ and $f^o_{\text{upper}}$ are the global lower and upper bounds on that performance objective for the software, respectively. That is, the true scale of the performance objective is used as the bounds.

In practical software configuration tuning, however, $f^o_{\text{lower}}$ and $f^o_{\text{upper}}$ are likely to be unknown a priori. Therefore in our FSE work, these bounds are updated by using the maximum and minimum values discovered so far during the tuning to approximate the true scales. Note that using the true scales of the objectives (if known) or their close approximations for normalization is a widely used method in SBSE [4], [32], [80], [89], [98].

Essentially, normalization plays a similar role to $w$ in that they both scale the relative positions of the configurations (but the $w$ primarily works on $f_o$). As such, with the FSE normalization, our hope was that its resulting scaling could reduce the range of the ideal $w$ values, hence relieving the effort of adjusting it. In the best scenario, we anticipate that the configurations on $f_i$ and $f_o$ can be naturally scaled to ideal positions even when $w = 1$ (i.e., no scaling), thereby there is a good mix of incomparable and comparable configurations after the rotation, leading to more balanced exploitation and exploration, e.g., in Fig. 6(c).

### E. What Was Wrong?

Indeed, we have shown that the FSE normalization method can be effective in narrowing down the ideal range of $w$ to achieve superior results [27], but with one ineffective outcome: our analysis thereafter reveals that the weight $w$ in the MMO model remains a highly sensitive parameter, even within a narrower range, and finding the right setting for a system still requires much effort of trial and error. In [27] and this work (Section V), we examined a set of the weight settings for MMO model (i.e., $0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 10$)\(^6\). A key finding is that the weight achieving the best performance differs drastically on different configurable software systems: as shown in Fig. 7, some systems work better with a tiny weight value, e.g., $w = 0.01$ for the latency of STORM under the WORDCOUNT benchmark, while some others do best with a much bigger value, e.g., $w = 1$ for the energy usage of system X264. In what follows, we will explain what caused this issue that deviates from our original expectation by means of both theoretical analysis and empirical evidence.

1) An Analysis: The above occurrence is due to the severe discrepancy between the range of the current search population and the performance objectives’ scales in software configuration tuning, which obscures the benefit of the normalization schema we used for the FSE work. To provide a sound analysis thereof, recall the analysis on the effect of $w$ and the rotation from Section III-C1, using lower/upper bound in the normalization might lead to two cases in the presence of discrepant performance objective values:

Case 1: If the $f_o$ of the configurations in the population shrinks into a tiny range compared to its true objective scale in the whole search space (while the $f_i$ does not), then the $f_o$ in the population after the normalization (Equation 6) will be very close (see Remark 6). Fig. 8(a) shows an example, from which we see that without normalization, all configurations will be nondominated after rotation, i.e., the scale of $f_i$ (e.g., CPU load) is much smaller than that of $f_o$ (e.g., latency), which is not ideal. However, when the bounds for $f_o$ is $[0, 5000]$ while that for $f_i$ is $[0, 10]$, with the FSE normalization (Fig. 8(b)), all configurations (except $A$) will be shrunk towards the boundary between $r_1$ and $r_2$ of $A$ (or $r_5$ and $r_6$), meaning that they will become dominated (or dominate to) by $A$ after the rotation. This is also devastating to the tuning since too many comparable configurations will over-emphasize exploitation. To mitigate such, we need to set a larger $w$ for stretching $f_o$, i.e., in this case, $w = 5$ as shown in Fig. 8(c), thereby the number of incomparable configurations after rotation can be more reasonable to balance the exploration and exploitation.

Case 2: On the other hand, if the $f_i$ in the population evolves into a range that is tiny compared to its objective scale in the whole search space (while the $f_o$ does not), then the values of the $f_i$ of configurations in the current population after the normalization (Equation 6) will be very close (see Remark 7). As can be seen in Fig. 8(d), suppose that before normalization the scale of $f_o$ (e.g., CPU load) is much smaller than that of $f_i$ (e.g., latency), then this will cause $A$ to dominate all other configurations when rotating, which is harmful. With the FSE

\[^6\text{We chose these values because they are originally used in the FSE work and it is found that }w\text{ values outside }[0.01, 10]\text{ only degrade the results.}\]
Fig. 8. Illustration of why the MMO designs in our FSE work is highly sensitive to \( w \). A, B, C, and D are four configurations with the auxiliary and target performance objective values as \((2 \times 10^2, 4)\), \((4 \times 10^2, 5.5)\), \((3 \times 10^2, 8)\), and \((0.75, 7 \times 10^2)\) in (a), respectively. In (d), the values are \((2 \times 10^2, 4)\), \((4 \times 10^2, 5.5)\), \((3 \times 10^2, 8)\), and \((0.75, 7 \times 10^2)\), respectively. The format is the same as Fig. 6. (a), (b), and (c) show the case where the \( f_a \) is normalized into a tiny range with bounds of \([0, 10^5]\) as opposed to that of \([0, 10]\) for \( f_t \); (d), (e), and (f) demonstrate the case where the \( f_t \) is normalized into a tiny range with bounds of \([0, 10^5]\) as opposed to that of \([0, 10]\) for \( f_a \).

As a result, from the above, it is clear that although the FSE normalization helps to reduce the ideal ranges of \( w \) values (0.2 and 5 instead of the range on, e.g., \([0.001, 1000]\)), it can still negatively influence the appropriate number of incomparable configurations following the rotation in MMO, because a very large upper bound for one performance objective might be reached and being used throughout the tuning, even if such an extreme configuration has been ruled out later on. Therefore, we still need to non-trivially adjust \( w \) to mitigate such a side-effect from the FSE normalization, which collectively influences the effects of rotation. This is the key reason that the MMO designs in our FSE work remain highly sensitive to its only parameter \( w \).

2) Some Empirical Evidences: To demonstrate the devastating impact of FSE normalization on the MMO model, Fig. 9 illustrates an intermediate population of MMO model during the tuning for system LRZIP. As can be seen from Fig. 9(b), since the range of auxiliary performance objective in the population becomes “very small” (around \([1.03 \times 10^4 ms, 1.3 \times 10^4 ms]\)),
where the configurations are discriminative virtually based on key extension in this work. That is, we replace Equation 6 with translation as the basis of bounds in the normalization, which is the Fortunately, this can be fixed by considering the current population being either dominating or dominated by each other within the transformed space of our MMO model (Fig. 9(c)), which effectively means that the problem degenerates to the original single-objective problem, where the configurations are discriminative virtually based on their target performance objective (thus easily being trapped in local optima).

Fig. 10 gives yet another example, where we visualize an intermediate population of MMO model (with the FSE normalization) during the tuning for system STORM/WC. As can be seen in Fig. 10(b), since the range of the target performance objective (latency) in the population becomes "very small" (around [95ms, 1800ms]), compared to the objective scale ([3ms, 55209ms]), after the normalization the values of the target performance objective become tiny, condensing in the range of [0, 0.03] only. The auxiliary performance objective (throughput), in contrast, are more evenly spread over the range of [0, 1] after the normalization. As with our analysis, this can lead to all the configurations in the population being nondominated to each other within the transformed MMO space (Fig. 10(c)). Unfortunately, all configurations in the population being nondominated is detrimental to the search since there is no selection pressure (i.e., discriminative power); everyone is incomparable even the one with the best target performance objective.

F. A New Normalization

The above analysis and observations suggest that the FSE normalization based on the (approximate) true scales of the performance objectives may not be suitable for the MMO model. Fortunately, this can be fixed by considering the current population as the basis of bounds in the normalization, which is the key extension in this work. That is, we replace Equation 6 with the following:

\[
f(x) = \frac{f(x) - f_{\min}}{f_{\max} - f_{\min}}\]

(7)

where \(f(x)\) denotes the original value of the configuration \(x\) on the performance objective \(f\), and \(f_{\min}\) and \(f_{\max}\) are the maximum and minimum values of the current population on \(f\), respectively. As such, instead of using the global bounds throughout the search, the local bounds (in the population of configurations of every generation along with the evolution) are used in the normalization.

To better explain how the new normalization differs from the FSE normalization in the MMO space, Fig. 11 shows an example. Here, from Fig. 11(a) and 11(c), suppose that during the tuning a configuration \(E\) with a very large value of \(f_1\) is discovered, then this will cause both normalizations to shrink the configurations along \(f_1\), leaving a negative impact as most configurations will become nondominated (incomparable) after rotation. However, it is possible that \(E\) will be subsequently ruled out due to it being the only configuration dominated by \(A\) in the MMO space. Yet, with the FSE normalization (Fig. 11(b)), the bounds remain unchanged hence all configurations are still nondominated, i.e., the side effect left by \(E\) will remain present. In contrast, with the new normalization in this work (Fig. 11(d)), the bounds are updated locally within the population of preserved configurations and hence they can be scaled more reasonably, leading to a better mix of comparable and incomparable configurations after \(E\) is eliminated. This will strike a good balance between imposing the selection pressure toward the best target performance objective and preserving the diversity of the auxiliary performance objective.

Figs. 12 and 13 give the results of the examples from Figs. 9 and 10 after the new normalization is implemented, respectively. As can be seen, the configurations in the population after the normalization do not concentrate into one value on either objective (Figs. 13(b) and 12(b)), and in our MMO space there now exist both dominated and nondominated configurations in the population (Figs. 13(c) and 12(c)). In this case, there is less need to adjust the \(w\) in the MMO model to mitigate the side-effect of normalization, which balances the number of incomparable configurations after rotation, since the two performance objectives after the normalization are always commensurable. As such, we can generally remove the weight, i.e., setting \(w = 1\) for all cases.

G. Integrating With an Optimizer

Since MMO model is an optimization model, it can fit with different population-based multi-objective optimizers such as NSGA-II. A pseudo-code for using the MMO model with the normalization on top of NSGA-II has been demonstrated in Algorithm 1. As can be seen, there are two amendments required (the red crossed statements are the code for the FSE work and the green ones are the code changed in this work):

1) Keeping track of the bounds on \(f_1(x)\) and \(f_a(x)\) for normalizing both the target and auxiliary performance objectives (lines 6–7 and 23–24). The definitions of those bounds differ depending on the normalization methods, i.e., with lines 23–24 instead of lines 21–22, the bounds are locally restricted to the current population or otherwise, they would be the global bounds so far.

2) Performing the normal Pareto search procedure in NSGA-II within the transformed meta-objective space \((g_1(x)\) and \(g_2(x)\)) of MMO model without a weight, instead of the original target-auxiliary space \((f_1(x)\) and \(f_a(x)\)), as shown at lines 10, 15, 28, and 30.
Indeed, we do not need to make a significant amount of refactoring on MMO at the code level, but as we will show later on (Section V), such a simple change can lead to dramatic improvements in its effectiveness while saving the overhead of adjusting the weight. It is worth noting that, proposing a simple method that leads to large improvements is not easy, as this requires in-depth understanding and reasoning about the principles/causes behind the observations, as what we have shown in our theoretical and empirical analysis, which requires a large amount of effort.

IV. EXPERIMENTAL EVALUATION

In this section, we articulate the experimental methodology for evaluating our MMO model with the new normalization. To better distinguish this work and the FSE work [27], we use the following terminology:

- **MMO-FSE**: This refers to our MMO model with the normalization method from the FSE work.

## Algorithm 1: MMOONNSGA-II

**Input:** Configuration space \(V\); the system \(F\); weight \(w\)

**Output:** \(s_{best}\) the best configuration on \(f_k(x)\)

Declare: bound vectors \(z_{max}\) and \(z_{min}\)

1. Randomly initialize a population of \(n\) configurations \(P\)

2. \(z_{max} = \emptyset\), \(z_{min} = \emptyset\)

3. /* measuring \(f_k\) and \(f_0\) of the configurations in \(P\) on the system */

4. MEASURE\((P, F)\)

5. /* initializing the bounds for normalization */

6. \(z_{max} = \text{UPDATEUPPERBOUNDS}(P)\)

7. \(z_{min} = \text{UPDATELOWERRBOUNDS}(P)\)

8. /* updating \(g_1\) and \(g_2\) */

9. COMPUTEMMOModel\((P, z_{max}, z_{min})\)

10. while The search budget is not exhausted do

11. \(P' = \emptyset\)

12. while \(P' < n\) do

13. /* selecting parents based on \(g_1\) and \(g_2\) */

14. \(\{s_x, s_y\} \leftarrow \text{MATING}(P)\)

15. \(o_x, o_y \leftarrow \text{DOCROSSOVERANDMUTATION}(V, s_x, s_y)\)

16. /* measuring \(f_k\) and \(f_0\) for configurations \(o_x\) and \(o_y\) on the system (if unique) */

17. MEASURE\((o_x, o_y, F)\)

18. \(P' \leftarrow P' \cup \{o_x, o_y\}\)

19. /* the bounds are reset based on the current population at each generation regardless of the previous bound values */

20. \(z_{max} = \text{UPDATEUPPERBOUNDS}(P, z_{max})\)

21. \(z_{min} = \text{UPDATELOWERBOUNDS}(P, z_{min})\)

22. \(U' \leftarrow P' \cup U\)

23. /* updating \(g_1\) and \(g_2\) */

24. COMPUTEMMOModel\((U', z_{max}, z_{min})\)

25. /* sorting based on \(g_1\) and \(g_2\) */

26. \(U \leftarrow \text{top}\ n\ \text{configurations from}\ U'\)

32. return \(s_{best} \leftarrow \text{BESTCONFIGURATION}(P)\)
A. Research Questions

Our experiment answers a few research questions (RQs):

— **RQ1**: How effective is the MMO?

As the most fundamental question, we ask **RQ1** to verify whether our MMO can better help to mitigate the issue of local optima, i.e., by providing better results than the MMO-FSE, PMO, and state-of-the-art single-objective counterparts. However, even if the MMO can lead to promising results by mitigating local optima, it would be less useful if it requires a significantly large amount of resources to do so. Under the same settings as **RQ1**, our second research question is, therefore:

— **RQ2**: How resource-efficient is the MMO?

In **RQ2**, we are interested in examining whether the MMO can utilize the resource (the number of measurements) efficiently when reaching a certain level of performance.

One of the key novelties for MMO, compared with MMO-FSE, is weight-free. Yet, this would be meaningless if the MMO-FSE achieves similarly promising results over different weights on the systems studied; or if the effort for finding the best weight is trivial. Hence, our next **RQ** is:

— **RQ3**: How meaningful is the weight-free design in MMO?

**RQ3** seeks to understand two aspects: (1) how does MMO perform when compared with MMO-FSE under different weights; and (2) How much extra resource is required to tune the MMO-FSE for finding a promising weight value.

To reduce unnecessary noise, we investigate **RQ1-3** by directly measuring the systems, which belongs to the measurement-based tuning methods for software configuration tuning [71], [95], [102]. However, there exist studies leveraging on the model-based tuning methods where a surrogate is built to serve as a cheap evaluator to predict the performance of a configuration, under the assumption of the single-objective model. Since the key difference between measurement-based and model-based tuning methods lies in whether a surrogate is used to guide the search, the MMO, which itself is an optimization model, can be considered complementary to the model-based alternative. Therefore, our final research question is concerned with:

— **RQ4**: Can MMO consolidate the existing model-based tuning method?

To that end, we extend **FLASH** [70] and BOCA [17]—two recent tools from the Software Engineering community for configuration tuning—with our MMO and examine whether its performance can be improved.

B. Software Systems

To improve the generality of this work, we chose systems from existing studies according to the following criteria:

- To ensure complexity, we exclude simple systems, i.e., those with less than 10 configuration options and all of them are binary.
- The system should involve at least two performance objectives.
- To expedite the experiments, the system should contain readily available data of the measurements on all the valid configurations.
- To improve the diversity of the subject, for the system under different benchmarks, we use the one with the largest search space and the one with the highest deviation on the performance, providing that the above points are satisfied.

As shown in Table I, we experiment on 11 real-world software systems and environments that have been commonly used in prior work [51], [52], [67], [70], [75]. They come from diverse domains, e.g., SQL database, video encoding, and stream processing, while having different performance attributes, scale, and search space of valid configurations. Each software system has two performance objectives, which are chosen from prior work [51], [52], [67], [70], [75]. In all experiments, we use each of their two performance attributes as the target performance objective in turn while the other serves as the auxiliary performance objective, leading to 22 cases in total. We apply the same configuration options and their ranges as studied previously since those have been shown to be the key ones for the software systems under the related environment.

Noteworthily, it can be rather expensive even for a single measurement under those systems, e.g., it may take up to 341 seconds to measure a configuration on MONGODB. To ensure
realism and expedite the experiments, we use the datasets of those systems collected by existing work, in which each measurement is extracted from 3-5 repeats [51], [75].

### C. Settings for RQ1, RQ2, and RQ3

1) **Optimizers**: For the single-objective optimization model, we examine four state-of-the-art optimizers that are widely used in software configuration tuning, all of which deal with local optima in different ways:

- Random Search (RS) with a high neighbourhood radius to escape from the local optima, as used in [8], [71], [99].
- Stochastic Hill Climbing with restart (SHC-r), which is exploited by [62], [95], aiming to avoid local optima by using different starting points.
- Single-Objective Genetic Algorithm (SOGA) from [7], [78], [80], [84] that seeks to escape local optima by using variation operators.
- Simulated Annealing (SA) that tackles local optima by stochastically accepting inferior configurations as used in [37], [42].

While the MMO does not tie to any specific multi-objective optimizer, we use NSGA-II for the MMO, MMO-FSE, and PMO in this work, because (1) it has been predominately used for software configuration tuning in prior work when multiple performance attributes are of interest [26], [31], [58], [83], [85]; (2) it shares many similarities with the SOGA that we compare in this work. However, it is worth noting that MMO may not be able to work with some multi-objective optimizers designed for SBSE problems where the objectives are not treated equally, such as [45], [46], [73].

2) **Weight Values for MMO-FSE**: In our experiments, we evaluate a set of weight values, i.e., \( w \in \{0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 10\} \), for the MMO-FSE. Those are merely pragmatic settings, but we found that weight beyond this range only degraded the performance, please kindly refer to Section III-D for a theoretical explanation. Further, a similar setting is also what has been commonly followed for SBSE work in general [29]. In this way, we aim to examine whether the MMO-FSE can perform as well as MMO under the best weight chosen from a set of diverse weight values (or indeed worse on all of them).

3) **Search Budget**: In this work, we use the number of measurements to quantify the search budget and resource consumed, as it is language-/platform-independent and does not suffer from the interference caused by the background processes of the operating system.

Since one of our goals is to examine how badly a model/optimizer can suffer from trapping at undesired local optima when tuning software configuration, it is important to study the result under reasonable convergence, i.e., increasing the search budget is unlikely to change the outcomes. To that end, for every optimizer/model on each system (and its performance objectives), we examine different search budgets from \( \{100, 200, ..., X\} \) where \( X \) refers to the smaller one between 3,000 and the size of the search space. The purpose is to set a search budget as the smallest number of measurements for all optimizers/models, such that they all have less than 10% changes of configuration in the population (or no better configuration found when no population is involved) within the last 10% of the successive measurement count\(^7\). The settings are similar to those used by existing work which were found in a similar way, e.g., Gerasimou et al. [38] set no better configuration found for the last 20% of the iteration count as a sign of convergence; Krall et al. [56] use 15% as an indicator of convergence. Note that we additionally monitor the percentage of the population change rather than purely whether a better configuration is found, since Harman [44] suggests that a good sign of little realistic chance of further improvements on population-based optimizers is that the population has become homogeneous. To ensure the realism of the setting, we make sure that the actual time taken for exhausting the search budget does not exceed 48 hours for a run overall. The identified search budgets are then used as the termination criterion in our experiments, as shown in Table II. It is worth noting that the search budget identified remains much smaller than the corresponding search space. For example, it only allows for measuring 0.42% of the configurations for TRIMESH.

Since each measurement has considered the noise [51], [75] and only the profiling of systems is expensive in practice, in each run, we cached the measurements of distinct configurations, which can be reused directly when the same configuration appears again during the tuning. As such, only the distinct configurations would consume the budget.

To account for the stochastic nature of the optimizers, we repeat all experiments 50 runs under the search budget.

4) **Other Parameters**: For the other key parameters of the optimizers, we apply the binary tournament for mating selection, together with the boundary mutation and uniformed crossover in SOGA and NSGA-II, as used in prior work [26], [31], [80]. The mutation and crossover rates are set to 0.1 and 0.9, respectively, as commonly set in software configuration tuning [26].

What we could not decide easily is the population size for SOGA and NSGA-II. Therefore, for each software system, we additionally examine a set of population sizes, i.e., \( \{10, 20, ..., 100\} \), under the search budget identified previously. Similarly, we set the largest population size that can still ensure there are less than 10% changes of configurations at the last 10% of the measurement count. The results are shown in Table II. In this way, we seek to reach a good balance between convergence (smaller population change) and diversity (larger population size) under a budget.

---

\(^7\)For SOGA and NSGA-II, the population size is initially fixed to 10, which is the smallest size that we will examine subsequently.
Algorithm 2: FLASH

**Input:** Configuration space \( \mathcal{V} \); the system \( \mathcal{F} \)

**Output:** \( s_{\text{best}} \) the best configuration on \( f_t(x) \)

**Declare:** vector of surrogates \( \mathcal{M} \) (one for each performance objective)

1. Randomly initialize a size of \( k \) configurations \( \mathcal{P} \)
2. MEASURE(\( \mathcal{P}, \mathcal{F} \))
3. /* removing measured configurations */
4. \( V \leftarrow V - \mathcal{P} \)
5. while The search budget is not exhausted do
6. \( \mathcal{M} = \text{TRAINCARTS}(\mathcal{P}) \)
7. /* searching an estimated-best configuration for measurement */
8. \( o = \text{FINDBESTCONFIGURATION}(\mathcal{V}, \mathcal{M}) \)
9. MEASURE(\( o, \mathcal{F} \))
10. \( V \leftarrow V - o \)
11. \( \mathcal{P} \leftarrow \mathcal{P} + o \)
12. if \( o \) is measured to be better than \( s_{\text{best}} \) on \( f_t(x) \) then
13. \( s_{\text{best}} = o \)
14. return \( s_{\text{best}} \)

Algorithm 3: BOCA

**Input:** Configuration space \( \mathcal{V} \); the system \( \mathcal{F} \)

**Output:** \( s_{\text{best}} \) the best configuration on \( f_t(x) \)

**Declare:** vector of surrogates \( \mathcal{M} \)

1. Randomly initialize a size of \( k \) configurations \( \mathcal{P} \)
2. MEASURE(\( \mathcal{P}, \mathcal{F} \))
3. while The search budget is not exhausted do
4. \( \mathcal{M} = \text{TRAINRANDOMFOREST}(\mathcal{P}) \)
5. \( I = \text{GETALLSETTINGSONIMPORTANTOPTIONS}(\mathcal{M}, K) \)
6. for \( i \in I \) do
7. \( c = \text{DECAY}(j) \)
8. \( I = \text{GETIMPORTANTSETTINGS}(c) \)
9. \( \mathcal{P} = \text{COMBINESAMPLES}(I, \mathcal{M}) \)
10. /* searching an estimated-best configuration for measurement */
11. \( o = \text{FINDBESTCONFIGURATION}(\mathcal{P}) \)
12. MEASURE(\( o, \mathcal{F} \))
13. \( \mathcal{P} = \mathcal{P} + o \)
14. if \( o \) is measured to be better than \( s_{\text{best}} \) on \( f_t(x) \) then
15. \( s_{\text{best}} = o \)
16. return \( s_{\text{best}} \)

D. Settings for RQ4

1) Optimizers: For model-based tuning methods, we consider FLASH\(^8\) (TSE’20) [70] and BOCA (ICSE’21) [17] in this work, because

- they are recent efforts from the software engineering community to tune software configuration.
- they have been specifically tailored to cater for the key properties of the tuning problem, e.g., high sparsity and expensive measurements.
- their authors have shown that they outperform other more general configuration tuning approaches, e.g., BOCA is better than TPE [17] and FLASH is superior to c-PAL [70], as well as better than some older methods for software configuration tuning, e.g., the one by Jamshidi and Casale [70].
- both have been tested on some of the systems studied in this work, e.g., X264 and LLVM.

In a nutshell, FLASH was derived from the Sequential Model-Based Optimization (SMBO) paradigm, which is a generalization of the Bayesian Optimization (BO) [81]. As shown in Algorithm 2, the basic idea is to build a surrogate that learns the correlation between configurations and their values of a performance objective (line 6). Such a surrogate is then used to guide the search to decide which promising configuration to measure next via an acquisition function (line 8), after which the surrogate would be updated by using the newly measured configuration. Like other measurement-based tuning methods, the process terminates when the search budget is exhausted. Yet, unlike the classic BO, FLASH does two major changes for the problem to tune a single performance objective:  

1. The surrogate is a CART [14] instead of the classic Gaussian Process in BO [81].
2. The acquisition function no longer considers uncertainty but solely targets the best-predicted performance value.

Note that FLASH originally uses an exhaustive search to find the best-predicted configuration at each sampling iteration (line 8), but this may not be ideal for our study because of two reasons: (1) exhaustively traversing the whole configuration space itself is still a lengthy process, especially on some of the large systems. For example, on each iteration for TRIMESH, it can take several minutes on a standard machine to run even for a surrogate. (2) Since the surrogate is not always accurate [102], the exhaustive search could amplify the side effects caused by the errors in misleading the search. Therefore, we replace the exhaustive search with a random search, which works well and has been recommended as a replacement for SMBO [8].

Similar to FLASH, BOCA (Algorithm 3) also leverages Bayesian Optimization but it uses Random Forest [13] and Expected Improvement [53] as the surrogate and acquisition function, respectively. Further, BOCA takes the top \( K \) most important configuration options into account based on the rank from the Random Forest; it then creates a set of candidate configurations that cover \( c \) settings for the unimportant options combined with every setting of the important ones, where \( c \) is determined proportionally to the search progress (lines 5-9). The one with the best acquisition value from the set is then measured.

2) Search Budget: To ensure fairness, we set the same search budget as used in the original work of FLASH [70], i.e., 50 measurements. Further, we also use the same initial sample size (\( k = 30 \) in Algorithm 2) to pre-train the surrogate. As for the search process with the surrogate, we allow for 1,000 surrogate evaluations (including redundant ones) which is a typical setting from the other work for optimizing the surrogate when an exhaustive search is undesirable [54], [76].

Similar to RQ1-3, each experiment is repeated 50 runs.

E. Statistical Validation

We use the following methods for statistical test:

- Non-parametric test: To verify statistical significance, we leverage the Wilcoxon signed-rank test [94] (for
paired comparisons between two approaches) and Kruskal-Wallis test [66] (for multiple comparisons). In particular, to understand which pairwise comparisons are significant in the Kruskal-Wallis test, we use Dunn’s test [91] as the post-hoc method together with the Holm-Bonferroni correction [2], which will significantly reduce the chances of Type-I error. All of the above are widely used non-parametric test for SBSE and has been recommended in software engineering research for their strong statistical power [6]. The standard α = 0.05 is set as the significance level over 50 runs.

— Effect size: To ensure the differences are not generated from a trivial effect, we use $\hat{A}_{12}$ [93] to verify the effect size of the comparisons on target performance objectives over 50 runs. According to Vargha and Delaney [93], when comparing our MMO and its counterpart in this work, $A_{12} > 0.5$ denotes that the MMO is better for more than 50% of the times (MMO wins); MOO will lose if $\hat{A}_{12} < 0.5$ and it is a tie when $\hat{A}_{12} = 0.5$. In particular, $0.56 \leq \hat{A}_{12} < 0.64$ indicates a small effect size while $0.64 \leq \hat{A}_{12} < 0.71$ and $\hat{A}_{12} \geq 0.71$ mean a medium and a large effect size, respectively.

As such, we say a result of the comparison is statistically significant only if it has $\hat{A}_{12} \geq 0.56$ (or $\hat{A}_{12} \leq 0.44$) and $p < 0.05$ (after correction if needed).

V. Results

In this section, we present and discuss the experiment results. All code and data can be accessed at: https://github.com/idea-labo/mmo.

A. RQ1: Effectiveness

1) Method: To answer RQ1, we compare MMO with the best state-of-the-art single-objective counterparts (as discussed in Section IV-C1), as well as the MMO-FSE with a best-tuned weight and PMO, over all the 22 cases of study. Since the best single-objective optimizer (denoted as $SO_{best}$) and the MMO-FSE with the best weight (denoted as MMO-FSE$_{best}$) differ across the systems/objectives, we use the following procedure to select the best representative in each case:

1) Run all candidates under the full-scale experiment.
2) Rank the results using Scott-Knott test [79] according to the target performance objective.\footnote{Scott-Knott test is a widely used test in SBSE [96] to distinguish different approaches into clusters based on an indicator (target performance objective in this work), between each of which are guaranteed to have statistically significant differences; the approaches within the same cluster are said to be statistically similar. The clusters are then ranked.}
3) Select the one with the best rank; if there are multiple candidates under the best rank, the one with the best average (over 50 runs) on the target performance objective would be used.

To ensure statistical significance, the statistical test\footnote{Since there are multiple comparisons, we use Kruskal-Wallis test and the corrected $p$ values (via Holm-Bonferroni correction) of Dunn’s test for all the 3 comparisons between MMO and its counterpart.} and effect size are reported for every pairwise comparison between our MMO and the other counterparts over 50 runs.

$2)$ Findings: From Table III, we can see that MMO performs considerably better than the best single-objective counterpart $SO_{best}$ (which can vary depending on the case), winning 18 out of 22 cases within which 14 of them show statistical significance\footnote{All Kruskal-Wallis tests show $p < 0.001$ at the global level, hence the details are omitted for simplicity of exposition.} ($A_{12} \geq 0.56$ and $p < 0.05$); the remaining 4 cases are all tie and there are no cases of losses. The magnitudes of gains are also clear. The improvements over PMO are also clear: MMO wins 20 cases (15 have statistically significant differences) under mostly large magnitude of gains; there are also one tie and one loss.

When comparing to the MMO-FSE with the best weight (MMO-FSE$_{best}$), MMO wins 15 out of the 22 cases with 7 of them showing statistical significance; loses on 5 cases with no statistically significant ones, together with two ties. This means that, although MMO-FSE$_{best}$ is competitive, MMO can still obtain further improvement in general thanks to the new normalization method. This is especially true in some cases, such as STORM/RS-O2, where the target performance objective values are much more skewed than the auxiliary ones (recall from Fig. 10). Even though MMO-FSE$_{best}$ was pre-tuned with some best weights, such finding is not surprising because: firstly, despite that the range of good weights can be reduced compared with when no normalization is used, the given set of candidate weights may not be exhaustive. Indeed, as we will show in Section V-C, the weight tuning itself can be profoundly expensive, making exhaustive search unrealistic. As such, the chosen weight may still be far from the truly optimal weight setting. Secondly, as the population evolves, the objective values keep changing, particularly on the target performance objective. A fixed weight typically does not stay ideal during the entire evolution process. For example, the weight may be a good fit at the beginning of the evolution when the population has a relatively large range of the target performance objective values, but it may become unsuitable when the population converges into a tiny region with respect to the target performance objective.

To examine whether MMO can indeed improve the chances for reaching the global optimum of the target performance objective, in Fig. 14, we plot the average number of runs that each model/optimizer reaches the global optimum across all cases. We see that, as expected, MMO cannot find the global optimum for all runs under the systems studied. However, in general, it hits the global optimum more regularly than the others with statistical significance and large effect size.\footnote{We use the Wilcoxon signed-rank test here since the comparisons cut across the subject systems, i.e., they are paired.}

To understand why our MMO can outperform the state-of-the-arts, we took a closer look at the configurations explored during the runs. We identified two most common patterns shown in Fig. 15. As can be seen from Fig. 15(a), the first pattern is where MMO reaches the global optimum while the others do not; the second represents a run where the global optimum has never been found, but MMO produces a result that is much closer to it than that of the others, as shown in Fig. 15(b). It is worth noting that, under both patterns, there exist some large regions of local optima that cause the others to suffer more than MMO. This is evident by Fig. 15 where the highlighted local
optima regions are mostly crowded with points explored by the other counterparts. The MMO, in contrast, escapes from these local optima by exploring an even larger area while keeping the tendency towards better target performance objective, which is precisely our Goals 1 and 2 from Section III.

In summary, we can answer RQ1 as:

**The MMO is effective because we found that:**
- it provides considerably better results than the SO\textsubscript{best} (82%, 18 out of 22 cases) and PMO (90%, 20 out of 22 cases).
- it also obtains relatively good improvement over MMO-FSE\textsubscript{best}, thanks to the new normalization.

**B. RQ2: Resource Efficiency**

1) **Method:** To understand the resource efficiency of MMO in RQ2, for each case out of the 22, we use the following procedure:

1) Identify a baseline, \( b \), taken as the smallest number of measurements that the best single-objective counterpart

Fig. 14. The number of times to find the global optimum within 50 runs for all cases. The reported figures are the average and standard error across all cases. For all statistical comparisons with MMO, \( \hat{A}_{12} \) > 0.5 means MMO wins. All comparisons show large effect sizes.

### TABLE III

|                | \( \hat{A}_{12} \) (\( p \) value) | \( \hat{A}_{12} \) (\( p \) value) | \( \hat{A}_{12} \) (\( p \) value) | \( \hat{A}_{12} \) (\( p \) value) |
|----------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| SO\textsubscript{best} | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |
| MMO-FSE\textsubscript{best} | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |
| PMO | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |
| MMO | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |

|                | \( \hat{A}_{12} \) (\( p \) value) | \( \hat{A}_{12} \) (\( p \) value) | \( \hat{A}_{12} \) (\( p \) value) | \( \hat{A}_{12} \) (\( p \) value) |
|----------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| SO\textsubscript{best} | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |
| MMO-FSE\textsubscript{best} | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |
| PMO | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |
| MMO | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) | \( \text{Mean/SE} \) |

**Comparing MMO With the Other State-of-the-Arts Over 50 Runs.** SO\textsubscript{best} and MMO-FSE\textsubscript{best} denote the best single-objective Model/Optimizer and the MMO-FSE with the Best Weight, respectively. The Table shows the Average and Standard Error (SE) on the Target Performance Objective Achieved (All Objective Values Are Normalized Within \([0,1]\)) and the Closer to the Left, the Better. The best denotes the best average among others. Column \( \hat{A}_{12} \) (\( p \) value) shows the \( \hat{A}_{12} \) and corrected \( p \) value when comparing the corresponding counterpart with MMO. "T", "S", "M", and "L" denotes Trivial, Small, Medium, and Large Effect Size, respectively. The Blue Cells denote MMO wins (\( \hat{A}_{12} > 0.5 \)) while Red Cells Mean It Loses (\( \hat{A}_{12} < 0.5 \)); Otherwise It Is A Tie (\( \hat{A}_{12} = 0.5 \)). Statistically significant comparisons, i.e., \( \hat{A}_{12} \geq 0.56 \) (\( \hat{A}_{12} \leq 0.44 \)) and \( p < 0.05 \) are highlighted in bold.
Fig. 15. Projected landscapes of the explored configurations for two example systems. Each point is a configuration measured in the run, regardless of whether it is preserved or not. (a) represents a case where MMO finds the global optimum while the others do not; (b) showcases the scenario where none of them found the global optimum, but MMO produces results that are much closer than those of the others.

Fig. 16. Illustrating the calculations of speedup $s$.

measurements, hence is more efficient when achieving the best of what can be produced by the baseline.

- In Fig. 16(b), $s < 1$ suggests that the approach reaches the converging performance of $SO_{best}$ by using more measurements, hence is less efficient even though it can lead to better results when the full budget is exhausted.
- In Fig. 16(c), we use $s < 0$ to denote the case where the approach has never been able to reach the converging performance of $SO_{best}$, denoted as “failed”.

Of course, when $s = 1$, both approaches reach the same performance at exactly the same number of measurements.

Clearly, the greater the $s$, the better speedup, and hence more resources can be saved against that consumed by $SO_{best}$. In particular, if the MMO is resource-efficient, then we would expect at least $s = 1$ and ideally $s > 1$. Since in our context, the resource is the number of measurements, it reflects the time and computation required by a model. Again, we use the same $SO_{best}$ and MMO-FSE$_{best}$ from RQ1.

2) Findings: As can be seen from Fig. 17, despite a very small number of cases where the MMO uses more resources to reach the performance level achieved by the best single-objective counterpart, most commonly it uses less number of measurements than, or at least identical to, the baseline to find the same or better results, e.g., it obtains a speedup up to $2.09 \times$. In particular, the MMO achieves 17 cases of $s > 1$; 3 cases of $s = 1$; and 2 cases of $0 < s < 1$. Remarkably, there is no case where it fails to reach the performance level (the divided bars, denoted as $s < 0$). This indicates that the MMO overcomes local optima better and more efficiently—a key attraction to software configuration tuning due to its expensive measurements. The MMO-FSE$_{best}$ does show competitive results with respect to MMO: it has 13 cases of $s > 1$ and 4 cases of $s = 1$; but there are 5 cases of $s < 0$ due to the issues discussed in Section III-E, which could be undesirable on certain domains. Again, the above is due to MMO covering the key properties of software configuration tuning (Section III) while making it much less sensitive to the weight parameter.

In contrast, the PMO exhibits the worst resource efficiency in terms of the speedup over $SO_{best}$, as it has 3 cases of $s > 1$, together with 3 cases of $s = 1$; 1 case of $0 < s < 1$, respectively,
while the remaining 16 cases are $s < 0$. This is a clear sign that PMO is generally resource-hungry as discussed in Section III.

Interestingly, however, we see that PMO is considerably resource-efficient in 3 cases (KERAS-DNN/SA-O1, KERAS-DNN/SA-O1, and KERAS-DNN/ADAIC-O1). Despite being rare, this is indeed possible, because it can be severely affected by the relationship between the target and auxiliary performance objective. When there is a conflicting or weak relationship, then certainly optimizing the auxiliary performance objective would be harmful to the target one, as the valuable budget is wasted on something meaningless. However, when the relationship is harmonic, optimizing one can in fact beneficial to the other. In such a case, the drawback of PMO we discussed in Section III would become blurred. We observed that the inference time and AUC on the three cases tend to be harmonic objectives. Note that the relationship between the two objectives need not be symmetric. For example, improving inference time can also help to significantly improve AUC, but finding configurations with better AUC may slightly improve inference time. In other words, the extent of interaction can be different. This is why PMO tends to be efficient on KERAS-DNN/ADAIC-O1 but not on KERAS-DNN/ADAIC-O2.

As the conclusion for RQ2, we say:

The MMO is resource-efficient as we found that
- it saves generally more resources than the best single-objective counterpart to reach the same or better results (for 17 out of 22 cases with up to 2.09× speedup).
- it leads to very competitive resource-saving and fewer cases of “failed” when compared with MMO-FSE$_{\text{best}}$.
- the PMO, in contrast, is much more resource-hungry.

C. RQ3: Benefits Over MMO-FSE

1) Method: In RQ3, we seek to verify the benefits provided by the weight-free design in MMO are indeed meaningful over MMO-FSE. Particularly, on each of the 22 systems/environments, we examine how MMO performs against the MMO-FSE under different weights using the full-scale experiment. Indeed, if the differences between MMO and MMO-FSE over all weight settings are small, then perhaps using the weight in MMO-FSE in order to find a promising one, which discovers at least one of the promising weights as identified previously under the full-scale experiment.

Fig. 18 shows the mean on MMO and MMO-FSE under different weights; the promising ones have been highlighted. As can be seen, we observe that:
- In the majority of cases, MMO can outperform MMO-FSE over all weight settings.
- For all the cases, the performance of MMO-FSE deviates significantly under different weights.
- The promising weights often achieve considerably superior results than most of the others, i.e., up to 10× better.
- It is difficult to conclude a generally promising weight over the cases. Most commonly, the promising weight can be radically diverse, e.g., it is $w = 0.01$ for MARIADB-O2 while $w = 0.9$ for X264-O1—a 90× difference.

To understand how much extra resource is required to tune the weight in MMO-FSE in order to find a promising one, Fig. 19 illustrates the results. Clearly, we see that for 13 out of the 22 cases, it needs 50% or more of the full-scale search budget to identify a promising weight, which may not be acceptable when using the MMO-FSE under a case; or otherwise, the quality of tuning could be compromised. Even for the 9 cases where the extra resources required are between 10% and 30%, it may still be undesirable since some systems, such as WP9, can take up to 190 seconds to measure a single configuration. Therefore, for RQ3, we say:

Thanks to the new normalization, the weight-free feature is meaningful in MMO as we found that
- the MMO-FSE can be highly sensitive to the weight setting—there exist some rather diverse, case-dependent, and promising weights that perform significantly better than the others, which can only be discovered via pre-tuning.
- the effort required by the MMO-FSE to identify a promising weight is non-trivial, i.e., it takes 50% or more of the full-scale search budget for 13 out of 22 cases, which would otherwise be saved by using MMO instead without compromising the quality.

D. RQ4: Consolidating Model-Based Tuning

1) Method: For RQ4, we extended FLASH and BOCA with our MMO, denoted as FLASH$_{\text{MMO}}$ and BOCA$_{\text{MMO}}$, respectively. As shown in Algorithm 4 and 5, the change is highlighted in colors, from which we see that the amendment is merely a single line of code which changes the original search strategy...
Fig. 18. Comparing MMO and MMO-FSE with different weights on the normalized target performance objective (mean and standard error) over 50 runs; the smaller, the better.

that solely optimizes the target performance objective to searching over the space of MMO (working with NSGA-II). In this way, the search is conducted in the transformed meta-objective space of the surrogate-predicted objectives, in which the system $F$ is replaced by the surrogates $M$.

Similar to the previous sections, the statistical test$^{13}$ and effect size are reported for every pairwise comparison between the FLASHMMO and FLASH (BOCAMMO and BOCA) over 50 runs.

2) Findings: From Table IV, it is clear that FLASHMMO obtains better results than FLASH in general: it wins 15 out of 22 cases while loses 7 others. In particular, in those cases where FLASHMMO wins, 12 of them are statistically significant.

We use the Wilcoxon signed-rank test here since the comparisons are paired, i.e., on each run, all optimizers use the same set of randomly sampled training data for building the surrogate.

13Similar results have also been registered for comparing BOCA MMO and BOCA. This means that, even when searching within the surrogate-predicted space, our MMO can bring considerable improvement on model-based tuning methods like FLASH and BOCA.

To take a closer investigation, Fig. 20 shows the overall search trajectories for the 20 measurements that are actually spent on tuning. Clearly, FLASHMMO produces a trajectory with a steeper slope than that of FLASH over all cases and runs. The standard error of the average performance is also smaller, implying that MMO can also consolidate the stability of outcomes.
Fig. 19. The necessary resource consumed $p$ (in terms of % on the full-scale experiments’ search budget) for the MMO-FSE to find the promising weight(s) as that identified under the full-scale experiments; these are the resources that would have been saved by using MMO. As a reference, the dashed line highlights the 50% threshold of the budget. All cases are sorted in descending order.

Algorithm 4: **FLASHMMO**

| Input: Configuration space $V$; the system $F$ |
| Output: $s_{best}$ the best configuration on $f_1(x)$ |
| Declare: vector of surrogates $M$ (one for each performance objective) |
| 1 Randomly initialize a size of $k$ configurations $\mathcal{P}$ |
| 2 $\mathcal{V} \leftarrow \mathcal{V} - \mathcal{P}$ |
| 3 while The search budget is not exhausted do |
| 4 /* searching an estimated-best configuration in the transformed meta-objective space defined by the MMO (with NSGA-II), as shown in Algorithm 1 */ |
| 5 $M = \text{TRAINCARTS}(\mathcal{P})$ |
| 6 $o = \text{MMONNSGA-II}(\mathcal{V}, M)$ |
| 7 $\mathcal{V} \leftarrow \mathcal{V} - o$ |
| 8 $\mathcal{P} \leftarrow \mathcal{P} + o$ |
| 9 if $o$ is measured to be better than $s_{best}$ on $f_1(x)$ then |
| 10 $s_{best} = o$ |
| 11 return $s_{best}$ |

at as little as the 2nd measurement into the tuning (as the first 30 are for pre-training the surrogate); while the latter reflects that the MMO helps to improve resource efficiency, achieving a $\frac{50}{47} = 1.22 \times$ speedup over FLASH when reaching its best outcome under the search budget. When comparing BOCA$\text{MMO}$ with BOCA, the improvement on efficiency is less obvious since BOCA leverages the information of important options. However, we see that at the 46th measurement, BOCA$\text{MMO}$ starts to become superior to its counterpart while achieving the best of BOCA at the 47th measurement, enabling a $\frac{50}{47} = 1.06 \times$ speedup.

Although the use of a surrogate can transform the original configuration landscape into a different one according to the estimated value, the issue of local optima remains present [40], providing that the accuracy is sufficient. Therefore, the reason that improves BOCA can be attributed to the fact that MMO relieves the issue of local optima trap, as the search strategy in BOCA is restricted to a certain region of the search space with respect to the important options. As for FLASH, which is resilient to local optima due to the random search nature, the improvement is the result of MMO being able to preserve the tendency towards the best of target performance objectives, providing better guidance in the tuning.

Overall, for RQ4, we have:

The MMO can significantly consolidate model-based tuning methods because

- it improves the results on both FLASH and BOCA for 15 out of 22 cases (68%), within which 12 and 13 cases are statistically significant, respectively.
- it enables a $1.22 \times$ and $1.06 \times$ speedup over FLASH and BOCA, respectively, with gradually more stable outcomes across all the cases overall.

VI. USING MMO IN PRACTICE

Using MMO in practical scenarios for any new system is straightforward. In what follows, we describe the basic steps for the application of MMO in practice:

1) Build the benchmark under which the configurable software system needs to be tuned. This is often selected from some well-known ones (e.g., WordCount for Storm) or emulated depending on the software engineers’ understanding of the software’s running environment.

2) Identify the key configuration options and their possible values. In practice, one can either derive these from previous studies (as we have done in this work) or based on experience and domain understanding.
3) Confirm the target performance objective and an arbitrarily chosen auxiliary performance objective, as well as how they are measured. It is important to ensure that both objectives can be influenced by different configurations. For example, when runtime is an important target for a system, then the auxiliary performance objective may be chosen from CPU load, memory consumption, and energy usage, etc.

4) Define a tuning budget in terms of the number of measurements. The parameters of the underlying NSGA-II of MMO can be chosen from some widely used ones. At this point, one can also decide on whether to use MMO as a measurement-based optimizer or a model-based one (e.g., by pairing MMO with optimizers like FLASH or BOCA). In general, the model-based version is recommended if the budget is small.

5) Run MMO on the system.

### VII. Threats to Validity

Threats to internal validity can be related to the search budget. To tackle this, we have set a budget that reaches a reasonable convergence for all the optimizers compared—a typical setup when the study aims to examine the effectiveness of mitigating local optima [74]. The other parameter settings follow what has been pragmatically used from the literature [10], [26], [31], [54], [76], [80]. or tuned through preliminary runs. However, we acknowledge that examining alternative parameters can be an interesting topic and we leave this for future work. To mitigate bias, we repeated 50 experiment runs under each case.

The metrics and evaluation used may pose threats to construct validity. Since there is only a single performance concern, there is no need to consider metrics with respect to multi-objective optimization [63]. We conduct the comparison
based on the target performance objective and to the resource (number of measurements) required to converge to the same result. Both of these are common metrics in software configuration tuning [36], [70]. To verify statistical significance and effect size, we use the Wilcoxon test (including both the paired and non-paired versions depending on the research questions) and \( A_{12} \) to examine the results. While the MMO model is optimizer-agnostic, we examine mainly on NSGA-II in this work; using alternative multi-objective optimizers is unlikely to invalidate our conclusion but we admit the usefulness of evaluating over a wide range of optimizers with MMO, which can be part of the future work.

Threats to external validity can be raised from the subjects studied. We mitigated this by using 11 systems/environments that are of different domains, scales, and performance attributes, as used by prior work [51], [52], [67], [70], [75]. We also compared the proposed MMO (under the new normalization) with four state-of-the-art single-objective counterparts for software configuration tuning, PMO model, and the MMO-FSE. Further, we examine how it can help to consolidate FLASH, a recent model-based tuning method from the software engineering community. Nonetheless, we agree that studying additional systems and optimizers may prove fruitful.

VIII. RELATED WORK

Broadly, optimizers for software configuration tuning can be either measurement-based or model-based.

A. Measurement-Based Tuning

In measurement-based tuning methods, the optimizer is guided by directly measuring the configuration of the software systems. Despite the expensiveness, the measurements can accurately reflect the good or bad of a configuration (and the extents thereof). A wide range of optimizers have been studied, such as random search [8], [71], [78], [99], hill climbing [34], [62], [95], single-objective genetic algorithm [7], [80], [84] and simulated annealing [37], [42], to name a few.

Under such a single-objective model, a key difference for those optimizers lies in the tricks that attempt to overcome the issues of local optima. For example, some extend the random search to consider a wider neighboring radius of the configuration structure, hence it is more likely to jump out from the local optima [71]. Others rely on restarting from a different point, such as in restarted hill climbing, hence increasing the chance to find the “right” path from local optima to the global optimum [95], [102]. More recently, Krishna et al. [57] has relied on probabilistically accepting worse configurations to jump out of local optima—a typical feature of the simulated annealing [34], [42].

Our MMO differs from all the above as it lies in a higher level of abstraction—the optimization model—as opposed to the level of optimization method. In particular, with the new normalization, the purposely-crafted Pareto relation in MMO has been shown to be able to better overcome the local optima for software configuration tuning.

B. Model-Based Tuning

Instead of solely using the measurements of software systems, the model-based tuning methods apply a surrogate (analytical [30], [31], [59] or machine learning based [39], [41], [51], [70]) to evaluate configurations, which guides the search in an optimizer. The intention is to speed up the exploration of configurations as the model evaluation is much cheaper. Yet, it has been shown that the model accuracy and the availability of initial data can become an issue [102].

Studies on model-based tuning for software systems differ mainly on the way of building surrogates and the choice of acquisition function. Among others, Jamshidi and Casale [51] use Bayesian optimization to tune software configuration, wherein the search is guided by the Gaussian Process based surrogate trained from the data collected. Nair et al. [70] follow a similar idea to propose FLASH, but the CART is used instead as the surrogate. More recently, Chen et al. [17] also follow BO and CART to propose BOCA, but they additionally identify the “important configuration options” from the Random Forest model. Such information would then inform the optimization of the acquisition function in determining what to sample next.

Since MMO lies in the level of optimization model, it is complementary to the model-based methods in which the MMO would take the surrogate values as inputs instead of the real measurements. This, as we have shown using FLASH as a case, can better consolidate the tuning results.

C. General Parameter Tuning

Optimizers proposed for the parameter tuning of general algorithms can also be relevant [9], [12], [47], [77], including IRace [64], ParamILS [49], SMAC [48], GGA++ [5], as well as their multi-objective variants, such as MO-ParamILS [11] and SPRINT-Race [101]. To examine a few examples, ParamILS [49] relies on iterative local search—a search procedure that may jump out of local optima using strategies similar to that of SA and SHC-r. Further, a key contribution is the capping strategy, which helps to reduce the need to measure an algorithm under some problem instances, hence saving computational resources. This is one of the goals that we seek to achieve too. Similar to Nair et al. [70], SMAC [48] uses Bayesian optimization but relies on a Random Forest model, which additionally considers the performance of an algorithm over a set of instances.

However, their work differs from ours in two aspects. Firstly, general algorithm configuration requires working on a set of problem instances, each coming with different features. The software configuration tuning, in contrast, is often concerned with tuning software systems under a given benchmark (i.e., one instance) [26], [51], [70], [102]. Therefore, most of their designs for saving resources (such as the capping in ParamILS) were proposed to reduce the number of instances measured. Of course, it is possible to generalize the problem to consider multiple benchmarks at the same time, yet this is outside the scope of this paper. Secondly, none of them works on the level of optimization model, and therefore, similar to the case of...
FLASH and BOCA, our MMO is still complementary to their optimizers.

D. Multi-Objectivization in SBSE

Multi-objectivization, which is the notion behind our MMO model, has been applied in other SBSE problems [33], [68], [86], [100]. For example, to reproduce a crash based on the crash report, one can purposely design a new auxiliary objective, which measures how widely a test case covers the code, to be optimized alongside with the target crash distance [33]. A multi-objective optimizer, e.g., NSGA-II, is directly used thereafter. A similar case can be found also for the code refactoring problem [68]. However, during the tuning process, such a model, i.e., PMO in this paper, can result in poor resource efficiency as it wastes a significant amount of resources in optimizing the auxiliary objective, which is of no interest. This is a particularly unwelcome issue for software configuration tuning where the measurement is expensive. As we have shown in Section V, PMO performs even worse than the classic single-objective model in most of the cases.

IX. CONCLUSION AND FUTURE WORK

To mitigate the local optimia issue in software configuration tuning, this paper takes a different perspective—multi-objectivizing the single objective optimization scenario. We do this by proposing a meta multi-objective model (MMO), at the level of optimization model (external part), as opposed to existing work that focuses on developing an effective single-objective optimizer (internal part). This work provides a sound analysis to interpret the principle behind MMO and explain what causes its limitation, namely eliminating the need for the weight parameter in the MMO model. Deriving on the theoretical understanding of the root cause, we then overcome such a limitation by proposing a simple yet effective new normalization.

We compare MMO under the new normalization with four state-of-the-art single-objective optimizers, the plain multi-objectivization model, and the MMO model with the old normalization from our prior FSE work over 22 cases that are of diverse performance attributes, systems, and environments. The results reveal that the MMO model:

• can generally be more effective in overcoming local optima with better results;
• and does so by utilizing resources more efficiently (better speedup) in most cases;
• saves a considerable amount of extra resources that would otherwise be required for identifying the best weight.

Furthermore, we use MMO as part of FLASH and BOCA, two recent model-based efforts from the software engineering community for configuration tuning, and revealing that it can:

• considerably consolidate the results;
• while enabling good speedup overall.

Future directions of this work are exciting and fruitful, as it paves a new way of thinking about the resolution for mitigating local optima in software configuration and perhaps in a wider context of SBSE—multi-objectivizing at the level of optimization model instead of working at the level of an optimizer/algorithm. Specifically, the most immediate next steps include extending MMO beyond two meta-objectives (e.g., through considering multiple auxiliary objectives if any) and exploring the possibility of designing a tailored multi-objectivization model for other SBSE problems. At the same time, through the analysis discussed in this work, we hope that there will be subsequent studies that take the unique characteristics of MMO into account, which can further advance software configuration tuning and beyond.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for their highly constructive and insightful comments. Tao Chen would like to thank Yvonne and Savion for their assistance.

REFERENCES

[1] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, “Automated repair of feature interaction failures in automated driving systems,” in Proc. 29th ACM SIGSOFT Int. Symp. Softw. Testing Anal. (ISTE’20), Virtual Event, USA, New York, NY, USA: ACM, 2020, pp. 88–100, doi: 10.1145/3395363.3397386.
[2] H. Abdi, “Holm’s sequential Bonferroni procedure,” Encyclopedia Res. Des., vol. 1, no. 8, pp. 1–8, 2010.
[3] M. A. Ahmad, C. Eckert, and A. Teredosi, “Interpretable machine learning in healthcare,” in Proc. ACM Int. Conf. Bioinfo., Comput. Biol., Health Inform. (BCB), Washington, DC, USA, New York, NY, USA: ACM, 2018, pp. 559–560, doi: 10.1145/3233547.3233667.
[4] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl, “A scalable approach for QoS-based web service selection,” in Proc. Int. Conf. Service-Oriented Comput. (ICSOC), Sydney, Australia, in Revised Selected Papers, Lecture Notes in Computer Science, vol. 5472. New York, NY, USA: Springer, 2008, pp. 190–199, doi: 10.1007/978-3-642-01247-1_20.
[5] C. Ansötegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tinerney, “Model-based genetic algorithms for algorithm configuration,” in Proc. 24th Int. Joint Conf. Artif. Intell. (IJCAI), Buenos Aires, Argentina. AAAI Press, 2015, pp. 733–739. [Online]. Available: http://ijcai.org/Abstract/15/109
[6] A. Arcuri and L. C. Briand, “A practical guide for using statistical tests to assess randomized algorithms in software engineering,” in Proc. 33rd Int. Conf. Softw. Eng. (ICSE’11), New York, NY, USA: ACM, 2011, pp. 1–10.
[7] B. Behzad et al., “Taming parallel I/O complexity with auto-tuning,” in Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC’13), Denver, CO, USA, New York, NY, USA: ACM, 2013, pp. 68:1–68:12, doi: 10.1145/2503210.2503278.
[8] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” J. Mach. Learn. Res., vol. 13, no. 2, pp. 281–305, 2012.
[9] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle, “Automatic configuration of multi-objective optimizers and multi-objective configuration,” in High-Performance Simulation-Based Optimization, (Studies in Computational Intelligence), vol. 833. New York, NY, USA: Springer-Verlag, 2020, pp. 69–92, doi: 10.1007/978-3-030-18764-4_4.
[10] Z. Bingul, “Adaptive genetic algorithms applied to dynamic multiobjective problems,” Appl. Soft Comput., vol. 7, no. 3, pp. 791–799, 2007, doi: 10.1016/j.asoc.2006.03.001.
[11] A. Blot, H. H. Hoos, L. Jourdan, M. Kessaci-Marmion, and H. Trautmann, “MO-ParamILS: A multi-objective automatic algorithm configuration framework,” in Proc. 10th Int. Conf. Learn. Intell. Optim. (LION), Ischia, Italy, in Revised Selected Papers, Lecture Notes in Computer Science, vol. 10079. New York, NY, USA: Springer-Verlag, 2016, pp. 32–47, doi: 10.1007/978-3-319-50349-3_3.
[12] A. Blot, M. Marmion, L. Jourdan, and H. H. Hoos, “Automatic configuration of multi-objective local search algorithms for permutation problems,” Evol. Comput., vol. 27, no. 1, pp. 147–171, 2019, doi: 10.1162/evco_a_00240.
[13] L. Breiman, “Random forests,” Mach. Learn., vol. 45, pp. 5–32, Oct. 2001.
