Revealing Domain-Spatiality Patterns for Configuration Tuning: Domain Knowledge Meets Fitness Landscapes

YULONG YE, IDEAS Lab, University of Birmingham, United Kingdom
HONGYUAN LIANG*, University of Electronic Science and Technology of China, China
CHAO JIANG, University of Birmingham, United Kingdom
MIQING LI, University of Birmingham, United Kingdom
TAO CHEN†, IDEAS Lab, University of Birmingham, United Kingdom

Configuration tuning for better performance is crucial in quality assurance. Yet, there has long been a mystery on tuners' effectiveness, due to the black-box nature of configurable systems. Prior efforts predominantly adopt static domain analysis (e.g., static taint analysis), which often lacks generalizability, or dynamic data analysis (e.g., benchmarking performance analysis), limiting explainability. In this work, we embrace Fitness Landscape Analysis (FLA) as a bridge between domain knowledge and difficulty of the tuning. We propose `Domland`, a two-pronged methodology that synergizes the spatial information obtained from FLA and domain-driven analysis to systematically capture the hidden characteristics of configuration tuning cases, explaining how and why a tuner might succeed or fail. This helps to better interpret and contextualize the behavior of tuners and inform tuner design. To evaluate `Domland`, we conduct a case study of nine software systems and 93 workloads, from which we reveal several key findings: (1) configuration landscapes are inherently system-specific, with no single domain factor (e.g., system area, programming language, or resource intensity) consistently shaping their structure; (2) the core options (e.g., `pic-struct` of x264), which control the main functional flows, exert a stronger influence on landscape ruggedness (i.e. the difficulty of tuning) compared to resource options (e.g., `cpu-independent` of x264); (3) Workload effects on landscape structure are not uniformly tied to type or scale. Both contribute to landscape variations, but their impact is system-dependent.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; *Software performance*.

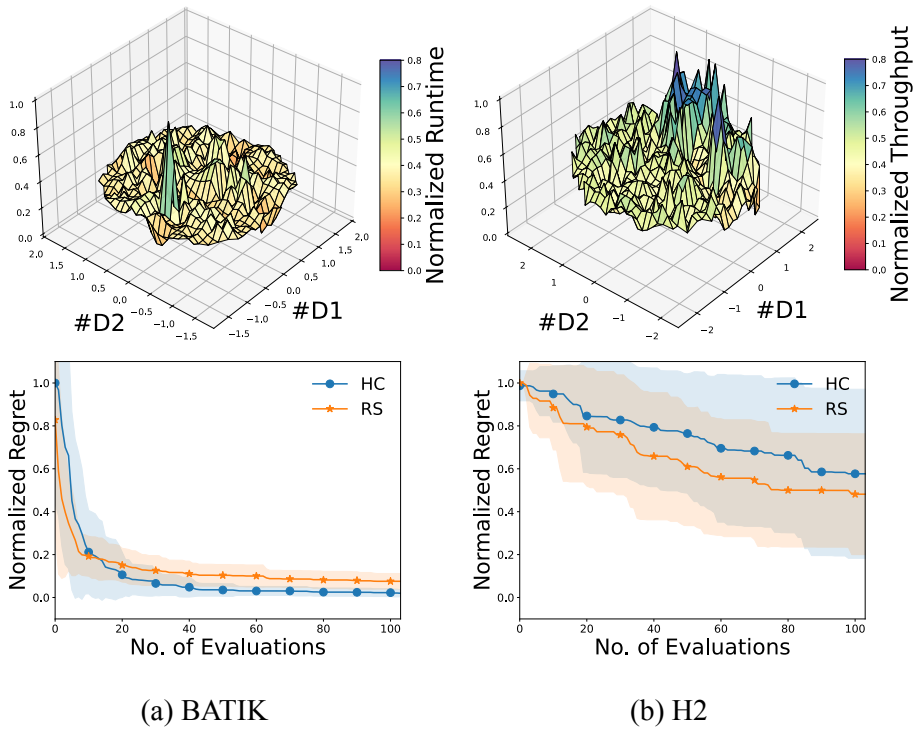
Additional Key Words and Phrases: Configuration tuning, fitness landscape analysis, domain analysis, search-based software engineering.

ACM Reference Format:
Yulong Ye, Hongyuan Liang, Chao Jiang, Miqing Li, and Tao Chen. 2026. Revealing Domain-Spatiality Patterns for Configuration Tuning: Domain Knowledge Meets Fitness Landscapes. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (March 2026), 43 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Hongyuan Liang is also supervised in the IDEAS Lab.
†Corresponding author: Tao Chen, t.chen@bham.ac.uk.

Authors' Contact Information: Yulong Ye, yxy382@student.bham.ac.uk, IDEAS Lab, University of Birmingham, United Kingdom; Hongyuan Liang, lianghy16@outlook.com, University of Electronic Science and Technology of China, China; Chao Jiang, cxj249@student.bham.ac.uk, University of Birmingham, United Kingdom; Miqing Li, University of Birmingham, United Kingdom, m.li.8@bham.ac.uk; Tao Chen, t.chen@bham.ac.uk, IDEAS Lab, University of Birmingham, United Kingdom.

ACM 1557-7392/2026/3-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) BATIK (b) H2

Fig. 1. The regret (i.e., gap to the optimal performance) using Random Search (RS) and Hill Climbing (HC) to tune Batik and H2. The top shows their landscapes processed by MDS [34].

1 Introduction

Software systems are highly configurable, offering numerous tunable configuration options. The goal thereof is straightforward: by allowing flexible configuration, software systems can achieve greater applicability across a wide range of domains and cater to varying performance requirements, e.g., runtime and throughput [25, 26, 87]. Yet, excessive configurability comes with its own costs: it has been shown that globally 59% of the software performance issues—where performance requirements were severely violated—are attributed to poorly chosen configurations rather than code [43].

To mitigate these challenges, various automated configuration tuners, relying on algorithms such as Bayesian optimization (BO) [19, 39, 41, 100] and genetic algorithms (GAs) [16, 74, 97], have been developed to search for optimal configurations. However, it has been reported that the effectiveness of these tuners varies significantly across different contexts [99]. For instance, Figure 1 depicts the tuning trajectories of two tuners, random search (RS) [73] and hill climbing (HC) [60, 95], on Batik and H2—two systems that exhibit distinct configuration landscapes. Clearly, the convergence and tuning quality differ, with an advantage of HC over RS on Batik, as sharply opposed to on H2. The effectiveness of tuning algorithms heavily depends on the underlying characteristics of the system/workload, e.g., the landscape of H2 is more rugged than that of Batik (see the top panel of Figure 1), causing the ineffectiveness of HC but RS, as an random sampling, is immune to the ruggedness level. However, to the community, it remains unclear how the configuration landscape and domain-specific factors (e.g., system areas, structures, and workload characteristics)

influence tuners' behaviors. Practitioners of configuration tuning need a systematic approach to understand the behaviors of tuners while linking/explaining such within the context of domain knowledge/characteristics of the system, which is a challenging desire.

Indeed, static domain analysis of configurable systems exist [30, 45, 58, 81], mainly leveraging *domain-specific knowledge* or *code-level analysis* without the need of running the system. The former refers to manually extracted knowledge from system documentation and expert-derived rules of thumb [45, 58, 81]; the latter uses code tracking, trace analysis, and software module reviews to understand how configuration options influence system behaviors [30]. However, existing domain analysis are mostly static, heavily relying on expert assumptions, which may fail to provide accurate, in-depth insights with respect to the difficulty of tuning a system.

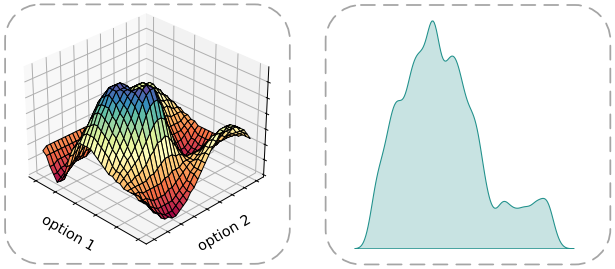
To overcome the shortcomings of static domain analysis, an alternative way to study configurable systems is dynamic data analysis, which primarily relies on observing the runtime states of the system, understanding the data distributions, and their correlations, for the values of configuration options and performance [51, 70]. However, this method only statistically considers the value of options and performance as isolated points, ignoring their spatial information in the configuration landscape, which is what a tuner's behaviors would be sensitive to [47]. Recently, fitness landscape analysis (FLA) [66, 94], which explicitly incorporates spatial information, has been explored as a means to uncover key characteristics of configurable systems [47]. Specifically, it provides an analytical lens to understand the structural characteristics of configuration tuning cases, explaining how and why a tuner might succeed or fail. Nevertheless, interpreting the system solely from a FLA perspective might not provide meaningful guidelines to the software practitioners, since the linkage between the outcomes of FLA and domain-specific factors is still missing.

In this paper, we present `Domland`, a holistic methodology for characterizing configuration tuning problems from both spatial and domain perspectives, and thus delivering insights for interpreting tuning behaviors/difficulty. Unlike the others, `Domland` provides guidelines that synergize the spatial information of the configuration landscape, mined using FLA, with domain knowledge at the option, system, and workload levels. This provides comprehensive, intuitive, and actionable insights to practitioners of configuration tuning. In summary, our main contributions include:

- We provide a new dynamic data analysis guideline to spatially analyze the configuration landscape using the selected/designed categories of analysis from FLA, as well as the metrics, that are the most relevant to configurable systems.
- We document a procedure of system-specific domain analysis at three levels of abstraction, together with general categorization of the information.
- Drawing both the configuration spatiality and domain analysis above, we outline a method for domain-spatiality synergy, linking the information from two different sources of analysis, providing actionable insights to tuners and system designers.

To showcase how `Domland` works and evaluates its usefulness, we conduct a case study by applying it to analyze nine widely-studied systems with a total of 93 workloads. From this, and the unique domain-spatiality synergy provided by `Domland`, we find several previously unknown/unconfirmed observations, such as:

- the configuration landscapes across the systems are inherently system-specific, with no single domain factor (e.g., system area, programming language, or resource intensity) consistently shaping their structure;
- despite system-specific landscape variations, core options commonly exert a stronger influence on landscape ruggedness than resource options. This suggests that certain option properties universally impact optimization complexity across different systems;



(a) Configuration Landscape (b) Performance Distribution

Fig. 2. Illustration of configuration landscape and performance distribution.

- the workload-induced landscape shifts are system-dependent—some remain stable, while others exhibit significant changes. Workload type and scale exhibit no uniform influence, implying complex interactions with system attributes.

All code and data can be accessed at our repository: https://github.com/ideas-labo/domland.

The remainder of this paper is as follows: Section 2 introduces preliminaries. Section 3 presents our methodology. Section 4 describes the empirical setup, followed by Section 5 which reports the results and analysis. Section 6 discusses the implications of our findings. Threats to validity, related work, and conclusion are presented in Sections 7, 8, and 9, respectively.

2 Preliminaries

2.1 Software Configuration Tuning

A configurable software system often comprises a set of configuration options, each taking categorical or numerical values. The objective of software configuration is to identify a configuration that optimizes the specific performance attribute, e.g., minimizing runtime or maximizing throughput, of the target system. Mathematically, this can be formulated as:

$$\arg\min f(\boldsymbol{c}) \text{ or } \arg\max f(\boldsymbol{c}), \tag{1}$$

where $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$ is a configuration with the values of n options in search space C. f represents the performance attribute of the target system. Since the internal behavior of the system is often opaque and lacks of explicit analytical form of f, this is considered a black box optimization problem [93], which has long been a fundamental challenge for software engineering [44].

2.2 Fitness Landscape Analysis

A way to know the hardness of an optimization problem is to use fitness landscape analysis (FLA). By characterizing the configuration space, FLA can help understand the complexity of the problem through measuring key features of the configuration of a system. For example, by analyzing the configuration landscape in Figure 2, FLA helps to answer several key questions regarding the configuration-performance landscape, each of which provides actionable insights for configuration tuning:

- **Where is the optimum located?** Understanding the position and accessibility of the optimum helps assess how easily a tuner can converge to the best-performing configuration(s). If the landscape exhibits strong global guidance (e.g., better configurations tend to lie closer

to the optimum), exploitation-driven tuners can efficiently reach the optimum; otherwise, exploration-driven strategies may be required.
- **What does local structure look like?** Analyzing the local structure reveals the distribution and quality of local optima in the landscape, including how many exist and how good they are. This informs whether a tuner may benefit from escaping suboptimal regions through global exploration, or whether convergence to a high-quality local region is sufficient to achieve satisfactory performance.
- **How rugged is the search space?** Ruggedness reflects how smoothly performance changes with small configuration perturbations. Smooth landscapes favor model-based tuners that rely on generalization, while rugged landscapes may demand exploration-driven strategies.

By answering these questions, FLA provides not only a descriptive understanding of configuration–performance relationships but also actionable guidance for tuner design. For instance, even partial landscape knowledge derived from limited sampling or historical configuration–performance data can be used to approximate the structural properties of the configuration space. Such analysis helps practitioners anticipate the relative difficulty of optimization, explain/predict algorithmic behavior, and guide appropriate tuning algorithm selection/design: for example, model-based methods such as Bayesian optimization are suitable for smoother landscapes, whereas evolutionary algorithms are more effective for highly rugged ones.

Formally, a fitness landscape can be represented as a triplet $(C, \mathcal{N}, f)$, where C defines the search space; $\mathcal{N}$ denotes a neighborhood structure that specifies the neighbors of each configuration; and $f: C \rightarrow \mathbb{R}$ represents the fitness function [76]. As from Figure 2, comparing with existing statistical distribution analysis [51, 70], FLA differs in the following aspects:

- **Spatial awareness:** FLA can capture the spatial structure of search and performance spaces while statistical distribution analysis focuses solely on performance/option variation.
- **Topographical insights:** FLA can additionally quantify critical spatial features in the landscape, e.g., local optima and ruggedness, to reveal landscape structure.
- **Hardness quantification:** The landscape metrics can be used to assess the difficulty of a problem with respect to an optimization algorithm.

2.2.1 Distance Measures and Neighborhood. A key aspect of the fitness landscape is the concept of distance between two configurations $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ (i.e., $d(\boldsymbol{c}_i, \boldsymbol{c}_j)$), where $\boldsymbol{c}_i$ and $\boldsymbol{c}_j \in C$. The most common distance metrics include Hamming distance, Manhattan distance, and Euclidean distance. Based on the distance measure(s), the neighborhood $\mathcal{N}(\boldsymbol{c})$ of a configuration $\boldsymbol{c}$ is commonly defined as the set of configurations differing from $\boldsymbol{c}$ within a small distance ϵ, i.e., $\mathcal{N}(\boldsymbol{c}) = \{\boldsymbol{c}' \mid d(\boldsymbol{c}', \boldsymbol{c}) \leq \epsilon\}$ [33].

2.2.2 Local Optima. Local optima capture the underlying local structural properties of a problem's landscape. Formally, a configuration $\boldsymbol{c}^\ell$ is considered as a local optimum if $f(\boldsymbol{c}^\ell)$ is better than $f(\boldsymbol{c})$, $\forall \boldsymbol{c} \in \mathcal{N}(\boldsymbol{c}^\ell)$, where $\mathcal{N}(\boldsymbol{c}^\ell)$ is the neighborhood of $\boldsymbol{c}^\ell$.

3 The Domland Methodology

As illustrated in Figure 3, Domland consists of three key phases: **Configuration Spatiality Analysis** (dynamic data analysis), **Configuration Domain Analysis** (static domain analysis), and **Domain-Spatiality Synergy**. The former two can run simultaneously, extracting the spatial information of the configuration landscape and domain-specific knowledge, respectively. Deriving from those, the last phase synergizes them to provide intuitive and comprehensive explanations to the configuration tuning for the relevant system(s).

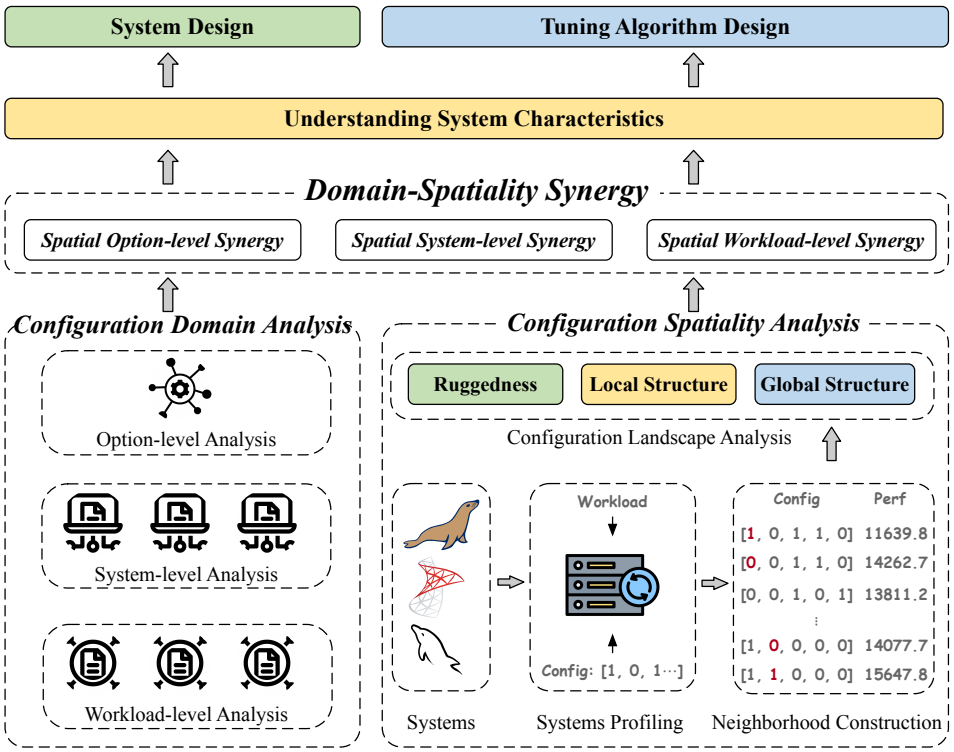


Fig. 3. Overview of the Domland methodology.

The information analyzed in **Configuration Spatiality Analysis** is almost applicable to any scenarios and systems. In contrast, depending on the goal, the levels of abstraction in the **Configuration Domain Analysis** can be tailored to fit the requirements:

- When there is needed to comparatively compare different systems and workloads, then this phase can be applied to the option, system, and workload levels.
- When only one system is concerned, one can emphasize on the option and workload levels.
- When there is only one system and one merely needs to understand the tuning under a specific workload, then Domland can be adopted to the option level only.

In what follows, we delineate every phase in Domland.

3.1 Configuration Spatiality Analysis

In data analysis, we leverage FLA to explore and quantify the topographical characteristics of configurable software systems. By treating the configuration-performance as a structured landscape, Domland provides spatial insights into the relationships between configurations and their associated performances.

3.1.1 Systems Profiling. An essential part of any dynamic data analysis is data collection, and for our problem, this means profiling the configurable system(s). Formally, given an arbitrary software system S, the corresponding configuration space C, and performance objective f, we need to obtain a representative set of data points $\{(\boldsymbol{c}, f(\boldsymbol{c}))\}$ from C.

To that end, several approaches can be employed. For example, for certain scenarios, it is possible to use grid search that comprehensively covers the configuration space. However, since measuring

configurable systems is often profoundly expensive [27, 50], it makes sense to apply sampling-based methods [54, 78] to select a subset of representative samples. In Domland, we support both categories of the sampling and it is up to the software engineers to choose depending on the cost and coverage trade-off [46].

3.1.2 Neighborhood Construction. After collecting the configuration-performance data, constructing the configuration landscape requires defining a neighborhood structure $\mathcal{N}$, which determines how the configurations are connected based on a distance measure $d(\boldsymbol{c}_i, \boldsymbol{c}_j)$. The choice of this measure is crucial, as it directly influences the topographical properties of the landscape that will be measured. The most common distance metrics include Hamming distance, Manhattan distance, and Euclidean distance.

3.1.3 Configuration Landscape Analysis. Leveraging the definition of neighborhood, we can now analyze the spatial information of a configuration landscape in the following three aspects[1]:

Global structure captures a holistic view of configuration-performance spatial information, assessing how easily or hardly a tuner can navigate toward the global optimum. Domland quantifies this aspect using two metrics:

- **Fitness distance correlation (FDC, ϱ)** [53] measures the correlation between performance and proximity to the global optimum, capturing the overall searchability of the landscape. As suggested in [14, 32, 53], a correlation $\varrho \geq 0.15$ indicates a globally guided search, whereas $-0.15 < \varrho < 0.15$ and $\varrho \leq -0.15$ reflects irregular and deceptive structures, respectively.
- **Basin of attraction** [66] refers to the set of points in the search space that eventually converge to a global/local optimum under a local search process. A larger attraction basin indicates that more configurations converge to the optimum, making it easier to reach. Here, we regard the basin size larger than 21% of the whole search space as an "easy" problem, given the fact that randomly sampling 20 configurations (same as many population-based search in software configuration tuning [20, 98]) for local search will lead to a probability of 99% of locating at least one optimum in the search space (i.e., $1 - (1 - 0.99)^{1/20} \approx 0.21$).

Fitness distance correlation captures the directional guidance of the landscape, i.e., whether better configurations tend to be closer to the optimum, reflecting global searchability or deceptiveness. Basin of attraction reflects the likelihood that the global optimum can be reached from random initial configurations, quantifying the global optimum's attractiveness. Together, they provide an interpretable and practical assessment of the tuning difficulty from a global perspective [67].

Local structure examines the spatial distribution of suboptimal configurations in the landscape, assessing *how frequently local traps appear; their quality against the global optimum; how strong their attractiveness is*. In Domland, we capture these properties through the following:

- **The number of local optima** quantifies the modality of a landscape, indicating how many distinct suboptimal configurations exist.
- **The quality of local optima** measures the performance of suboptimal configurations relative to the global optimum, indicating how competitive these configurations are within the landscape. Here, we classify the performance of local optima into three tiers based on the relative quality metric ℓ_q: high-quality ($\ell_q \geq 0.67$); medium quality ($0.33 < \ell_q < 0.67$); and low quality ($\ell_q \leq 0.33$).
- **Basins of attraction** of the local optima reflect the strength of their pull and how likely the tuning is to be trapped.

[1]The detailed specification of the chosen landscape metrics will be introduced in Section 4.2.2.

The number of local optima reflects how fragmented the landscape is, indicating how likely a tuner is to be caught in multiple competing regions [47]. The quality of local optima indicates whether these regions are sufficiently competitive relative to the global optimum, which determines how acceptable a locally converged configuration may be. The basins of attraction quantify the attractiveness of local optima, indicating how likely a tuner is to reach a particular region [66]. By jointly analyzing these dimensions, we can assess whether the landscape favors quick convergence to good enough configurations or demands sophisticated strategies to escape suboptimal traps. For example, when local optima exhibit competitive performance and hold a large basin size, a tuner might easily locate satisfactory configurations. Conversely, an abundance of low-quality local optima with a large basin size can mislead the search and increase the risk of convergence to suboptimal regions, requiring stronger exploratory capabilities.

Ruggedness captures the degree of fluctuation in performance with respect to the neighboring configurations, indicating whether the landscape exhibits smooth trends or abrupt, irregular changes. This reflects the difficulty of navigating the landscape. `Domland` measures this property using **autocorrelation** (r) [92], which quantifies performance similarity across neighboring configurations. A high autocorrelation ($r \geq 0.5$) suggests a smoother landscape; Moderate autocorrelation ($0.2 < r < 0.5$) indicates medium ruggedness; And low autocorrelation ($r \leq 0.2$) suggests a highly rugged landscape with sharp performance fluctuations [47, 80, 92].

The landscape metrics used in `Domland` and their implications for configuration tuning are summarised in Table 1. The detailed specification of these metrics will be introduced in Section 4.2.2. Notably, the metrics used in our analysis are chosen based on their well-established roles in the FLA literature [67, 67, 102] and their demonstrated applicability in the software engineering community [21, 24, 38, 47]. While we adopt a specific set of metrics in this study, our methodology is not limited to these choices. It remains extensible and can incorporate alternative metrics depending on the characteristics of the target systems (e.g., search-space type, dimensionality, sparsity) or the specific analysis goals (e.g., neutrality detection, evolvability assessment). We will further discuss this in Section 7.

In summary, the above configuration spatiality analysis lays the groundwork for extracting spatial insights into system behaviors, performance trends, and landscape navigability, which, in turn, serve as a foundation for guiding higher-level decisions (e.g., tuner selection/design).

3.2 Configuration Domain Analysis

Alongside configuration spatiality analysis, the other important phase in `Domland` is the static domain analysis, mining characteristics specific to the system(s) being studied. In `Domland` we provide generic "domain features" (or categories) extracted through our experiences and empirical observations. Those features, together with detailed guidelines, provide structural domain information to be synergized with the spatial information obtained previously.

To that end, three aspects are important as below.

3.2.1 System Level. For analyzing at the system level, we consider the following categories of features that are important for domain analysis and for cross-system comparisons:

- **Area:** A nominal feature that covers different application areas of the system, e.g., H2 is for database, X264 is for video encoding, Kanzi is for compression.
- **Language:** A nominal feature describes the main languages that built a system, such as Java and C++.
- **Resource intensity:** According to the nominal feature of resource intensity, most systems can be classified into four features: CPU, I/O, memory, and storage. Note that a system might belong to more than one type.

Table 1. Summary of landscape metrics and their implications for configuration tuning.

Landscape	Metric	Interpretation	Implications for tuner design/selection
Global structure	FDC (ϱ) [53]	Guided: $\varrho \geq 0.15$ Irregular: $-0.15 < \varrho < 0.15$ Deceptive: $\varrho \leq -0.15$	A high correlation indicates that better configurations lie closer to the optimum, revealing predictable structural characteristics that favour exploitation-driven strategies (e.g., hill climbing [60, 95]). In contrast, weak or negative correlations illustrate that the landscape provides little or misleading structural signal, requiring exploration-driven methods (e.g., SWAY [18], GA [16, 74]).
	Basin of Attraction [66]	> 0.21: easy to reach optimum < 0.01: hard to reach optimum	A large global basin implies that even simple local search can easily reach the optimum. A small basin, by contrast, calls for exploration-enhancing mechanisms such as restarts or diversity injection to avoid premature convergence (e.g., ParamILS [49]).
Local structure	Number (ℓ_p)	Large: high multimodality Small: low multimodality	A large number of local optima increases the risk of premature convergence, favoring diversity-preserving methods (e.g., multi-objectivization [20, 27]). Fewer optima allow effective use of deterministic local search (e.g., hill climbing [60, 95]).
	Quality (ℓ_q)	High: $\ell_q \geq 0.67$ Medium: $0.33 < \ell_q < 0.67$ Low: $\ell_q \leq 0.33$	High-quality local optima suggest that suboptimal configurations may still be acceptable, enabling smarter budget allocation and reducing unnecessary exploration. In contrast, low-quality local optima require escape strategies (e.g., multi-objectivization [20, 27]) to avoid stagnation in poor regions.
	Basin of Attraction [66]	> 0.21: strong attractiveness < 0.01: weak attractiveness	The larger the basins of local optima, the more likely the search will be attracted and trapped, motivating the use of restart or perturbation mechanisms (e.g., ParamILS [49]).
Ruggedness	Autocorrelation (r) [92]	Smooth: $r \geq 0.5$ Moderate: $0.2 < r < 0.5$ Rugged: $r \leq 0.2$	Smooth landscapes imply gradual performance transitions between neighbors, enabling model-based methods to generalize effectively (e.g., FLASH [71], SMAC [62], TPE [17]). Highly rugged landscapes exhibit abrupt performance fluctuations between neighboring configurations, favoring exploration-driven strategies (e.g., GA [16, 74]).

- **Complexity:** This numeric feature reflects the system's complexity and configurability, e.g., features such as lines of code (LOC), number of options, and configuration space.

The process of representing a given system using the above features is straightforward, as this is the common domain knowledge that most software engineers would have. For example, it is easy to know that the system H2 is a database built in Java, characterized by high I/O, memory, and storage intensity, reflecting its resource demands for transactional operations.

3.2.2 Workload Level. Since a system often runs in diverse workloads, which can profoundly impact the performance [70], `Domland` can analyze the workload level characteristic via two features:

- **Workload Type:** A nominal feature that defines the nominal nature of workload inputs and processing tasks, for example, when tuning the system DConvert, the workloads vary by

Table 2. Domain features of option for configurable systems.

Type		Descriptions	Exampled Option
Functional	Core (F_1)	Governs the system's core logic, directly impacting execution behavior.	`MVSTORE` in H2
	Utility (F_2)	Provides auxiliary functionalities applicable across multiple modules.	`anyScriptOrigin` in Batik
Resource	CPU (R_1)	Controls CPU usage, parallel processing, or multi-threading.	`threads` in Dconvert
	Storage (R_2)	Manages persistent data storage.	`VBRRange` in Jump3r
	Memory (R_3)	Handles cache, memory pools, and buffer.	`blocksize` in Kanzi
	Queue (R_4)	Regulates scheduling, timeouts, and delays.	`FlushTimeout` in Xz

image format such as `png` and `svg`; for system H2, they vary by transaction type (e.g., `tpcc` and `ycsb`).

- **Workload Scale:** An ordinal feature representing magnitude of the workload in terms of input size or operational intensity, e.g., for Dconvert, workloads are divided into small, medium, and large scales of the image; for system x264, workloads are classified into small, medium, and large scales of the resolution/file size.

Similar to the features at the system level, representing a workload with respect to the above features can often be easily achieved with domain understanding of the system.

3.2.3 Option Level. `Domland` encourages domain analysis at the option level, relying on understanding the code logic related to the configuration options. As such, the results would provide fine-grained details on the system's internal working structure. Particularly, we categorize the options using a single nominal feature, representing two groups of types, i.e., ***functional feature*** and ***resource feature***, as shown in Table 2. Specifically, functional options capture logic- or behavior-related mechanisms that determine what a system does, whereas resource options capture control parameters that influence how efficiently the system executes (e.g., CPU, memory, or storage).

When executing the option level analysis, it might require some efforts to categorize a system with respect to the above. Therefore, `Domland` also provides guidelines to that end using both ***manual comprehension*** and ***code analysis*** [61]. Manuals (or API) are software documentation that contains rich information about the configuration options and their implications on the system [61]. In manual comprehension, one can follow the steps below:

(1) **Screening:** Given a list of keywords related to the features above[2], one can filter which function descriptions are relevant to a feature. For example, "compress" and "storing" are common keywords in the manual of H2 for storage options.
(2) **Option backtracking:** Once the functions are categorized into the above features, we can then examine whether each function's description mentions the name of the options (which is often available on `yaml` or other files). For example, `FlushTimeout` is an option for Xz named in its API.
(3) **Feature assignment:** Finally, we say an option belonging to a feature that contains the most relevant functions to that option. For example, in Xz, `FlushTimeout` has most functions belongs to *Queue*, hence it is also of this type.

While manual is important, it can still be misleading. To ensure precision, `Domland` also suggests a semi-automatic code analysis:

[2]Details can be accessed at: https://tinyurl.com/7dt2zrd4.

(1) **Variable identification:** Knowing the names of the options does not mean that we know exactly where they are referred to as variables in the code. To that end, we use three methods to create the option-variable mappings: (1) finding in a centralized configuration file; (2) discovering the variables in the `main` file; and (3) searching the keywords on `setter` and `getter`, then track back to the variables.
(2) **Taint analysis:** When the variables are localized, we then use taint analysis tools [11, 12] to mine all functions that can be affected by a variable, linking functions to the options.
(3) **Feature assignment:** Same as the manual comprehension.

We unionize the features assigned to the same options from both manual comprehension and code analysis, e.g., when an option O is linked to F_1 from manual and to R_1 from code, then eventually we say O is associated with both F_1 and R_1. As such, an option might be linked to more than one feature above. Since manual comprehension and code analysis involve human judgment rather than full automation, we adopt a reliability-controlled labeling process to ensure rigor. Specifically, multiple annotators independently performed the classification and resolved disagreements through discussion or, when necessary, external consultation. We quantified the inter-rater agreement using Cohen's Kappa κ [40] and required $\kappa \geq 0.7$ before finalizing the labels. This multi-annotator, reliability-validated process helps mitigate subjectivity of the option-level categorization.

Considering that manual option classification can be labor-intensive and difficult to scale across large systems, especially when multiple annotators are required to ensure reliability, `Domland` optionally incorporates large language models (LLMs) as additional annotators to reduce human workload and improve scalability. In this setting, the LLM acts as an independent labeling agent, analogous to a human annotator in the multi-annotator workflow. The LLM's outputs are then compared with the human-derived labels, where any conflicts will trigger human review. This governance structure preserves consistency, traceability, and reliability while reducing manual burden.

Specifically, the LLM-assisted workflow consists of the following steps:

(1) **Label/rule definition:** Before applying LLM-based classification, we should first explicitly define the labeling criteria using the domain taxonomy in Table 2. This includes: (i) the top-level distinction between functional options (F_1 : Core, F_2 : Utility) and resource-related options ($R_1 - R_4$: CPU, Storage, Memory, Queue), and (ii) the decision rules for determining which category an option belongs to. These definitions are embedded directly into the prompt template (see Figure 4), guiding the LLM through a transparent reasoning process.
(2) **Input preparation:** For each configuration option, we then construct a structured input that aggregates all available evidence: the option name, a semantic description extracted from manuals/APIs, representative code snippets, and brief system background. These elements are organized in a fixed order (system context → option name → documentation description → code snippets), with the description serving as the primary decision basis and the other information, if any, as auxiliary context.
(3) **LLM inference:** The prepared evidence is then fed into the prompt template (see Figure 4), which already includes the labeling rules defined before and a few short illustrative examples. Given this structured input and guidance, the LLM will produce one label from the closed set $\{F_1, F_2, R_1, R_2, R_3, R_4\}$, encoded as a single ASCII label.
(4) **Label adjudication:** After LLM inference, the LLM-generated labels are compared with the human-derived labels. Any disagreement between the two is explicitly flagged and routed for adjudication. Notably, when clear documentary/code evidence supports one label over the other, the label supported by that evidence should be adopted. When no such

Task Context

You are a software engineer tasked with classifying a set of software configuration options into predefined categories. Your goal is to accurately assign each option to a primary category and sub-category and provide the corresponding symbol.

Reasoning Instructions (with symbolic knowledge)

When given information, follow these steps:

(1) First make a high-level decision: does this option mainly control system functional behavior (feature switches, modes, core logic), or does it mainly regulate resource usage (CPU, storage, memory, queues)? Based on this, choose between the F class (F_1/F_2) and the R class (R_1–R_4).
(2) If it belongs to the F class:
- If it directly determines the core algorithm or main execution flow → classify as F_1 (Core).
- If it acts as an auxiliary tool, is reused across modules, and is not essential to the main execution flow → classify as F_2 (Utility).

(3) If it belongs to the R class:
- Emphasis on thread count, parallelism, CPU usage, or similar aspects → classify as R_1 (CPU).
- Emphasis on disk, database, or file-based persistent storage → classify as R_2 (Storage).
- Emphasis on caches, memory pools, buffers, or memory footprint → classify as R_3 (Memory).
- Emphasis on queue length, scheduling strategy, timeouts, delays, or dispatching → classify as R_4 (Queue).

(4) If multiple categories appear partially applicable, compare which category is the most direct and essential target of the option's effect. Finally keep only a single category.

Example 1:
System Context: Brief background information about the software system S
Option to classify: O
Description: Semantic description $\mathcal{D}(O)$ extracted from manuals or APIs.
Code snippet: Relevant code for options O, $\boldsymbol{C(O)}$
Output: One of $\{F_1, F_2, R_1, R_2, R_3, R_4\}$, chosen according to the above rules.
.....

The Actual Task

Now, please analyze the provided code and configuration options O, then output using the format demonstrated above.

System Context: S **Option to classify:** O **Description:** $\mathcal{D}(O)$ **Code snippet:** $C(O)$

Fig. 4. Prompt template. [ ] denotes the content that will be substituted by actual data.

decisive evidence exists, the disagreement is simply recorded and both labels are retained for subsequent agreement analysis. This ensures that corrections occur only when justified, preserving both the reliability of the labels while keeping manual intervention limited.

Similarly, to maintain consistency with the reliability-controlled process described above, we treat the labels generated across all annotators, including human annotators and the LLMs, as independent labeling sources and assess their agreement using Cohen's Kappa κ. Specifically, we first compute κ among the human annotators to quantify baseline human-human agreement. This yields an initial reliability measure based solely on human judgments. Then, the LLM annotators are incorporated sequentially. Let the consensus label set produced by the human annotators be denoted as $\boldsymbol{\mathcal{H}}$. For each LLM annotator, we compute a separate agreement score against this human consensus. For example, with three LLMs, we can obtain $\kappa(\boldsymbol{\mathcal{H}}, LLM_1)$, $\kappa(\boldsymbol{\mathcal{H}}, LLM_2)$, and $\kappa(\boldsymbol{\mathcal{H}}, LLM_3)$. The overall inter-rater consistency is then defined as the average of the human-human κ and these human-LLM κ values, providing a measure of agreement across all annotators while preventing the disagreements between LLMs from dominating the results. If the overall agreement falls below the commonly accepted threshold ($\kappa < 0.7$), we follow a standard reconciliation procedure used in prior studies [31, 88]:

(1) **Identify disagreements:** We locate the specific options where annotators assigned different categories.
(2) **Evidence-guided review:** For each conflicting option, the annotators jointly review the corresponding manuals and code snippets, discussing the conflicting interpretations and striving to reach a consensus grounded in the documentary/code evidence.

- When the evidence clearly supports one category, the label is corrected accordingly.
- When the evidence remains ambiguous, no forced correction is made at this stage.

(3) **Reassessment:** κ is recalculated over all options to assess whether the disagreements have been sufficiently reduced.

This reconciliation may be repeated when necessary, but only evidence-supported corrections, i.e., those justified by documentation/code, are introduced in each round. Once the overall agreement reaches $\kappa \geq 0.7$, the labeling process is deemed reliable. For any remaining options without decisive evidence, the majority (or most consistently assigned) label across annotators is used. This procedure ensures that every option receives an evidence-grounded and reliable final label.

3.3 Domain-Spatiality Synergy

To synergize domain-spatiality information, we use the three levels of domain knowledge as the root (option, system, and workload), associating each of them with the one (or three) aspect of spatial information from landscape, i.e., global structure, local structure, and ruggedness. This domain-spatiality synergy provides an analytical integration between domain contextual and spatial landscape characteristics. Specifically, domain knowledge (e.g., option category, system property, or workload feature) serves as contextual grounding for interpreting spatial landscape metrics, allowing us to observe and characterize how domain-related factors correspond to variations in landscape properties (e.g., certain option types or systems being associated with higher ruggedness or distinct local structures). The results can be concluded to form an explanation, linking the domain knowledge and the design/selection of tuner/system via properties of the configuration landscape.

3.3.1 Spatial Option-level Synergy. Here, `Domland` suggests to only measure the ruggedness of landscape (via autocorrelation). This is because ruggedness inherently reflects how an option influences performance variability, with certain settings simplifying dramatic neighboring performance fluctuations. While traditional sensitivity and feature-importance analyses (e.g., ANOVA [79], SHAP [64], and LIME [77]) focus on how much a configuration option contributes to overall performance variation, often through mean differences (ANOVA) or model-based attributions (SHAP/LIME), our notion of ruggedness-sensitivity instead captures how an option reshapes the structure of the configuration landscape. Specifically, it measures how the option alters fluctuation intensity across neighboring configurations, as reflected by autocorrelation under the neighborhood structure. From a landscape perspective, this option-level analysis goes beyond identifying performance-critical options; it reveals how certain options affect the search dynamics of different tuning strategies. For example, exploitation-driven tuners (e.g., Hill Climbing [60, 95]) are more sensitive to rugged landscapes, where prioritizing ruggedness-sensitive options might improve convergence efficiency. Tuners with strong exploration (e.g., Genetic Algorithm [16, 74]) are inherently more resilient to ruggedness and may show limited benefit from such prioritization. In contrast, with the classic ANOVA, it is likely to recommend that any tuner that takes this advantage (important options) would be benefited, e.g., both HC and GA, i.e., ANOVA cannot discriminate/link the behaviors of different tuner designs. Hence, our spatial option-level analysis complements feature-importance approaches by uncovering when and why different search strategies succeed or fail under different landscape structures. In this work, we focus on option-specific ruggedness, analyzing how each individual configuration option affects the smoothness of the landscape. This choice allows for interpretable insights into option-level structural influences while keeping the analysis consistent with the option-level perspective in Section 3.2.3 where each configuration option is examined independently. To evaluate how a single option's change influences the ruggedness of the configuration landscape, we perform the following on the profiled configuration data for each option:

(1) **Data partitioning:** Given a configuration option taking different values, we partition the dataset into symmetric subsets based on the option's values, ensuring all other factors remain identical except for the investigated option.
(2) **Autocorrelation computation:** Once the subsets are obtained, we can then calculate the autocorrelation metric for each subset, quantifying the ruggedness of the corresponding configuration landscape.
(3) **Impact assessment:** Finally, we evaluate the option's effect by analyzing the difference in autocorrelation values between subsets and computing the relative standard deviation (RSD) to measure the variability in ruggedness.

To link the outcome of ruggedness assessment to the types of options, one can summarize the common patterns on the interpretation of the ruggedness and the option's type in several ways:

- For a single system-workload pair.
- For all workloads of a system.
- For all workloads across all systems.

There is no requirement to complete all of the above actions, but a subset of them can be selected depending on the needs. Each of the above could lead to a finding. Even if no clear pattern can be observed, a conclusion related to all systems in general can also be made. For example, if it is observed that for a system $\mathcal{S}$ changing the core options (F_1), even across workloads, can lead to rugged configuration landscape ($r < 0.2$), we can find that: "*Core options exert a stronger impact on shaping the ruggedness of system $\mathcal{S}$".*

3.3.2 Spatial System-level Synergy. Here, we recommend using all landscape metrics included in `Domland`. Since each workload can significantly impact the configuration landscapes, we have a total of T landscapes to analyze where T is the total number of system-workload pairs.

To link the outcomes of landscape metrics to the features of systems, one can summarize the common patterns concerning each landscape metric with respect to system-level features for all systems. For example, we might find that "*the configuration landscape is generally highly rugged regardless of the systems and their features*"; yet, in another example, if it is found that most systems within an area (e.g., database) have a larger proportion of local optima than others (e.g., compression), the finding could be: "*database systems are prone to getting trapped into local optima compared to compression systems, requiring tuners with stronger global exploration capabilities to avoid early stagnation".*

Additionally, multiple landscape metrics can be combined to create a new connection to the domain knowledge. For example, if it is observed that in most systems with different workloads, the performance of local optima demonstrate a strong Spearman correlation ($\rho > 0.69$) with their basins of attraction, our conclusion is: "*in most software systems, high-quality local optima tend to exhibit larger basins".*

Since the system and option features can be correlated, the above outcomes can also be linked to the commonality of option types, but we are less concerned about the options' changes. For example, if it is found that those systems with a larger proportion of core options consistently show more rugged landscapes, the finding could be: *"tailoring tuners equipped with global exploration might be promising for tuning those systems with a higher proportion of core options".*

3.3.3 Spatial Workload-level Synergy. Again, for workload types and scales, all landscape metrics in `Domland` can be used. To correlate the outcomes of landscape metrics to those domain features, one can summarize the common patterns concerning each landscape metric with respect to workload level features in two ways:

- For all workloads of a system.

- For all workloads across all systems.

Each of the above actions can be chosen depending on the needs. For example, if no clear patterns can be observed between workload features and the landscape metrics, then the finding can be: "*workload effects on ruggedness are not uniformly tied to type or scale. Both contribute to variations, but their impact is system-dependent*". It is also possible to link the conclusions to the commonality of system/option level features. For example, if a system $\mathcal{S}$ exhibits a greater ruggedness fluctuation only in particular contexts (e.g., under a workload feature and an option type), the finding can be: "*there might be interactions between a particular workload feature and option type, making tuning under such extremely challenging [98].*"

Again, the metrics can be combined as, e.g., correlation, to further link to the domain knowledge.

4 How to Use `Domland`? A Case Study

To evaluate how `Domland` can be used, we conduct a case study on real-world configurable systems, aiming to answer the following research questions (RQs):

- **RQ1 (Option-level):** How do options shape the spatiality of configuration landscape?
- **RQ2 (System-level):** What spatiality can be extracted across the systems?
- **RQ3 (Workload-level):** How do the workloads impact configuration landscapes?

Overall, these research questions focus on characterizing configuration tuning problems by jointly examining configuration landscape spatiality and domain knowledge at the option, system, and workload levels, with the goal of deriving insights for interpreting tuning difficulty and informing tuning algorithm design.

Specifically, **RQ1** seeks to explore how configuration options and their domain characteristics (*functional* and *resource* options) shape the spatiality of the configuration landscape. **RQ2** investigates spatial characteristics across different configurable systems. By comparing multiple systems, this question examines whether domain attributes at the system level (e.g., system types, programming language, or resource intensity) lead to specific spatial patterns in their configuration landscapes. **RQ3** analyzes how workload features influence configuration landscapes within the same system. By comparing landscape metrics across workloads of different types and scales, this question aims to uncover whether specific workload features consistently shape the landscape structure. Through answering these research questions, we reveal domain-spatiality patterns in configuration tuning and demonstrate how these patterns can be translated into actionable insights (see Sections 5.4 and 5.5) and practical implications (see Section 6).

4.1 Systems Profiling

Our methodology is designed to be generally applicable to any configurable system. In practice, however, conducting large-scale configuration evaluations from scratch would be prohibitively time-consuming and resource-intensive. For example, measuring a single configuration of MySQL often takes several minutes even under controlled conditions [42]. To ensure both representativeness and practical feasibility, our case study uses data collected by Muhlbauer et al. [70] using selective sampling [54, 78], which provides extensive performance measurements across a wide range of configurable systems and diverse workload settings. In total, this dataset covers nine systems and 93 workloads, including roughly 25,258 configurations. We chose this dataset for two key reasons:

- **Comprehensive coverage:** Compared with other publicly available datasets, which often evaluate configurations under a single workload [71, 91] or focus on less diverse systems [47, 51, 52], the dataset collected by Muhlbauer et al. [70] spans multiple domains and provides 6-13 diverse workloads per system. Such diversity enables our multi-level analysis across options, systems, and workloads.

Table 3. Subject system characteristics. The workload details can be accessed via https://tinyurl.com/546svkb2.

System	Language	Area	Performance	Version	Resource Intensity	LOC	#O	#C	#W
JUMP3R [1]	Java	Audio Encoder	Runtime	1.0.4	CPU	29685	13	4196	6
KANZI [5]	Java	File Compressor	Runtime	1.9	I/O; Memory	28614	18	4112	9
DCONVERT [3]	Java	Image Scaling	Runtime	1.0.0-α7	I/O; Memory	6888	10	6764	12
H2 [2]	Java	Database	Throughput	1.4.200	I/O; Memory; Storage	333539	16	1954	8
BATIK [9]	Java	SVG Rasterizer	Runtime	1.14	CPU	360924	8	1919	11
Xz [7]	C/C++	File Compressor	Runtime	5.2.0	CPU	43130	12	1999	13
LRZIP [6]	C/C++	File Compressor	Runtime	0.651	CPU	20797	7	190	13
X264 [10]	C/C++	Video Encoder	Runtime	baee400…	CPU; I/O	86740	25	3113	9
Z3 [8]	C/C++	SMT Solver	Runtime	4.8.14	CPU	636268	12	1011	12

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

- **Transparency:** The dataset provides detailed metadata for each system, including system versions, configuration options, workload sources, and corresponding system source code. Such transparency allows us to precisely trace how each configuration and workload is defined and executed, enabling domain-level analyses such as taint analysis in our option-level study.

Details can be found at the repository of Muhlbauer et al.: https://zenodo.org/records/7504284.

4.1.1 Systems. The datasets cover well-known configurable systems, which have been widely studied in the literature [70, 75, 86, 90]. The chosen systems span diverse application areas, performance objectives, and languages, providing a robust foundation for evaluating system characteristics, as shown in Table 3. The system features can be easily summarized based on domain expertise. More details on how to use these systems for experiments can be found in [4, 70].

4.1.2 Workloads. We use data measured on various workloads as collected by Muhlbauer et al. [70] (see Table 3). These workloads again have diverse types and scales, which can be directly summarized by checking the specifications of those workloads. For example, the system X264 has various workloads that consist of different raw video frames, diverse resolutions, and file sizes.

Specifically, the workloads used for each system are summarized as follows:

- **JUMP3R** encodes raw WAVE audio signals to MP3, with variations in file size, signal length, sampling rate, and number of channels.
- **X264** encodes raw video frames (y4m format) across different resolutions (low/DS up to 4K) and file sizes.
- **KANZI, Xz, and LRZIP** compress mixed-type files (text, binary, structured data, etc) or single-type files from community benchmarks. Additionally, custom datasets, including the Hubble Deepfield image and a Linux kernel binary, are used to augment the workloads. For Xz and LRZIP, different parameterizations of UIQ2 benchmark[3] are added to study the effect of varying file size.
- **Z3** evaluates the satisfiability of logical problems in SMT2 format. The workload consists of the six longest-running instances from Z3's performance test suite, augmented with additional instances from the SMT2-Lib[4] repository to increase diversity in logic types.
- **BATIK** transforms SVG vector graphic into a bitmap. The workload consists of resources from the Wikimedia Commons collection[5], varying primarily in file size.

[3]http://mattmahoney.net/dc/uiq/
[4]https://smt-comp.github.io/2017/benchmarks.html
[5]https://commons.wikimedia.org/wiki/Category:Images

- **H2** executes database transactions using different OLTPBench [37] benchmarks (`Voter`, `SmallBank`, `TPC-H`, `YCSB`). The scale factor is varied to adjust transaction complexity.
- **DConvert** transforms resources (e.g., image files) at different scales. The workload includes various input formats (`JPEG`, `PNG`, `PSD`, `SVG`) with variations in file size.

In summary, compared with other publicly available datasets that usually evaluate configuration under a single workload or a narrow set of input scenarios [51, 52, 71, 91], the dataset collected by Muhlbauer et al. [70] provides a substantially more comprehensive workload coverage. Each of the nine systems is tested under 6-13 distinct workloads, spanning different workload types, scales, and complexities.

4.1.3 Options. We use the same options as Muhlbauer et al. [70]. Each system has varying option types (e.g., integer, boolean, and enumerated) and dimensions. To categorize the options, we follow the procedure of `Domland` as discussed in Section 3.2.3. Specifically, the options were independently labeled by the authors and by three LLM-based annotators (DeepSeek[6], Qwen[7], and GPT-5[8]). The LLMs received the domain taxonomy and prompt template introduced in Section 3.2.3, while human annotators applied the same taxonomy and decision rules when reading manuals and source code. We then computed the inter-rater agreement over these labels using Cohen's Kappa κ, following the procedures described in Section 3.2.3. First, we assessed the agreement between the two human annotators across all 121 options. In this initial round, the annotators agreed on 93 options and disagreed on 28 options, yielding an initial κ of 0.6700. To resolve these disagreements, we followed the adjudication procedure described in Section 3.2.3: examining disagreements, checking the supporting documentation and code snippets, and updating labels only when the evidence clearly favored a particular category. After this reconciliation step, the two annotators reached a κ of 0.7059, forming the evidence-grounded human consensus label set. Next, we compared this human consensus label set with the labels produced by each LLM-based annotator (i.e., DeepSeek, Qwen, and GPT-5), obtaining individual Cohen's Kappa κ values of 0.6870, 0.7513, and 0.7357, respectively. Finally, we summarize the overall agreement by averaging the consistency scores across all annotators, including humans and LLMs, and obtain an overall $\kappa = 0.718$, demonstrating substantial agreement among annotators. The full labeled option set is available in our repository: https://tinyurl.com/4stvjurm.

4.2 Customizing Fitness Landscape Analysis

4.2.1 Neighborhood Construction. In this case study, we use Hamming distance to define a configuration's neighborhood. Generally, when using Hamming distance as the distance metric, the neighborhood radius is set to $\epsilon = 1$, i.e., $\mathcal{N}(\boldsymbol{c}) = \{\boldsymbol{c}' \mid d(\boldsymbol{c}', \boldsymbol{c}) \leq 1\}$ (such that the number of neighbors is n, where n is the configuration dimensionality) [67]. However, when applying configuration spatiality analysis of `Domland` to sampling-based datasets [70], the strict definition of a neighborhood as $d(\boldsymbol{c}', \boldsymbol{c}) \leq 1$ can leave many configurations without neighbors, particularly in sparse regions. Thus, in this case study, the neighborhood radius is expanded to ensure that the expected average number of neighbors for configurations is n, where n is the configuration dimensionality. This adjustment mitigates sparsity while ensuring meaningful connectivity for analysis and aligns with the neighborhood size in a complete binary configuration space, where each configuration has exactly n neighbors at Hamming distance 1. Notably, in our case study, all metrics are computed following this neighborhood structure, ensuring consistency in landscape analysis.

[6]https://www.deepseek.com/en
[7]https://qwen.ai/home
[8]https://chatgpt.com/

Table 4. Comparison of fitness landscape metrics computed under different neighborhood definitions and sampling granularities, where results from the full-scale dataset serve as the ground truth. ℓ_p denotes the proportion of local optima; ℓ_q represents the quality relative to the global optimum; d is neighborhood radius. "Fixed" uses a constant neighborhood radius, i.e., $d = 1$, whereas "Adaptive" applies the proposed sparsity-aware radius.

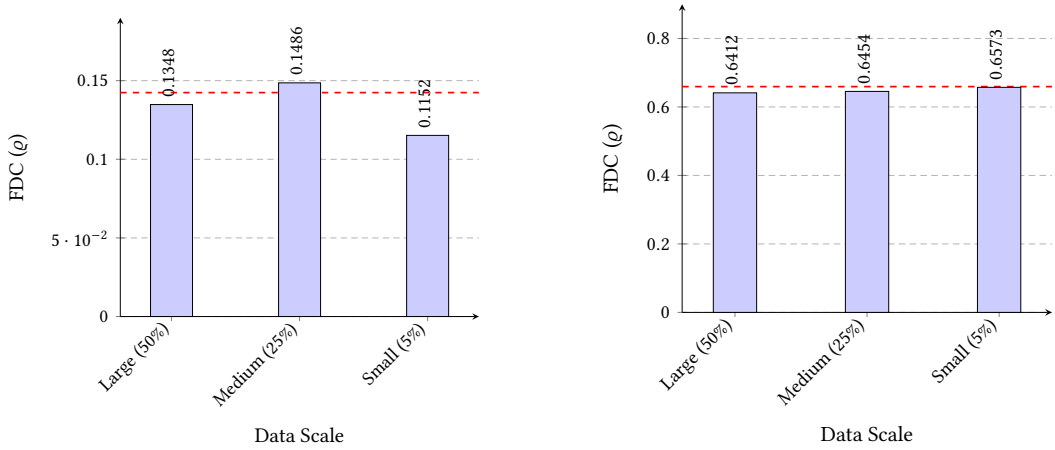
System		7z				LLVM			
Method	**Data Scale**	ℓ_p	ℓ_q	**Basin of Attraction**	d	ℓ_p	ℓ_q	**Basin of Attraction**	d
Fixed	Large (50%)	0.00758	0.85737	0.02092	1	0.01230	0.32979	0.08344	1
	Medium (25%)	0.03963	0.54504	0.00804	1	0.12177	0.15062	0.02307	1
	Small (5%)	0.43036	0.11389	0.00000	1	0.66575	0.02754	0.00000	1
Adaptive	Large (50%)	0.00758	0.85737	0.02092	1	0.00009	0.92532	0.68311	2
	Medium (25%)	0.03963	0.54504	0.00804	1	0.00018	0.90927	0.82104	2
	Small (5%)	0.00932	0.91993	0.13520	2	0.00061	0.93064	0.92247	3
Ground Truth	Full	0.00178	0.91104	0.15392	1	0.00003	0.91490	0.97508	1

To evaluate the validity of adaptive neighborhood construction and examine whether it can effectively mitigate sparsity-induced distortion in landscape characterization, we conducted an empirical analysis comparing it with the conventional fixed radius approach. In this validation, we selected two systems, 7z and LLVM, from an open-source dataset [91], as they possess high-dimensional configuration space representative of real-world tuning scenarios and provide complete configuration-performance data for compute ground-truth landscape metrics.

Specifically, the validation process consists of four main steps:

1) We first computed a set of representative fitness landscape metrics on the complete datasets, including the proportion of local optima ℓ_p, the quality of local optima ℓ_q, and the basin size of the global optimum (against the whole space). These results, obtained using the standard neighborhood radius of Hamming distance $d = 1$, serve as the ground truth.
2) From each full dataset, we then generated three subsets using Latin Hypercube Sampling (LHS) [68] with sampling rates of 50%, 25%, and 5%, corresponding to large, medium, and small data scales, respectively.
3) On each subset, we applied two neighborhood construction strategies: the conventional fixed radius ($d = 1$) and the adaptive radius, where the neighborhood radius is adjusted according to data sparsity to preserve meaningful connectivity.
4) We computed the same landscape metrics for all settings and compared them with the ground truth;

As shown in Table 4, applying a fixed neighborhood radius under sparse sampling leads to substantial distortion in the computed landscape metrics, and this distortion becomes more pronounced as the sampling density decreases. At the small data scale, for example, the number of detected local optima increases sharply, hundreds (0.43036/0.00178) or even thousands (0.66575/0.00003) of times greater than the ground truth, because many isolated configurations are incorrectly identified as local optima, which in turn degrades the overall estimation of local optima quality. Moreover, the measured basin of attraction of the global optimum collapses to zero for both 7z and LLVM, indicating that sparse sampling disrupts the continuity of configuration transitions toward the global optimum. In contrast, the adaptive neighborhood radius effectively mitigates this sparsity-induced bias, yielding landscape metrics that closely approximate those obtained from the full configuration space. Meanwhile, we also observed similar results on other systems from the same open-source dataset that provides complete configuration-performance data. These findings confirm that the adaptive neighborhood radius maintains meaningful connectivity and preserves the structural characteristics of the configuration landscape even under incomplete sampling. Notably, when



(a) 7z (b) LLVM

Fig. 5. Comparison of FDC (ϱ) under different sampling densities. The red dashed line indicates the FDC value computed from the full-scale dataset (ground truth).

the dataset itself already exhibits sufficient connectivity, as in the case of full or dense sampling, the adaptive neighborhood radius naturally switches to the standard definition with Hamming distance $d = 1$, thereby avoiding the introduction of any additional bias.

4.2.2 Landscape Metrics. The selection of landscape metrics can follow what has been listed in Section 3.1.3 as part of `Domland`. The detailed definitions and usage are as follows:

(1) Fitness Distance Correlation (FDC). FDC is a widely used metric for evaluating the relationship between the fitness (e.g., runtime and throughput) of solutions (i.e., configurations) and their distance to the global optimum in a given landscape [53]. It helps to characterize the global structure of the fitness landscape and evaluate the ease or difficulty of navigating toward the global optimum. Formally, FDC is calculated as:

$$\varrho(f, d) = \frac{1}{\sigma_f \sigma_d} \frac{1}{s} \sum_{i=1}^{s} \left(f_i - \bar{f}\right)\left(d_i - \bar{d}\right), \tag{2}$$

where s is the number of solutions considered in FDC. Within such a solution set, d_i denotes the distance of ith solution to the nearest global optimum. $\bar{f}$ ($\bar{d}$) and σ_f (σ_d) are the mean and standard deviation of fitness (and distance). In practice, since the true global optimum is typically unknown in high-dimensional configuration spaces, we approximate it using the best-performing configuration observed within the sampled configurations, following the setting adopted in the literature [48, 82]. To verify the reliability of using the observed best-performing configuration as the proxy of global optimum, we conducted an additional validation using the same systems and sampled subsets as in Section 4.2.1. We first computed the FDC on the full dataset as the ground truth. Then, for the three sampled subsets (with 50%, 25%, and 5% sampling rates), we calculated FDC by using the best-performing configuration within each subset as the proxy for the global optimum. As shown in Figure 5, the red dashed line denotes the FDC value computed from the full-scale dataset, which serves as the ground truth reference. The FDC values obtained across the three sampled subsets (50%, 25%, and 5%) remain close to this ground truth, confirming that using the best-performing configuration observed within a subset provides a reliable approximation even

under a sparse sampling subset. Similar patterns were also observed in other systems with complete configuration–performance data [91].

(2) Local optima. Local optima capture the underlying local structural properties of a problem's landscape. Formally, a configuration $\boldsymbol{c}^\ell$ is considered as a local optima if $f(\boldsymbol{c}^\ell)$ is better than $f(\boldsymbol{c})$, $\forall \boldsymbol{c} \in \mathcal{N}(\boldsymbol{c}^\ell)$, where $\mathcal{N}(\boldsymbol{c}^\ell)$ is the neighborhood of $\boldsymbol{c}^\ell$. Two key aspects are considered when analyzing local optima: (1) the number of local optima; and (2) the quality of local optima.

- **The number of local optima.** The number of local optima reflects landscape multimodality, indicating the presence of multiple high-performing regions and complexity of the problem. To quantify this, we define ℓ_p as the proportion of local optima among all sampled configurations:

$$\ell_p = \frac{|\mathcal{L}|}{|\mathcal{S}|}, \tag{3}$$

  where $\mathcal{L}$ and $\mathcal{S}$ denote the sets of local optima and all sampled configurations, respectively.
- **The quality of local optima.** The quality of local optima, particularly their relative performance compared to the global optima, indicates the prominence of local peaks and the susceptibility of the landscape to trapping suboptimal solutions. For minimization problems, we define the relative quality of local optima as:

$$\ell_q = \frac{\frac{1}{|\mathcal{S}|}\sum\limits_{\boldsymbol{c}\in S} f(\boldsymbol{c}) - \frac{1}{|\mathcal{L}|}\sum\limits_{\boldsymbol{c}^\ell\in\mathcal{L}} f(\boldsymbol{c}^\ell)}{\frac{1}{|\mathcal{S}|}\sum\limits_{\boldsymbol{c}\in\mathcal{S}} f(\boldsymbol{c}) - f(\boldsymbol{c}^g)}, \tag{4}$$

  where $\boldsymbol{c}^g$ represents the global optimum, and $\boldsymbol{c}^\ell$ denotes a local optima from $\mathcal{L}$. A higher ℓ_q indicates that local optima are of higher quality, being closer to the global optimum, while a lower ℓ_q suggests an abundance of low-quality local optima.

(3) Basin of attraction. The basin of attraction [66] of a local optimum $\boldsymbol{c}^\ell$, denoted as $\mathcal{B}(\boldsymbol{c}^\ell)$, is the set of all configurations that converge to $\boldsymbol{c}^\ell$ under a local search process. Formally, it can be defined as:

$$\mathcal{B}(\boldsymbol{c}^\ell) = \{\boldsymbol{c} \in C \mid \text{LocalSearch}(\boldsymbol{c}) \rightarrow \boldsymbol{c}^\ell\}. \tag{5}$$

Here, we adopt hill climbing with the first-improvement strategy as the local search algorithm. The step size follows the neighborhood structure defined in Section 4.2.1. In other words, if the neighborhood radius is set to d, the local search step size is also defined as d to ensure consistency in basin identification. As a result, the basin of attraction of a local optimum is identified as the set of all configurations that lead to it under this search process.

(4) Autocorrelation. Autocorrelation [92] quantifies the ruggedness of a fitness landscape by measuring how fitness values change across neighboring configurations. Formally, it is defined as:

$$r(d) = \frac{\sum_{i=1}^{N-d}\left(f(\boldsymbol{c}_i) - \bar{f}\right)\left(f(\boldsymbol{c}_{i+d}) - \bar{f}\right)}{\sum_{i=1}^{N-d}\left(f(\boldsymbol{c}_i) - \bar{f}\right)^2}, \tag{6}$$

where $f(\boldsymbol{c}_i)$ is the fitness of ith visited configuration $\boldsymbol{c}_i$ in the random walk [83], $\boldsymbol{c}_{i+d}$ is the next visited neighbor at step size d, and N is the walk length. A high $r(d)$ indicates a smooth landscape, where similar configurations yield comparable fitness values, while a low autocorrelation suggests a rugged landscape with abrupt fitness variations. Notably, to accommodate analysis on sampled datasets, our random walk starts with a step size of $d = 1$. If no neighbor is found, d is incrementally increased until a valid neighbor is located, ensuring continuity. The walk

Table 5. The relative standard deviation (RSD) of autocorrelation across configuration options in nine software systems. For each case, blue cells indicate options with significant influence on landscape ruggedness (RSD ≥ 5%); or red cells otherwise. The "Type" column denotes the functional-related or resource-related roles of options as categorized in Table 2. The ✗ markers indicate that there are insufficient data samples for analysis.

Option	Lrzip		Xz		Z3		Dconvert		Batik		Kanzi		X264		H2		Jump3r	
	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)	Type	RSD (%)
O1	F_1	107.254	R_3	2.118	F_2	1.333	F_1	7.422	F_2	0.14	F_1	1.136	F_1	1.162	R_1	6.703	F_1	1.695
O2	F_2	210.332	F_1	7.29	F_2	3.881	F_1	1.105	F_2	0.376	F_1	121.468	F_1	6.467	R_1	5.301	F_1	✗
O3	F_2	83.073	F_1	19.927	F_1	0.276	F_2	0.2	F_1	0.705	F_1	0.081	F_1	11.448	R_1	5.947	F_1	1.412
O4	R_1	499.477	R_3	20.944	F_1	1.182	F_1	0.099	F_2	1.224	F_1	164.22	F_1	8.785	R_1	0.133	F_1	✗
O5	F_1	306.08	F_1	2.042	F_1	2.047	F_1	0.615	F_2	0.476	F_1	32.901	F_1	13.159	R_1	0.152	F_1	9.008
O6	F_1	67.485	R_1	0.288	F_2	3.192	F_1	0.679	F_2	1.354	F_1	3.946	F_1	10.602	R_3	23.42	F_1	1.418
O7	F_2	57.613	R_3	23.014	F_2	0.875	R_1	0.086	F_2	0.556	F_1	18.394	F_1	7.546	F_1	18.454	F_2	6.916
O8			R_4	5.157	F_2	1.07	F_2	2.224	F_2	1.38	F_1	11.745	F_1	9.864	F_1	7.833	F_2	10.665
O9			F_1	2.822	F_2	1.103	F_2	3.458			F_1	20.259	F_1	1.393	F_2	5.601	R_1	✗
O10			F_2	84.806	F_2	0.302	F_2	35.124			F_1	58.027	F_1	15.91	R_1	0.809	R_1	24.122
O11			F_1	23.679	F_1	1.343					F_2	2.179	F_1	1.717	F_1	4.294	F_2	1.796
O12			F_1	✗	F_2	0.312					F_1	8.859	F_2	0.168	R_2	0.359	R_2	18.146
O13											F_1	25.665	F_1	3.63	F_2	0.286	F_1	9.132
O14											F_1	18.236	F_1	0.196	R_2	6.037		
O15											F_1	216.355	F_1	5.904	F_2	16.952		
O16											R_1	142.712	F_2	12.122	F_1	28.767		
O17											R_3	46.585	F_1	10.031				
O18											F_1	42.856	F_1	11.83				
O19													R_1	1.692				
O20													R_1	4.996				
O21													R_1	19.874				
O22													F_1	15.94				
O23													F_1	6.854				
O24													R_1	0.671				
O25													R_1	5.52				
RSD ≥ 5%	7/7		7/11		0/12		2/10		0/8		14/18		16/25		10/16		6/10	

proceeds until all configurations in the dataset are visited, recording the total walk length N. Finally, autocorrelation is computed using Equation (6).

5 Results and Analysis

5.1 RQ1: Spatial Option-level Synergy

Here, we follow the procedures in Section 3.3.1 and summarize the common patterns for all workloads across all systems. To simplify exposition, we aggregate results across workloads by taking the median autocorrelation. As such, we focus on findings for *all workloads of a system* and *all workloads across all systems* in `Domland`.

The results are summarized in Table 5. Here, we adopt a 5% variation threshold to characterize whether a configuration option has a significant impact on ruggedness, aligning with the setting of the significant variation factor in the field [30, 61]. This threshold is widely accepted in configurable system analysis because performance fluctuations below 5% are often attributed to measurement noise or environmental variability rather than genuine configuration effects. Therefore, adopting this level helps distinguish substantive configuration/option-induced changes from normal experimental variance, ensuring both consistency with prior work and practical interpretability of ruggedness sensitivity. As can be seen, in Lrzip, all investigated options exhibit a significant effect (RSD ≥ 5%) on ruggedness. Systems such as Xz, Kanzi, X264, H2, and Jump3r show a relatively high proportion of impactful options, taking more than half of the options. In contrast, Dconvert displays a lower proportion of options significantly affecting ruggedness. Notably, the ruggedness of Z3 and Batik demonstrate a striking insensitivity to option variations, with nearly all options having negligible impact on landscape ruggedness. On the other hand, the degree of impact also

Table 6. The FDC (ϱ) across nine systems with different workloads (blue cells : $\varrho \geq 0.15$; red cells : $-0.15 < \varrho < 0.15$; gold cells : $\varrho \leq -0.15$).

Workload	Lrzip	Xz	Z3	Dconvert	Batik	Kanzi	X264	H2	Jump3r
W1	0.025	0.381	0.488	0.461	0.319	0.207	-0.083	0.637	0.439
W2	0.186	0.460	0.535	0.524	0.280	0.328	-0.015	0.627	0.067
W3	-0.037	-0.198	0.545	0.468	0.520	0.123	-0.014	0.641	0.363
W4	-0.078	-0.099	0.492	0.511	0.550	0.181	0.065	0.257	0.267
W5	-0.050	0.566	0.098	0.523	0.360	0.123	0.031	0.637	0.443
W6	0.019	0.612	0.018	0.524	0.438	0.192	0.155	0.653	0.117
W7	0.196	0.262	0.318	0.512	0.341	0.334	-0.045	0.405	
W8	-0.075	0.383	0.488	0.232	0.252	0.150	0.015	0.336	
W9	-0.025	0.504	0.538	0.471	0.428	0.138	0.148		
W10	0.016	0.456	0.372	-0.005	0.583				
W11	-0.072	0.450	0.225	0.292	0.188				
W12	0.026	0.479	0.245	0.345					
W13	0.073	0.565							
Median	**0.016**	**0.456**	**0.430**	**0.469**	**0.360**	**0.181**	**0.015**	**0.632**	**0.315**
IQR	**0.076**	**0.123**	**0.263**	**0.183**	**0.180**	**0.070**	**0.080**	**0.250**	**0.265**

differs considerably—Lrzip exhibits extreme sensitivity with RSD values reaching up to 499%, whereas in Batik, the most influential option only induces an RSD of 1.38%.

> ***Finding 1:*** *The proportion and magnitude of ruggedness-sensitive options vary greatly within and across systems.*

A further observation is that core options (F_1) account for the majority (37 out of 62) of impactful options where RSD $\geq$ 5%, far exceeding other types (F_2: 10, R_1: 8, R_2: 2, R_3: 4, R_4: 1). This implies a strong connection between core options and landscape ruggedness.

> ***Finding 2:*** *Core options (F_1) exhibit a stronger impact on landscape ruggedness than the other types.*

5.2 RQ2: Spatial System-level Synergy

5.2.1 Fitness Distance Correlation (ϱ). The FDC results across different software systems and workloads are summarized in Table 6. As observed, the majority of systems (7 out of the 9) exhibit $\varrho \geq 0.15$, indicating a positive correlation between performance and proximity to the global optimum. However, the FDC values of Lrzip and X264 tend to fall into the range of $-0.15 \leq \varrho \leq 0.15$, suggesting irregular landscapes where proximity offers little guidance.

> ***Finding 3:*** *Apart from a few edge cases, configuration landscapes can generally provide rich guidance to a tuner.*

5.2.2 Basin of Attraction & Local Optima. For this, the results are summarized in Tables 7 and 8. (1) First, the proportion of configurations (against the whole search space) falling to the global optimum's basin are reported in Table 7. Overall, based on the median values across workloads for each system, most systems exhibit a proportion below 0.21, suggesting that the global optimum is not effortlessly attainable. To further investigate the relationship between local optima superiority and their basin sizes, we assessed Spearman correlation (ρ) between them, as shown in Figure 6. Overall, most systems show a moderate ($0.39 < \rho \leq 0.69$) and strong ($0.69 < \rho \leq 1$) correlations [21, 89],

Table 7. The proportion of configurations in the dataset that falls within the global optimum's basin of attraction. For each cases, the blue cells indicate basin size of the global optimum is larger than 0.21; or red cells otherwise.

Wrokload	LRZIP	XZ	Z3	DCONVERT	BATIK	KANZI	X264	H2	JUMP3R
W1	0.037	0.017	0.643	0.079	0.162	0.35	0.052	0.079	0.037
W2	0.565	0.449	0.735	0.107	0.11	0.239	0.004	0.06	0.081
W3	0.063	0.04	0.753	0.051	0.077	0.081	0.004	0.033	0.074
W4	0.052	0.121	0.142	0.107	0.09	0.301	0.013	0.045	0.109
W5	0.131	0.116	0.542	0.146	0.176	0.331	0.02	0.061	0.101
W6	0.304	0.102	0.642	0.085	0.109	0.11	0.003	0.079	0.032
W7	0.079	0.338	0.606	0.408	0.217	0.226	0.034	0.054	
W8	0.141	0.041	0.26	0.137	0.241	0.266	0.206	0.05	
W9	0.073	0.021	0.375	0.066	0.078	0.17	0.005		
W10	0.141	0.039	0.159	0.032	0.384				
W11	0.12	0.03	0.359	0.036	0.179				
W12	0.068	0.015	0.514	0.115					
W13	0.126	0.036							
Median	**0.12**	**0.04**	**0.528**	**0.096**	**0.162**	**0.239**	**0.013**	**0.057**	**0.078**
IQR	**0.073**	**0.086**	**0.307**	**0.058**	**0.098**	**0.131**	**0.03**	**0.017**	**0.05**

Table 8. The proportion and quality of local optima across workloads for each system. The blue cells, red cells, and gold cells indicate local optima with high ($\ell_q \geq 0.67$), medium ($0.33 \leq \ell_q \leq 0.67$), and low quality ($\ell_q \leq 0.33$).

Workload	LRZIP		XZ		Z3		DCONVERT		BATIK		KANZI		X264		H2		JUMP3R	
	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q	ℓ_p	ℓ_q
W1	0.120	0.972	0.025	0.767	0.015	0.628	0.006	0.787	0.017	0.754	0.086	-0.333	0.180	0.096	0.026	0.244	0.041	0.376
W2	0.194	0.992	0.203	0.592	0.125	0.784	0.005	0.753	0.009	0.906	0.105	-0.234	0.164	0.072	0.025	0.071	0.042	0.534
W3	0.099	0.987	0.025	0.749	0.024	0.967	0.026	0.569	0.010	0.788	0.147	0.421	0.176	0.098	0.024	0.232	0.044	0.510
W4	0.105	0.945	0.021	0.818	0.016	0.966	0.005	0.739	0.011	0.890	0.084	-0.338	0.173	0.075	0.028	0.316	0.048	0.450
W5	0.099	0.830	0.025	0.633	0.012	0.992	0.008	0.718	0.011	0.879	0.086	-0.247	0.176	0.092	0.030	0.101	0.042	0.396
W6	0.099	0.959	0.029	0.673	0.022	0.996	0.018	0.540	0.016	0.826	0.089	-0.296	0.264	0.146	0.027	0.060	0.056	0.467
W7	0.141	0.904	0.045	0.732	0.057	0.829	0.004	0.610	0.008	0.964	0.092	-0.282	0.194	0.138	0.028	0.283		
W8	0.089	0.974	0.021	0.680	0.070	0.973	0.009	0.652	0.012	0.857	0.087	-0.293	0.246	0.143	0.028	0.376		
W9	0.089	0.972	0.021	0.694	0.147	0.938	0.016	0.578	0.022	0.767	0.089	0.266	0.228	0.129				
W10	0.115	0.945	0.020	0.774	0.020	0.803	0.033	0.671	0.011	0.900								
W11	0.120	0.935	0.019	0.698	0.013	0.695	0.020	0.510	0.007	0.964								
W12	0.105	0.951	0.023	0.792	0.010	0.995	0.019	0.775										
W13	0.110	0.980	0.025	0.637														
Median	**0.105**	**0.959**	**0.025**	**0.698**	**0.021**	**0.952**	**0.012**	**0.661**	**0.011**	**0.879**	**0.089**	**-0.282**	**0.180**	**0.098**	**0.028**	**0.238**	**0.043**	**0.458**
IQR	**0.021**	**0.029**	**0.005**	**0.094**	**0.046**	**0.180**	**0.013**	**0.167**	**0.004**	**0.096**	**0.006**	**0.062**	**0.053**	**0.045**	**0.003**	**0.198**	**0.005**	**0.090**

especially for complex systems with larger LOC like Z3, BATIK, and H2, indicating that higher-quality local optima generally attract more configurations, forming larger basins; (2) Second, the proportion (ℓ_p) and quality (ℓ_q) of local optima are depicted in Table 8. The median proportion of local optima across workloads ranges from 1.1% to 18%, highlighting the multimodal nature of configuration landscapes. However, these local optima exhibit promising quality—LRZIP, XZ, Z3, DCONVERT, and BATIK contain mainly high-quality local optima ($\ell_q \geq 0.67$), while JUMP3R mainly falls within the moderate-quality range ($0.33 < \ell_q < 0.67$); In contrast, KANZI, X264, and H2 tend to possess lower-quality local optima ($\ell_q \leq 0.33$).

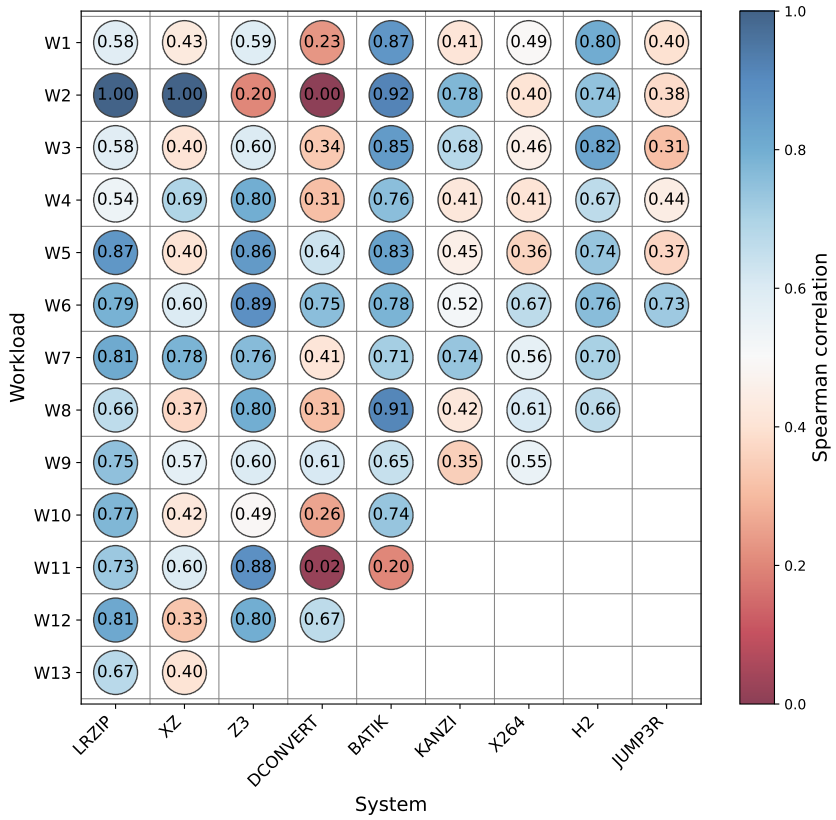


Fig. 6. The Spearman correlation between local optima superiority and basin size across workloads per system.

> ***Finding 4:*** *The global optimum's basin of attraction is not dominant and configuration landscapes have many superior local optima (especially for complex systems), often with larger basin size.*

5.2.3 Autocorrelation. The autocorrelation results are summarized in Table 9. Based on median values across workloads, different systems exhibit varying ruggedness—Xz, Z3, Dconvert, Batik, and H2 demonstrate a relatively smooth landscapes ($r \geq 0.5$); Kanzi, X264, and Jump3r fall into the moderately rugged category ($0.2 < r < 0.5$), while Lrzip illustrates a highly rugged landscape ($r \leq 0.2$).

> ***Finding 5:*** *Software configuration landscapes exhibit diverse ruggedness patterns. Unlike prior studies [47] that predominantly observe highly rugged landscapes, our investigation reveals that 5 out of the 9 studied systems display relatively smooth landscapes. On the other hand, Lrzip, Kanzi, X264, and Jump3r, despite spanning different areas, programming languages, and resource intensity, consistently exhibit higher degree of ruggedness.*

To understand what domain factors shape the ruggedness patterns, we examine the relations between the proportion of core options in each system and landscape characteristics. The results show that Kanzi (83%), X264 (72%), and Jump3r (54%) have a higher proportion of core options

Table 9. The autocorrelation across different workloads for nine software configurable systems. The blue cells, red cells, and gold cells indicate smooth landscapes ($r \geq 0.5$), moderately rugged landscapes ($0.2 < r < 0.5$), and highly rugged landscapes ($r \leq 0.2$), respectively.

Workload	Lrzip	Xz	Z3	Dconvert	Batik	Kanzi	X264	H2	Jump3r
W1	-0.012	0.630	0.659	0.781	0.688	0.339	0.254	0.779	0.411
W2	0.167	0.196	0.697	0.779	0.734	0.365	0.263	0.828	0.320
W3	0.017	0.653	0.604	0.441	0.796	0.329	0.258	0.770	0.400
W4	-0.034	0.491	0.729	0.772	0.825	0.316	0.244	0.129	0.420
W5	-0.007	0.626	0.605	0.754	0.736	0.316	0.211	0.807	0.442
W6	0.026	0.597	0.617	0.591	0.725	0.336	0.282	0.803	0.438
W7	0.180	0.534	0.659	0.892	0.670	0.377	0.222	0.608	
W8	-0.023	0.550	0.680	0.767	0.723	0.329	0.246	0.687	
W9	-0.024	0.581	0.741	0.557	0.694	0.440	0.316		
W10	-0.009	0.550	0.678	0.005	0.723				
W11	-0.035	0.568	0.703	0.322	0.636				
W12	0.024	0.612	0.578	0.672					
W13	0.035	0.714							
Median	**-0.007**	**0.581**	**0.669**	**0.713**	**0.723**	**0.336**	**0.254**	**0.774**	**0.415**
IQR	**0.049**	**0.076**	**0.085**	**0.246**	**0.044**	**0.037**	**0.019**	**0.137**	**0.031**

compared to Xz (50%), Z3 (33%), Dconvert (50%), Batik (13%), and H2 (20%). This exhibits a clear trend with respect to the ruggedness of configuration landscape:

> ***Finding 6:*** *Systems with a higher proportion of core options tend to exhibit greater landscape ruggedness, more deceptive structures, and a higher prevalence of local optima.*

5.3 RQ3: Spatial Workload-level Synergy

At workload level, following Domland, we focus on findings related to *all workloads of a system* and *all workloads across all systems.*

FDC values (see Table 6) and autocorrelation (see Table 9) show that some systems (e.g., Dconvert, H2) exhibit high variability, while others remain relatively stable. Similarly, the proportion of configurations in the global optimum's basin (see Table 7) remains stable across workloads for most systems, except Z3, which shows notable fluctuations. The proportion and quality of local optima (see Table 8) remain largely consistent across workloads, with minor deviations in Z3 and Dconvert.

> ***Finding 7:*** *Workload-induced landscape variations are generally stable with only a minority of systems pronounce shifts.*

To uncover the factors driving workload-induced variations, we investigate whether workload properties consistently correlate with landscape shifts. Specifically, we focus our analysis on systems where landscape ruggedness exhibits significant fluctuations across workloads, assessing whether workload type and scale consistently impact landscape characteristics. Among all systems, Dconvert, H2, and Xz display the highest inter-workload variability in autocorrelation values (see Table 9). However, Xz is excluded from further analysis because all of its workloads belong to the same type and lack comprehensive workload scale information, limiting its suitability for correlating workload characteristics (e.g., type and scale) with landscape shifts. In contrast, Dconvert and H2 contain diverse workload types and scale variations, enabling a more meaningful investigation. Therefore, we conduct a detailed analysis of Dconvert and H2, examining their workload characteristics (including type and scale) alongside their autocorrelation values.

Table 10. Autocorrelation of workloads in DConvert and H2.

DConvert		H2	
Workload (Type-Scale)	**Autocorrelation**	**Workload (Type-Scale)**	**Autocorrelation**
jpeg-large	0.781	smallbank-1	0.779
jpeg-medium	0.779	smallbank-10	0.828
jpeg-small	0.441	tpcc-2	0.770
png-large	0.772	**tpcc-8**	**0.129**
png-medium	0.754	voter-16	0.807
png-small	0.591	voter-2	0.803
psd-large	0.892	ycsb-2400	0.608
psd-medium	0.767	ycsb-600	0.687
psd-small	0.557		
svg-large	**0.005**		
svg-medium	0.322		
svg-small	0.672		

As shown in Table 10, even within the same type (e.g., `tpcc`), variations in scale cause substantial shifts in landscape structure. Likewise, workloads of the similar scale but different types also induce notable changes (e.g., comparing `png-large` to `svg-large`).

Finding 8: *Workload effects on landscape structure are not uniformly tied to type or scale. Both contribute to variations, but their impact is system-dependent, highlighting complex interactions with system characteristics.*

5.4 Actionable Insights

Findings 1 & 2 suggest that the proportion and magnitude of ruggedness-sensitive options are system-dependent, with core options (F_1) exhibiting a stronger impact on landscape ruggedness than the other types.

Insight 1: Tuner design should be system-specific, where prioritizing ruggedness-sensitive options may benefit exploitation-driven tuners, potentially leading to significant performance improvements. For system designers, selectively exposing core options helps strike a trade-off between configurability and stability.

Finding 3 suggests that the majority of the configuration landscapes exhibit clear trends toward the global optimum, providing rich guidance to a tuner.

Insight 2: Given the observed global trends in many configuration landscapes, tuners that favor convergence/exploitation (e.g., iterated local search [63]) could often work well.

Finding 4 implies that tuners are prone to being trapped in suboptimal configurations, which often exhibit a generally acceptable quality.

Insight 3: While suboptimal configurations often yield acceptable results, global exploration is still crucial for performance-critical tasks. Strategies like multi-objectivization [20, 27] and landscape smoothing [57], have shown promise in counteracting premature convergence.

Findings 5 & 6 indicate that software configuration landscapes exhibit diverse ruggedness patterns, with some systems displaying smooth structures while others, particularly those with a higher proportion of core options, exhibiting a higher degree of ruggedness.

Insight 4: The system-specific ruggedness highlights the need to analyze the landscape characteristics before configuration tuning. Smooth landscapes benefit performance modeling, making model-based tuners preferable. Conversely, rugged landscapes require strong exploration to handle abrupt performance fluctuations. For system designers, coding fewer core options in the systems to configure can greatly help to relieve the difficulty of tuning them.

Findings 7 & 8 suggest that workload-induced variations are system-specific—most exhibit stable landscape properties across workloads, while only a few show dramatic shifts. Moreover, workload effects on landscape structure do not consistently align with type or scale, indicating complex interactions with system properties.

Insight 5: The tuning strategies should be workload-aware rather than one-size-fits-all. Systems with stable landscapes may benefit from static tuning [23, 56], while those with significant shifts require self-adaptation [98].

5.5 Experimental Validation of Actionable Insights

We have systematically analyzed the configuration landscapes from multiple perspectives, including spatial option-level synergy, spatial system-level synergy, and spatial workload-level synergy. Based on these analyses, we draw several actionable insights that aim to guide tuner researchers and system designers in making more informed decisions. To verify whether the excavated insights hold in practice, we conduct a series of algorithmic experiments on configuration tuning and discuss how the results can be related to our insights above.

5.5.1 Validating Insight 1: Prioritizing Ruggedness-Sensitive Options. To examine the role of ruggedness-sensitive options, we compare tuners that either uniformly treat all configuration options or prioritize those identified as ruggedness-sensitive. In addition, to further contextualize this design choice, we construct a variant where sensitive options are identified using ANOVA. This allows us to contrast landscape-derived sensitivity with a commonly used statistical sensitivity measure. We consider two representative types of tuners with different exploration-exploitation characteristics: hill climbing (HC) and genetic algorithm (GA), each of which is evaluated in two variants, as follows:

- **Vanilla HC:** This approach treats all configuration options equally. At each iteration, an option is selected uniformly at random and mutated to explore a neighboring configuration.
- **Priority HC:** This approach gives higher priority to sensitive options during mutation. Specifically, at each iteration, there is a 90% probability that a mutation is applied to one of the sensitive options, and only a 10% chance that a non-sensitive option is selected. In our experiments, sensitive options are identified either by ruggedness analysis or by ANOVA.
- **Vanilla GA:** This approach uses uniform crossover, where each configuration option has an equal 50% probability of being exchanged between parent configurations, ensuring all options are treated equally.
- **Priority GA:** This approach biases the crossover toward sensitive options. At each crossover operation, there is a 90% probability that a sensitive option is exchanged. As with HC, sensitive options are determined either through ruggedness analysis or ANOVA.

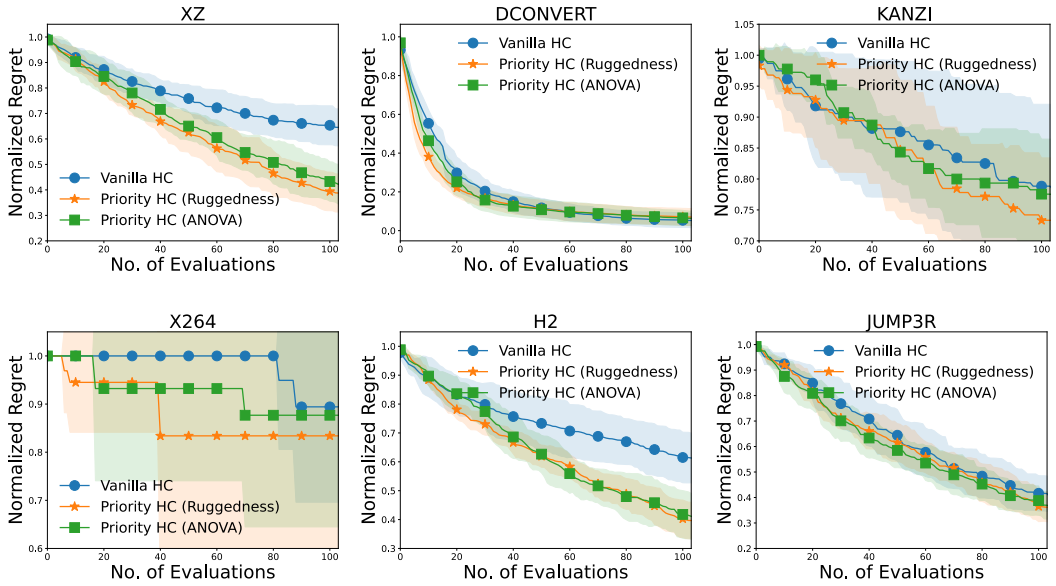


Fig. 7. The regret trajectories of Vanilla HC and Priority HC, aggregated across all workloads for each system.

In our experiment, we focus on systems that contain a mix of both ruggedness-sensitive and non-ruggedness-sensitive configuration options, i.e., Xz, Dconvert, Kanzi, X264, H2, and Jump3r. The remaining three systems were excluded because all their options either strongly or weakly affect landscape ruggedness, which makes the comparison between Vanilla HC and Priority HC (as well as Vanilla GA and Priority GA) uninformative for this validation. All experiments are repeated 30 times independently, and the tuning trajectories of the above tuners are illustrated in Figures 7 and 8, where the results are averaged over all workloads for each system to simplify the exposition. As shown in Figure 7, Priority HC achieves faster convergence and lower final regret than Vanilla HC in the majority of systems, suggesting that prioritizing ruggedness-sensitive options leads to more efficient tuning. Notably, prioritization based on ruggedness-sensitive options tends to yield larger improvements than prioritization based on ANOVA-identified options. This indicates that ruggedness analysis more accurately captures the option characteristics that influence the search behavior of HC. An exception is Dconvert, where both variants exhibit similar performance. This may be attributed to the limited number of ruggedness-sensitive options in this system—only two were identified—increasing the likelihood of premature convergence due to an overly narrow search focus. In contrast, Figure 8 presents the comparison between Vanilla GA and Priority GA. As expected, prioritizing ruggedness-sensitive options leads to nearly identical convergence trajectories across most studied systems. This suggests that tuners with strong exploration are inherently more robust to ruggedness and therefore gain limited advantage from prioritizing ruggedness-sensitive options. Surprisingly, contrary to the common intuition that prioritizing performance-sensitive options should generally improve tuning efficiency, prioritizing the options identified by ANOVA does not yield noticeable improvements. This suggests that relying solely on performance-sensitivity analysis, without considering the characteristics of the tuning algorithm, may lead to misleading guidance for search strategies. ANOVA measures how much an option contributes to overall performance variance, but it does not capture how options reshape the landscape structure or interact with the search dynamics of different tuners. Consequently, ANOVA-based prioritization

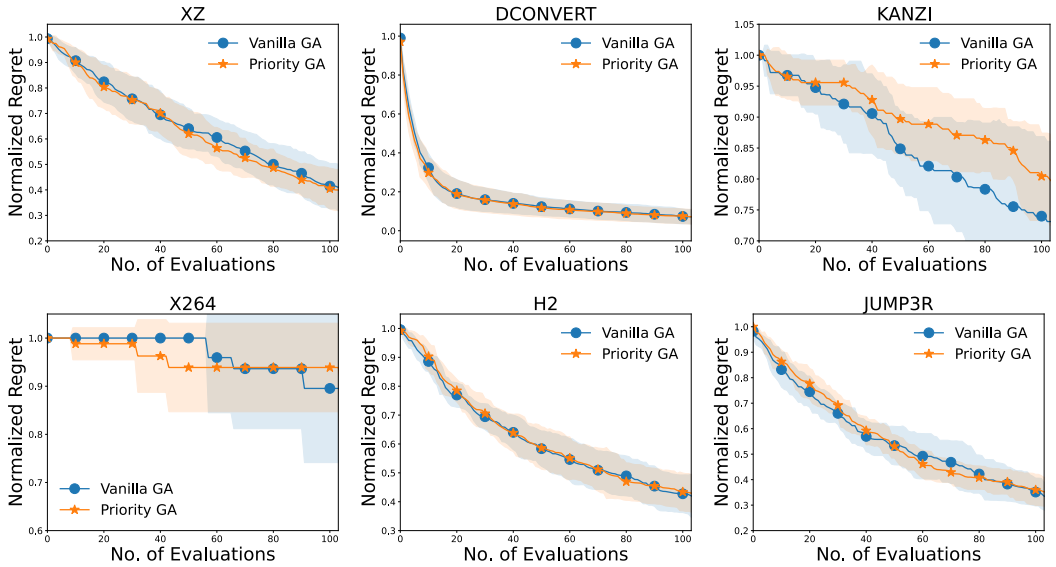


Fig. 8. The regret trajectories of Vanilla GA and Priority GA, aggregated across all workloads for each system.

fails to differentiate the behaviors of HC and GA, whereas ruggedness-based analysis provides algorithm-aware insights that better explain when option prioritization is beneficial.

Overall, these observations provide empirical evidence for ***Insight 1***, suggesting that tuner design should be system-specific. In systems where ruggedness-sensitive options are prevalent, giving them priority during the tuning process can enhance the efficiency, particularly for exploitation-driven tuners, whereas exploration-driven tuners tend to be inherently robust to such ruggedness effects and therefore benefit less from such prioritization.

5.5.2 Validating Insights 2 to 4: Tuner Performance across Landscape Characteristics. Insights 2, 3, and 4 are derived from a detailed analysis of configuration landscapes from three aspects, including global structure, local structure, and ruggedness. These insights suggest that the performance of tuners may depend on the underlying characteristics of the configuration landscape. For instance, model-based tuners are expected to perform well on landscapes with smooth global structures, where accurate performance modeling is feasible. In contrast, landscapes that are rugged or highly multimodal may favor exploration-driven tuners that can better escape local optima. To evaluate whether these actionable insights hold in practice, we conduct a comparative experiment involving four tuners (i.e., GA, RS, HC, and TPE) with diverse properties across nine representative systems.

Specifically, the characteristics and implementation details of the four tuners are summarized as follows:

- **Genetic Algorithm (GA) [16, 74]:** GA is a population-based tuner that explores configuration space via selection, crossover, and mutation, equipped with strong exploration capability. Specifically, we use binary tournament selection for mating, a population size of 20, and apply uniform crossover with a probability of 0.9 and boundary mutation with a probability of 0.1, respectively.
- **Random Search (RS) [73]:** RS is a pure exploration tuner that samples configurations randomly from the configuration space.

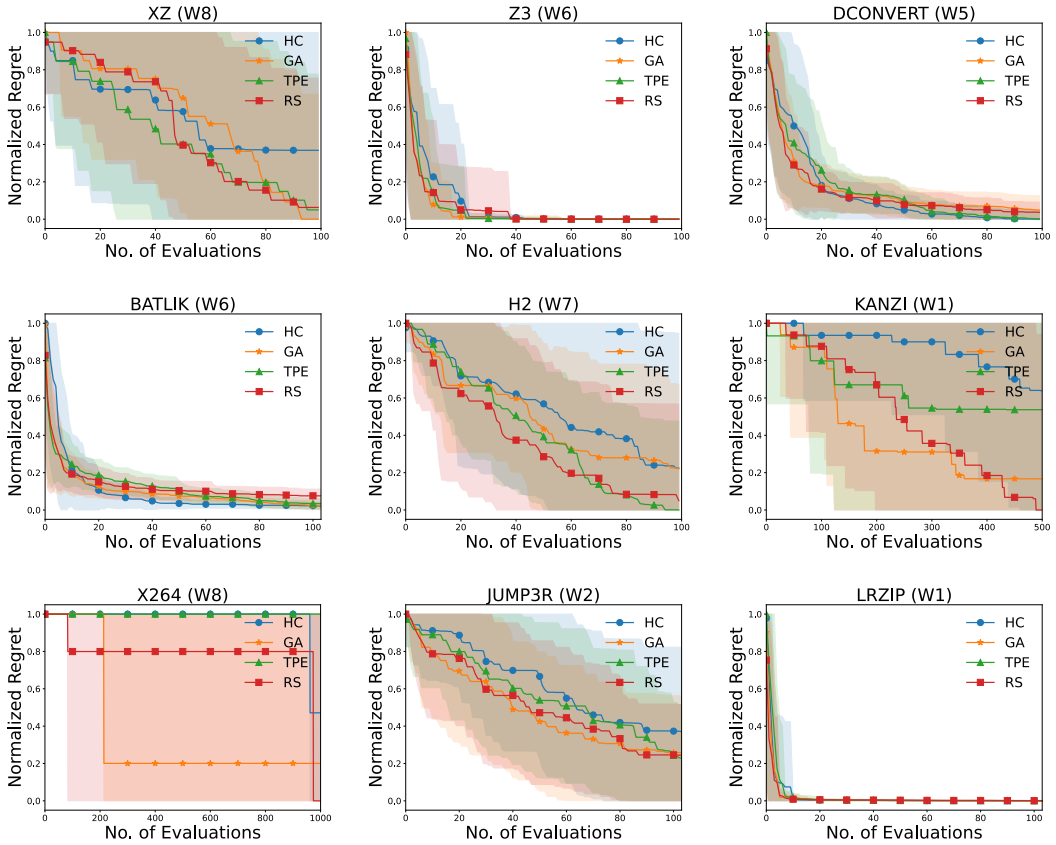


Fig. 9. The regret trajectories using GA, RS, HC, and TPE to tune nine systems.

- **Hill Climbing (HC) [60, 95]:** HC is a deterministic local search tuner that iteratively explores the neighborhood of the current configuration and moves to an improved one when available. In our implementation, we adopt the first-improvement strategy, where neighbors are evaluated at random and the search accepts the first configuration that yields better performance.
- **TPE [17]:** TPE is a model-based tuner that uses density estimation to model good and bad configurations separately and selects new configurations by maximizing expected improvement.

Accordingly, we summarize the landscape characteristics of the nine studied systems into three representative types as follows:

- **Type I (Xz, Z3, Dconvert, Batik, and H2)**: These systems often exhibit relatively smooth landscapes ($r \geq 0.5$) with a small number of local optima, most of which yield high-quality performance.
- **Type II (Kanzi, X264, and Jump3r)**: These system present moderately rugged landscapes ($0.2 < r < 0.5$), containing a certain number of local optima with medium quality.

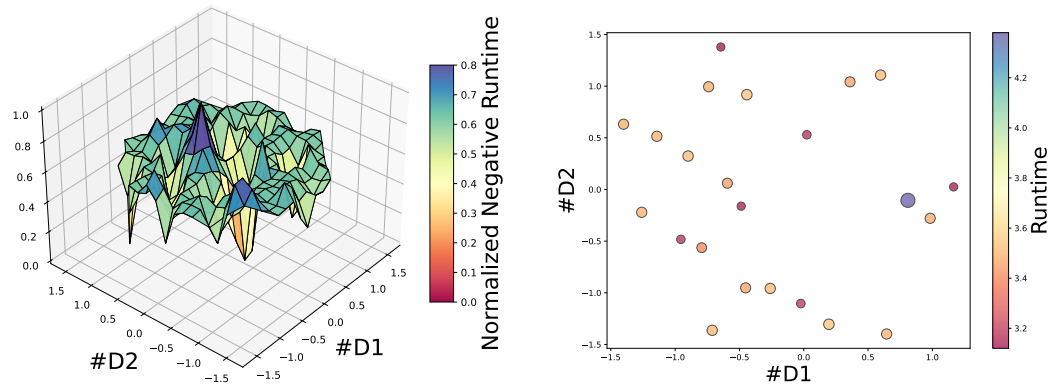
- **Type III (LRZIP)**: This system features a highly rugged landscape ($r \leq 0.2$) characterized by numerous local optima, many of which are of high-quality and close to the global optimum (ℓ_p=10.5% and ℓ_q=0.959).

In this experiment, we select one representative workload for each system, chosen to closely reflect the system's overall landscape characteristics. Each tuner is independently executed 30 times to mitigate stochastic variance, and the aggregated regret trajectories are illustrated in Figure 9. The results reveal several notable patterns, which we outline as follows:

1) **Type I**: These systems exhibit smooth landscapes with a clear global structure, HC converges slowly but steadily to high-quality configurations. Meanwhile, the model-based tuner TPE performs comparably well, probably due to the reliability of surrogate modeling in such a smooth landscape. These results align with ***Insights 2 & 4***, which suggest that exploitation-driven and model-based methods could be effective when the landscape exhibits clear global guidance and smooth structure.
2) **Type II**: These systems are characterized by moderate ruggedness and multimodality; both HC and TPE exhibit degraded performance. For HC, the relatively rugged landscape with abundant local optima may cause the search to get trapped in suboptimal regions and struggle to escape. For TPE, the performance drop may stem from the difficulty of learning an accurate surrogate model under such an irregular and multimodal landscape structure. In contrast, RS and GA, which offer stronger global exploration capabilities, achieve superior performance. These observations lend support to ***Insights 3 & 4***, underscoring the importance of exploration in rugged and multimodal landscapes.
3) **Type III**: On workload W1 of LRZIP, which is characterized by a highly rugged landscape with numerous high-quality local optima, all tuners, including those with limited exploration capabilities, unexpectedly achieve good performance and rapid convergence. This counterintuitive result can be attributed to the dense distribution of high-quality local optima (ℓ_p=10.5% and ℓ_q=0.959), which allows search processes to quickly settle into high-performing regions (***Insights 3 & 4***). To further investigate this seemingly counterintuitive phenomenon, we visualize the configuration landscape and distribution of local optima for LRZIP, as shown in Figure 10. From Figure 10(a), it can be observed that while a few configurations exhibit notably poor performance (corresponding to deep valleys), the majority of configurations achieve similar and relatively high performance, resulting in a flat upper region where the performance differences among good configurations are marginal. Furthermore, Figure 10(b) shows that the identified local optima are densely distributed (ℓ_p=10.5%) and exhibit small performance variations, which enables different tuners, even those with limited exploration, to quickly converge to high-performing regions. Moreover, the configuration space of LRZIP involves only seven configuration options, further reducing the search difficulty and explaining why all tuners achieve similar performance despite the high ruggedness.

Overall, the above analyses suggest that tuner performance often relies on the underlying landscape characteristics of the optimization problem. By systematically analyzing multiple aspects of the configuration landscape, such as ruggedness, local structure, and global structure, practitioners can better assess the problem's inherent difficulty and interpret the behavior of different tuning strategies. Such understanding not only facilitates informed tuner selection and design but also enhances the explainability of why a tuner succeeds or fails under specific scenarios.

5.5.3 Validating Insight 5: Transferability Across Workloads. To investigate whether configuration transfer across workloads can improve tuning performance, we conduct an experiment under dynamic workload settings, comparing tuning with and without configuration transfer. That is, we



(a) Landscape (b) Local Optima

Fig. 10. Illustration of the configuration landscape and local optima for workload W1 of Lrzip. (a) Normalized negative runtime landscape (higher values indicate better performance). (b) Spatial distribution of identified local optima and their corresponding performance values.

examine whether configuration transfer is beneficial under similar configuration landscapes, and conversely, whether it loses effectiveness—thus calling for self-adaptation strategies [98]—when landscape structures vary significantly across workloads, as suggested by ***Insight 5***. Specifically, we compare the following two variants of the genetic algorithm (GA) [16, 74]:

- **Restarted GA:** This approach responds to workload changes by triggering a new search from scratch with randomly initialized configurations.
- **Transfer GA:** This approach responds to workload changes by transferring all the configurations optimized from the most recent past workload as the initial population. For example, given a random sequence of workloads (e.g., W1 → W3 → W4 → W2 → W6 → W5), when tuning starts on W3, the configurations optimized under W1 are used as initialization; for W4, the ones from W3 are transferred, and so on.

In our experiment, each run begins with a randomly shuffled order of workloads, mimicking a dynamic operational environment. For each workload arrival, the tuner is triggered to search for an optimal configuration under a fixed budget of 80 measurements, a setting aligned with the prior study on configuration tuning [98]. To obtain a statistically sound comparison, all experiments are run 30 times independently. Then, we use the Wilcoxon rank-sum test (p) [15] with 0.05 significance level and $\hat{A}_{12}$ effect size [85] to assess statistical differences between the above tuners. We say a comparison is statistically significant only if it has $\hat{A}_{12} > 0.56$ (or $\hat{A}_{12} < 0.44$) and $p < 0.05$.

The experimental results are summarized in Table 11. Clearly, Transfer GA significantly outperforms Restarted GA, achieving better performance in 63 out of 93 cases. This advantage may stem from the fact that, in most systems, the configuration landscapes across different workloads exhibit structural similarity. As a result, directly reusing configurations optimized under the previous workload can provide a strong starting point for tuning under the current workload, reducing search overhead and accelerating convergence. However, on some systems such as Z3 and H2, the effectiveness of Transfer GA diminishes, yielding no clear advantage over Restarted GA, and in some cases even leading to negative transfer. This may be attributed to the relatively greater variability in landscape structure across workloads observed in these systems, as discussed in Section 5.3. Overall, this comparative experiment provides empirical support for ***Insight 5***, confirming that

Table 11. The Mean and Standard deviation (Std) of performance objectives between Transfer GA and Restarted GA for all cases over 30 runs; The blue cells, red cells, and gold cells respectively indicate Transfer GA performing significantly better than, similarly to, or worse than Restarted GA.

Workload	Tuner	Lrzip	Xz	Z3	Dconvert	Batik	Kanzi	X264	H2	Jump3r
		Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)
W1	Transfer GA	3.131 (0.014)	3.773 (0.955)	5.860 (0.016)	1.797 (0.009)	0.903 (0.009)	1.129 (1.529)	0.929 (0.258)	26336.284 (1348.381)	2.376 (0.478)
	Restarted GA	3.133 (0.017)	4.732 (1.299)	5.907 (0.226)	1.860 (0.092)	0.924 (0.019)	1.832 (1.368)	1.097 (0.348)	25721.608 (816.896)	2.768 (0.658)
W2	Transfer GA	0.030 (0.000)	0.011 (0.003)	2.333 (0.656)	1.078 (0.040)	1.334 (0.006)	0.133 (0.048)	3.778 (1.110)	18266.502 (1838.219)	0.753 (0.131)
	Restarted GA	0.030 (0.000)	0.014 (0.005)	1.761 (0.004)	1.110 (0.040)	1.360 (0.021)	0.160 (0.044)	4.005 (1.024)	18378.785 (822.054)	0.934 (0.272)
W3	Transfer GA	3.301 (0.003)	4.014 (1.188)	0.402 (0.695)	0.371 (0.003)	4.177 (0.020)	0.427 (0.311)	1.322 (0.300)	932.575 (51.553)	1.270 (0.357)
	Restarted GA	3.314 (0.024)	4.857 (1.347)	0.573 (0.905)	0.377 (0.006)	4.254 (0.089)	0.281 (0.066)	1.449 (0.375)	878.662 (63.803)	1.548 (0.487)
W4	Transfer GA	7.170 (0.022)	11.096 (3.137)	2.428 (0.247)	1.561 (0.003)	1.186 (0.020)	2.020 (4.084)	1.603 (0.412)	984.067 (141.645)	0.619 (0.063)
	Restarted GA	7.158 (0.032)	14.333 (4.261)	2.376 (0.332)	1.603 (0.048)	1.214 (0.019)	2.207 (1.423)	1.886 (0.617)	934.248 (131.108)	0.698 (0.121)
W5	Transfer GA	33.431 (0.195)	11.596 (3.487)	3.378 (0.876)	0.489 (0.005)	2.393 (0.008)	1.116 (1.692)	3.414 (0.834)	46953.445 (4509.310)	1.103 (0.282)
	Restarted GA	33.443 (0.194)	12.438 (3.627)	3.299 (0.275)	0.504 (0.014)	2.460 (0.061)	1.539 (0.982)	3.561 (0.774)	47123.373 (2904.705)	1.363 (0.503)
W6	Transfer GA	0.970 (0.000)	1.610 (0.458)	1.324 (0.131)	0.365 (0.008)	3.145 (0.042)	0.446 (0.396)	0.102 (0.014)	47071.542 (4573.392)	0.297 (0.021)
	Restarted GA	0.972 (0.005)	1.959 (0.585)	1.370 (0.190)	0.382 (0.009)	3.242 (0.076)	0.548 (0.328)	0.110 (0.022)	47652.393 (822.434)	0.319 (0.036)
W7	Transfer GA	0.191 (0.003)	0.199 (0.014)	0.319 (0.401)	16.148 (0.022)	1.135 (0.014)	0.173 (0.101)	0.595 (0.150)	19787.726 (1770.222)	
	Restarted GA	0.195 (0.005)	0.206 (0.017)	0.343 (0.306)	16.475 (1.411)	1.149 (0.016)	0.241 (0.131)	0.651 (0.134)	18676.683 (2001.725)	
W8	Transfer GA	10.902 (0.009)	23.830 (6.797)	8.747 (0.005)	1.013 (0.007)	7.065 (0.041)	3.448 (7.086)	0.141 (0.033)	27809.413 (1992.050)	
	Restarted GA	10.912 (0.023)	30.210 (8.771)	8.748 (0.004)	1.034 (0.025)	7.105 (0.062)	4.937 (4.133)	0.170 (0.043)	25719.406 (2141.486)	
W9	Transfer GA	9.127 (0.316)	21.255 (5.355)	3.182 (0.004)	0.462 (0.013)	1.046 (0.009)	0.780 (1.248)	0.251 (0.041)		
	Restarted GA	9.192 (0.413)	22.277 (6.795)	3.185 (0.005)	0.477 (0.016)	1.062 (0.017)	1.205 (0.951)	0.263 (0.053)		
W10	Transfer GA	5.291 (0.035)	10.306 (2.266)	6.838 (0.246)	1.435 (0.008)	1.112 (0.006)				
	Restarted GA	5.407 (0.270)	13.123 (4.756)	6.803 (0.251)	1.437 (0.008)	1.127 (0.019)				
W11	Transfer GA	2.081 (0.003)	2.825 (0.687)	8.240 (0.870)	1.429 (0.014)	1.619 (0.025)				
	Restarted GA	2.095 (0.028)	3.202 (0.880)	8.121 (0.730)	1.448 (0.020)	1.668 (0.035)				
W12	Transfer GA	3.483 (0.076)	5.653 (1.647)	3.899 (0.088)	0.480 (0.004)					
	Restarted GA	3.510 (0.097)	7.278 (1.908)	3.936 (0.262)	0.481 (0.007)					
W13	Transfer GA	2.525 (0.013)	2.947 (0.822)							
	Restarted GA	2.534 (0.020)	3.432 (0.927)							
Summary		8	9	2	10	11	8	4	5	6
		5	4	8	2	0	0	5	0	0
		0	0	2	0	0	1	0	3	0

transferability of configurations is generally beneficial when landscape structures remain stable across workloads, while its effectiveness diminishes as landscape variability increases, requiring more sophisticated self-adaptation approaches to handle such challenges [98].

6 Discussion

The results of our study demonstrate that tuning effectiveness is closely tied to the structural properties of the configuration landscape, which are shaped by the system's domain characteristics. These observations reinforce the importance of understanding not only what a tuner does, but why it succeeds or fails under specific tuning tasks.

Our proposed methodology, `Domland`, offers a systematic way to close this gap. By synergizing domain knowledge and spatial information of the configuration landscape mined via FLA, `Domland` enables researchers and practitioners to reason about tuning difficulty, explain tuner behavior, and inform the design of more robust tuning strategies. Therefore, our methodology has important implications for both tuner researchers and system designers, together with several relevant practices in software engineering, which we outline below.

6.1 Implications to Tuner Researchers

`Domland` provides actionable insights for tuner researchers, guiding decisions across three key aspects:

(1) **Tuner selection:** `Domland` offers a comprehensive analysis on configuration landscape (e.g., ruggedness, local structure, global structure), shedding light on tuner selection. For instance, smooth landscapes with high FDC (***Finding 3***) favor exploitation-driven methods such as

iterated local search [63], whereas highly rugged or with rich local optima structures (***Finding 5***) require global exploration strategies (e.g., multi-objectivization [20, 27]) to escape local optima. In practical optimization scenarios, this process can be carried out as follows:
- **Step 1:** Collect a small set of configuration–performance measurements, for example, from lightweight sampling or historical observations.
- **Step 2:** Apply analysis workflow described in Section 3 to characterize key structural properties of the landscape (e.g., ruggedness, local structure, and global structure).
- **Step 3:** Use the resulting structural patterns to guide tuner selection/design. For instance, landscapes exhibiting high ruggedness may call for stronger exploration, whereas smoother landscapes may favor exploitation-oriented or model-based approaches.

(2) **Operator design:** Beyond tuner selection, `Domland` can also guide operator design. For example, in landscape with structured basins of attraction and weak FDC correlation (e.g., Batik in our case study), integrating global exploration with local search operators (e.g., variable neighborhood search [69]) might improve search efficiency while mitigating premature convergence.

(3) **Budget allocation:** Researchers can also utilize `Domland` to assess the hardness of the tuning tasks and inform budget allocations. For instance, deceptive landscapes with numerous local optima (***Finding 4***) may demand increased evaluation budgets for global exploration, whereas smoother landscape allow faster convergence with fewer evaluations (***Finding 5***). By aligning tuner effort with system complexity, `Domland` helps the designs of smarter budget allocation.

6.2 Implications to System Designers

For system designers, `Domland`, for the first time, reveals how the options and the workloads contribute to the characteristics of tuning configurable systems, thereby making the (re-)designs thereof with more informed decisions:

(1) `Domland` helps to understand how domain-specific factors (e.g., option types, system attributes, and workload characteristics) shape the configuration landscape structures. For example, by synergizing the spatiality and domain analysis in `Domland`, we found that core options exert stronger impact on landscape ruggedness (***Findings 2 and 6***).

(2) `Domland` can help enhance configuration manuals by refining option descriptions. For example, highlighting the option's impacts on ruggedness and implication to tuner design in API documentation (***Findings 1 and 2***).

(3) `Domland` assists system designers in refining system configurability, such as selectively refactoring tunable options based on their impact on ruggedness (***Findings 2 and 6***).

In practical system (re-)design, practitioners can follow the procedures below:

- **Step 1:** Apply the workflow described in Section 3 to analyze the domain-spatiality patterns of the case and identify options (or option types) that are strongly associated with structural difficulty (e.g., sensitive to landscape ruggedness).
- **Step 2:** Based on these insights, implement system-level design adjustments that constrain or regulate such options, for instance, introducing new code that bounds their permissible value ranges or reduces harmful interactions.
- **Step 3:** Directly apply a tuner with predefined ranges/bounds. By limiting configurations that induce highly rugged landscapes, the resulting configuration space becomes smoother and more amenable to efficient tuning (e.g., by local search tuners).

6.3 Implications to Practitioners

In addition to the above, the domain-spatiality patterns revealed by Domland also offers a broader foundation for understanding and explaining diverse aspects of "difficulty" for tuning configurable systems. This would benefit other relevant practices of software engineering, such as exploratory analysis and performance analysis:

(1) **Exploratory analysis for understanding tuning cases:** Domland can be used for exploratory analysis to understand the structural characteristics of a tuning case and to help explain why a tuner behaves as observed (e.g., fast convergence, stagnation, or high variance). By analyzing the domain–spatiality patterns, practitioners can obtain a structured view of the tuning difficulty and the likely behaviors of different tuners. For example, when the goal is to explain why different tuners exhibit distinct search behaviors on a given configurable system, the following analysis can be carried out:
   - **Step 1:** Apply the workflow described in Section 3 to reveal the domain–spatiality patterns of the case, including structural properties such as ruggedness, local structure, and global structure.
   - **Step 2:** Summarize the core search mechanism of the tuners under comparison, such as their balance between exploration and exploitation, use of restart or diversity maintenance, or dependence on surrogate modeling.
   - **Step 3:** Interpret the observed behavioral differences by aligning the structural properties of the tuning case with the search characteristics of the tuners. For instance, highly rugged and multi-modal landscapes may hinder strongly exploitative strategies, whereas smoother and more coherent landscapes may favor model-based approaches.

(2) **Assisted performance analysis prioritization:** Domland helps to reveal the workloads (or options), including their types, that should be studied first, in order to analyze the system. In practical performance analysis scenarios, practitioners can follow the procedures below:
   - **Step 1:** Apply the workflow described in Section 3 to characterize the spatiality patterns of the target system across workloads or option settings, with particular attention to indicators such as autocorrelation and FDC.
   - **Step 2:** Identify workloads or options that are associated with structurally difficult landscapes, such those exhibiting low autocorrelation or weak FDC, since these patterns indicate irregular search spaces and potentially unstable performance behavior.
   - **Step 3:** Rank those workloads/options according to the landscape metrics. For example, testing configurations with prioritized workloads or options, as they are more likely to expose performance anomalies, hidden sensitivities, or bugs. This helps to reveal configuration-related performance bugs at earlier stage in configuration bug testing [65].

7 Threats to Validity

Threats to **internal validity** is related to the definition of neighborhood structures in Domland, since we do not have a complete dataset of all configurations. To mitigate this, we adopt an adaptive neighborhood strategy, attempting to ensure connectivity while preserving landscape integrity. However, this remains an approximation that may impact the characterization of the landscape. Besides, the exact landscape characteristics might vary depending on specific FLA metrics used in Domland, which, although widely adopted [47, 53, 66, 92], may not capture all structural aspects of complex configuration spaces. For example, alternative metrics may be more suitable for quantifying landscape ruggedness if the software system satisfies certain implicit assumptions required by those measures [36, 55]. The inclusion of alternative or complementary metrics could provide a more

comprehensive understanding of the spatial characteristics of a software system. Another potential threat arises from our use of the best-performing configuration within the sampled data as a proxy for the global optimum in computing landscape metrics. While this practice follows the setting in existing literature [48, 82], it may introduce bias if the sampled configurations do not sufficiently approximate the true global optimum. Nevertheless, the datasets we used were collected through diverse coverage-based and random sampling strategies, which help reduce this risk to a certain extent. A further limitation lies in our focus on option-specific ruggedness where each configuration option is analyzed independently. This design choice ensures interpretability and consistency with the option-level perspective adopted in earlier analyses. Nevertheless, if resources permitted, the methodology can be readily extended to capture interaction-level ruggedness by partitioning the configuration space jointly over multiple options (e.g., pairs or triplets) and computing the corresponding autocorrelation metric. Such an extension would enable systematic investigation of how option interactions, for example, between core and resource-related options, jointly reshape the configuration landscape. We consider this as an interesting direction for future work.

Threats to **external validity** may arise from dataset selection. While our case study spans nine configurable systems and 93 workloads, a more exhaustive exploration of the configuration space—potentially covering a complete set of configurations—could further strengthen the findings. Our case study used sampled datasets rather than exhaustive configuration enumeration, potential biases may arise from incomplete coverage of the configuration space. Sampling-based datasets, while practical for large-scale configurable systems, may omit certain configurations or interactions, potentially distorting the true configuration landscape. Nonetheless, the datasets we use were collected using coverage-based and random sampling strategies, which aims to approximate the landscape's overall structure with minimal bias. This trade-off between feasibility and completeness is consistent with realistic tuning scenarios, where exhaustive evaluation is infeasible. To the best of our knowledge, the adopted dataset is one of the largest publicly available configuration–performance datasets supporting multi-system and multi-workload landscape analysis. While no dataset can be fully representative of all modern configurable systems, the breadth of this dataset provides a strong empirical foundation for the domain–spatiality analyses conducted in this study. In addition, while our analysis reveals some common patterns across systems or workloads, different types of configurable software systems may exhibit fundamentally distinct landscape features. Therefore, the findings reported here might not directly or fully generalize to all configurable systems. Nonetheless, the methodology is designed to be general-purpose and can be applied to a wide range of black-box configurable systems. We believe the FLA perspective presented in this work offers a systematic lens through which diverse configuration-performance spatial correlations can be analyzed, and it can be extended to support system-specific insights in other application domains.

On domain analysis in `Domland`, the human analysis might pose threats to **construct validity**. To mitigate this, we followed a systematic classification protocol, involving multiple rounds of independent reviews. Yet, unintentional subjective biases are still possible. Future work could integrate automated program analysis techniques to enhance consistency and reduce potential biases.

8 Related Work

Static Domain Analysis: Over the past decade, there exist methodologies that seek to pry open the black box of systems for understanding their internal characteristics [61]. These, typically, leverage *domain-specific knowledge* (e.g., system manual) [45, 58, 81] or *code-level analysis* (e.g., static code inspection) to excavate how parameters influence system behaviors [30]. For instance, Li et al. [59] propose a static analysis approach to infer the performance properties of configuration

options by analyzing control and data dependencies between option values and performance-critical operations. Chen et al. [30] propose DiagConfig, a configuration-oriented diagnosis method that explains performance violations by tracing how configuration options propagate through code. DiagConfig combines static code analysis with performance violation detection to identify performance-sensitive options and construct cause-effect chains linking misconfigurations to observed performance issues. These analyses can help identify and prioritize critical options; reveal causal relationships between configuration options and system performance; and even extract implicit constraints [22]. Nevertheless, existing domain analysis methods heavily rely on expert assumptions, which alone cannot fully reveal when a tuner might be successful or fail to tune a system. In contrast, our methodology, `Domland` , synergizes the spatial information of the configuration space, mined using FLA, with domain knowledge, allowing us to better profile the characteristics of the systems and understand the behaviors of the tuners.

Dynamic Data Analysis: A common data-driven method to study configurable systems is to apply statistical methods to extract the characteristics of performance distribution [51, 70]. For example, Valov et al. [84] analyze performance distribution across platforms and find that they exhibit stable trends, enabling low-cost performance prediction via model transfer. Jamshidi et al. [51] study how performance distributions drift across environments and leverage this knowledge to guide transfer learning for performance modeling. Xiang et al. [96] model online configuration performance drifts. While statistical methods can mine useful information regarding the systems, such as trends and skewness, they typically consider performance values as isolated numerical observations, neglecting the inherent spatial information within the configuration space. In fact, each performance value is intrinsically tied to a specific configuration, and configurations themselves exhibit spatial relationships defined by their parameter values. As a result, the associated performance values are naturally distributed across the configuration space, forming spatially structured patterns rather than isolated data points. The lack of spatial information makes it challenging to answer fundamental questions such as: *"where is the global optimum located?", "what does the local structure look like?" and "how rugged is the search space?"*—the key factors that shape the tuner's behaviors. Recently, FLA, which explicitly incorporates spatial information, has been applied in software testing [13, 35, 72], configuration tuning [21, 47]. The most relevant work is perhaps `GraphFLA` [47], a graph-based method to extract spatial relations within configuration space, aiming to uncover key system characteristics. Compared with `GraphFLA`, `Domland` of this study differs in several key aspects:

- While `GraphFLA` analyzes configuration landscape purely from a data-driven spatial perspective, `Domland` extends this by incorporating a structured system-specific domain analysis at three levels of abstractions.
- Lacking domain-specific analysis, `GraphFLA` is unable to provide a fundamental explanation for spatial insights, while `Domland` synergizes configuration spatiality and domain analysis, aligning landscape with system-specific attributes. This synergy enables a more context-aware explanation of how configurations influence performance across systems.
- Our case study with `Domland` involves a more comprehensive empirical evaluation across nine software systems and 93 workloads, covering a wider range of configurable systems and workloads compared to the more focused case studies in `GraphFLA`, which examines only three systems.

Empirical Studies of Configuration: There exist a few empirical studies for configurable systems [21, 28, 29, 70, 91, 101]. Among others, Zhang et al. [101] present a source-code level study on how configuration design and implementation evolve in cloud systems. By analyzing over a thousand configuration-related commits, they summarize developer practices and call for new

techniques to proactively reduce misconfigurations. Chen et al. [21] conduct a large-scale empirical study to examine whether higher surrogate model accuracy truly leads to better tuning results. Their findings challenge the common assumption that model accuracy determines tuning quality and call for rethinking the role of accuracy in model-based configuration tuning. Muhlbauer et al. [70] systematically study how workload variation impacts the accuracy and generalizability of performance models. Their findings show that workload variation can lead to significant and non-monotonic performance shifts, revealing the limitations of single-workload modeling and the necessity to account for workload-configuration interactions. Weber et al. [91] investigate how runtime performance correlates with energy consumption across configurable software systems and observe that the relationship is highly system-dependent. However, these studies have not emphasized a systematic methodology that connects spatial information to domain understanding of the configurable systems and their tuning.

9 Conclusion

In this paper, we propose `Domland`, a methodology that bridges spatial fitness landscape analysis with domain knowledge to understand configuration tuning. `Domland` enables a structured interpretation of configuration-performance correlation, helping to assess tuning difficulty and inform algorithm design. Through extensive experiments across nine configurable systems and 93 workloads, we show how to follow `Domland` and reveal several key insights:

- **System-specific landscapes:** Configuration landscape vary across systems; no single domain (e.g., area, language, or resource intensity) consistently shapes their structure.
- **Impact of core option:** Core options have a stronger influence on landscape ruggedness than resource options.
- **Workload-induced variations:** Workload effects on landscape structure are not uniformly tied to type or scale, but are more system-dependent.

We hope that domain-spatiality synergies for understanding configuration tuning in this work pave the way for better adaptive tuner designs and the evolution of configurable systems. By integrating `Domland` into the tuner selection and design process, future work can move toward landscape-aware autotuning, where optimization strategies are dynamically matched to system and workload characteristics for better performance and adaptability. Additionally, integrating causal reasoning into `Domland` may represent a promising future direction to move beyond correlation analysis and toward a deeper understanding of how domain factors shape configuration landscapes.

Acknowledgments

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

References

[1] 2023. The Audio Encoder: JUMP3R. https://sourceforge.net/projects/jump3r/. Accessed: December 14, 2023.
[2] 2023. The Database: H2. https://github.com/h2database/h2database. Accessed: December 14, 2023.
[3] 2023. The Density Image Converter: Dconvert. https://github.com/patrickfav/density-converter. Accessed: December 14, 2023.
[4] 2023. Experimental Guideline. https://zenodo.org/records/7504284. Accessed: December 14, 2023.
[5] 2023. The File Compressor: Kanzi. https://github.com/flanglet/kanzi. Accessed: December 14, 2023.
[6] 2023. The File Compressor: Lrzip. https://github.com/ckolivas/lrzip. Accessed: December 14, 2023.
[7] 2023. The File Compressor: Xz. https://github.com/xz-mirror/xz. Accessed: December 14, 2023.
[8] 2023. The SMT Solver: Z3. https://github.com/Z3Prover/z3. Accessed: December 14, 2023.
[9] 2023. The SVG Rasterizer: Batlik. https://github.com/apache/xmlgraphics-batik. Accessed: December 14, 2023.
[10] 2023. The Video Encoder: X264. https://github.com/mirror/x264. Accessed: December 14, 2023.
[11] 2025. https://javaparser.org/. Accessed: March 01, 2025.

[12] 2025. https://clang.llvm.org/docs/LibASTMatchers.html. Accessed: March 01, 2025.
[13] Aldeida Aleti. 2021. On the Effectiveness of SBSE Techniques: Through Instance Space Analysis. In *International Symposium on Search Based Software Engineering*. Springer, 3–6.
[14] Lee Altenberg et al. 1997. Fitness Distance Correlation Analysis: An Instructive Counterexample.. In *Icga*. 57–64.
[15] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd international conference on software engineering*. 1–10.
[16] Thomas Bäck and Hans-Paul Schwefel. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation* 1, 1 (1993), 1–23.
[17] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger (Eds.). 2546–2554. https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html
[18] Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. 2018. "Sampling" as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering* 45, 6 (2018), 597–614.
[19] Pengzhou Chen and Tao Chen. 2026. PromiseTune: Unveiling Causally Promising and Explainable Configuration Tuning. In *48th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM.
[20] Pengzhou Chen, Tao Chen, and Miqing Li. 2024. MMO: Meta Multi-Objectivization for Software Configuration Tuning. *IEEE Trans. Software Eng.* 50, 6 (2024), 1478–1504. https://doi.org/10.1109/TSE.2024.3388910
[21] Pengzhou Chen, Jingzhi Gong, and Tao Chen. 2025. Accuracy Can Lie: On the Impact of Surrogate Model in Configuration Tuning. *IEEE Trans. Software Eng.* 51, 2 (2025), 548–580. https://doi.org/10.1109/TSE.2025.3525955
[22] Qingrong Chen, Teng Wang, Owolabi Legunsen, Shanshan Li, and Tianyin Xu. 2020. Understanding and discovering software configuration dependencies in cloud and datacenter systems. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 362–374.
[23] Tao Chen. 2022. Lifelong dynamic optimization for self-adaptive systems: Fact or fiction?. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 78–89.
[24] Tao Chen. 2022. Planning landscape analysis for self-adaptive systems. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 84–90.
[25] Tao Chen and Rami Bahsoon. 2015. Toward a smarter cloud: Self-aware autoscaling of cloud configurations and resources. *Computer* 48, 9 (2015), 93–96.
[26] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–40.
[27] Tao Chen and Miqing Li. 2021. Multi-objectivizing software configuration tuning. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 453–465.
[28] Tao Chen and Miqing Li. 2023. Do Performance Aspirations Matter for Guiding Software Configuration Tuning? An Empirical Investigation under Dual Performance Objectives. *ACM Trans. Softw. Eng. Methodol.* 32, 3 (2023), 68:1–68:41. https://doi.org/10.1145/3571853
[29] Tao Chen and Miqing Li. 2023. The Weights Can Be Harmful: Pareto Search versus Weighted Search in Multi-objective Search-based Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 32, 1 (2023), 5:1–5:40. https://doi.org/10.1145/3514233
[30] Zhiming Chen, Pengfei Chen, Peipei Wang, Guangba Yu, Zilong He, and Genting Mai. 2023. Diagconfig: Configuration diagnosis of performance violations in configurable software systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 566–578.
[31] Jithin Cheriyan, Bastin Tony Roy Savarimuthu, and Stephen Cranefield. 2021. Towards offensive language detection and reduction in four software engineering communities. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. 254–259.
[32] Manuel Clergue, Philippe Collard, Marco Tomassini, and Leonardo Vanneschi. 2002. Fitness distance correlation and problem difficulty for genetic programming. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. 724–732.
[33] John B Conway. 1994. *A course in functional analysis*. Vol. 96. Springer Science & Business Media.
[34] Michael AA Cox and Trevor F Cox. 2008. Multidimensional scaling. In *Handbook of data visualization*. Springer, 315–347.
[35] Victor Crespo-Rodriguez, Neelofar, Aldeida Aleti, and Burak Turhan. 2025. Instance Space Analysis of Testing of Autonomous Vehicles in Critical Scenarios. *ACM Trans. Softw. Eng. Methodol.* 34, 3, Article 61 (Feb. 2025), 36 pages. https://doi.org/10.1145/3699596

[36] J Arjan GM De Visser and Joachim Krug. 2014. Empirical fitness landscapes and the predictability of evolution. *Nature Reviews Genetics* 15, 7 (2014), 480–490.
[37] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. Oltp-bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment* 7, 4 (2013), 277–288.
[38] Chengwen Du and Tao Chen. 2024. Contexts Matter: An Empirical Study on Contextual Influence in Fairness Testing for Deep Learning Systems. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 107–118.
[39] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.
[40] Khaled El Emam. 1999. Benchmarking Kappa: Interrater agreement in software process assessments. *Empirical Software Engineering* 4, 2 (1999), 113–133.
[41] Ayat Fekry, Lucian Carata, Thomas Pasquier, Andrew Rice, and Andy Hopper. 2020. To tune or not to tune? in search of optimal configurations for data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2494–2504.
[42] Jingzhi Gong and Tao Chen. 2024. Deep configuration performance learning: A systematic survey and taxonomy. *ACM Transactions on Software Engineering and Methodology* 34, 1 (2024), 1–62.
[43] Xue Han and Tingting Yu. 2016. An empirical study on performance bugs for highly configurable software systems. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–10.
[44] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 1–61.
[45] Haochen He, Zhouyang Jia, Shanshan Li, Yue Yu, Chenglong Zhou, Qing Liao, Ji Wang, and Xiangke Liao. 2022. Multi-intention-aware configuration selection for performance tuning. In *Proceedings of the 44th International Conference on Software Engineering*. 1431–1442.
[46] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 517–528.
[47] Mingyu Huang, Peili Mao, and Ke Li. 2025. Rethinking Performance Analysis for Configurable Software Systems: A Case Study from a Fitness Landscape Perspective. *Proc. ACM Softw. Eng.* 2, ISSTA (2025), 1748–1771. https://doi.org/10.1145/3728954
[48] Ying Huang, Wei Li, Chengtian Ouyang, and Yan Chen. 2018. A self-feedback strategy differential evolution with fitness landscape analysis. *Soft Computing* 22, 23 (2018), 7773–7785.
[49] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of artificial intelligence research* 36 (2009), 267–306.
[50] Pooyan Jamshidi and Giuliano Casale. 2016. An uncertainty-aware approach to optimal configuration of stream processing systems. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 39–48.
[51] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 497–508.
[52] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 71–82.
[53] Terry Jones, Stephanie Forrest, et al. 1995. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *ICGA*, Vol. 95. 184–192.
[54] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-based sampling of software configuration spaces. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1084–1094.
[55] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=H1oyRlYgg
[56] Cody Kinneer, David Garlan, and Claire Le Goues. 2021. Information reuse and stochastic search: Managing uncertainty in self-* systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 15, 1 (2021), 1–36.
[57] Andrew Kirjner, Jason Yim, Raman Samusevich, Shahar Bracha, Tommi S. Jaakkola, Regina Barzilay, and Ila R. Fiete. 2024. Improving protein optimization with smoothed fitness landscapes. In *The Twelfth International Conference on*

Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net. https://openreview.net/forum?id=rxlF2Zv8x0

[58] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. *Proc. VLDB Endow.* 17, 8 (2024), 1939–1952. https://doi.org/10.14778/3659437.3659449

[59] Chi Li, Shu Wang, Henry Hoffmann, and Shan Lu. 2020. Statically inferring performance properties of software configurations. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[60] Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali R Butt, and Nicholas Fuller. 2014. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. 165–176.

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

[62] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research* 23, 54 (2022), 1–9.

[63] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2003. Iterated local search. In *Handbook of metaheuristics*. Springer, 320–353.

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

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

[66] Katherine M. Malan. 2021. A Survey of Advances in Landscape Analysis for Optimisation. *Algorithms* 14, 2 (2021), 40. https://doi.org/10.3390/A14020040

[67] Katherine M Malan and Andries P Engelbrecht. 2013. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences* 241 (2013), 148–163.

[68] Michael D McKay. 1992. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In *Proceedings of the 24th conference on Winter simulation*. 557–564.

[69] Nenad Mladenović and Pierre Hansen. 1997. Variable neighborhood search. *Computers & operations research* 24, 11 (1997), 1097–1100.

[70] Stefan Mühlbauer, Florian Sattler, Christian Kaltenecker, Johannes Dorn, Sven Apel, and Norbert Siegmund. 2023. Analysing the impact of workloads on modeling the performance of configurable software systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2085–2097.

[71] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding faster configurations using flash. *IEEE Transactions on Software Engineering* 46, 7 (2018), 794–811.

[72] Neelofar Neelofar, Kate Smith-Miles, Mario Andrés Muñoz, and Aldeida Aleti. 2022. Instance space analysis of search-based software testing. *IEEE Transactions on Software Engineering* 49, 4 (2022), 2642–2660.

[73] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 61–71.

[74] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*. 92–101.

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

[76] Erik Pitzer and Michael Affenzeller. 2012. A comprehensive survey on fitness landscape analysis. *Recent advances in intelligent engineering systems* (2012), 161–191.

[77] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 1135–1144. https://doi.org/10.1145/2939672.2939778

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

[79] Lars St, Svante Wold, et al. 1989. Analysis of variance (ANOVA). *Chemometrics and intelligent laboratory systems* 6, 4 (1989), 259–272.

[80] Peter F Stadler. 1996. Landscapes and their correlation functions. *Journal of Mathematical chemistry* 20, 1 (1996), 1–45.

[81] David G Sullivan, Margo I Seltzer, and Avi Pfeffer. 2004. Using probabilistic reasoning to automate software tuning. *ACM SIGMETRICS Performance Evaluation Review* 32, 1 (2004), 404–405.

[82] Ryoji Tanabe. 2020. Analyzing adaptive parameter landscapes in parameter adaptation methods for differential evolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 645–653.

[83] Jorge Tavares, Francisco B Pereira, and Ernesto Costa. 2008. Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38, 3 (2008), 604–616.

[84] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring performance prediction models across different hardware platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. 39–50.

[85] András Vargha and Harold D Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[86] Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, and Christian Kästner. 2020. Configcrusher: Towards white-box performance analysis for configurable systems. *Automated Software Engineering* 27 (2020), 265–300.

[87] Shihai Wang and Tao Chen. 2026. Light over Heavy: Automated Performance Requirements Quantification with Linguistic Inducement. In *48th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM.

[88] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. 2022. Machine/deep learning for software engineering: A systematic literature review. *IEEE Transactions on Software Engineering* 49, 3 (2022), 1188–1231.

[89] Supatsara Wattanakriengkrai, Dong Wang, Raula Gaikovina Kula, Christoph Treude, Patanamon Thongtanunam, Takashi Ishio, and Kenichi Matsumoto. 2022. Giving back: Contributions congruent to library dependency changes in a software ecosystem. *IEEE Transactions on Software Engineering* 49, 4 (2022), 2566–2579.

[90] Max Weber, Sven Apel, and Norbert Siegmund. 2021. White-Box Performance-Influence models: A profiling and learning approach. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1059–1071.

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

[92] Edward Weinberger. 1990. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics* 63, 5 (1990), 325–336.

[93] Thomas Weise. 2009. Global optimization algorithms-theory and application. *Self-Published Thomas Weise* 361 (2009), 153.

[94] Sewall Wright et al. 1932. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. (1932).

[95] Bowei Xi, Zhen Liu, Mukund Raghavachari, Cathy H Xia, and Li Zhang. 2004. A smart hill-climbing algorithm for application server configuration. In *Proceedings of the 13th international conference on World Wide Web*. 287–296.

[96] Zezhen Xiang, Jingzhi Gong, and Tao Chen. 2026. Dually Hierarchical Drift Adaptation for Online Configuration Performance Learning. In *48th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM.

[97] Gangda Xiong and Tao Chen. 2025. CoTune: Co-evolutionary Configuration Tuning. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*. IEEE, 1490–1502. https://doi.org/10.1109/ASE63991.2025.00126

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

[99] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. 2022. Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation. *Proc. VLDB Endow.* 15, 9 (2022), 1808–1821. https://doi.org/10.14778/3538598.3538604

[100] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2102–2114. https://doi.org/10.1145/3448016.3457291

[101] Yuanliang Zhang, Haochen He, Owolabi Legunsen, Shanshan Li, Wei Dong, and Tianyin Xu. 2021. An Evolutionary Study of Configuration Design and Implementation in Cloud Systems. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 188–200. https://doi.org/10.1109/ICSE43902.2021.00029

[102] Feng Zou, Debao Chen, Hui Liu, Siyu Cao, Xuying Ji, and Yan Zhang. 2022. A survey of fitness landscape analysis for optimization. *Neurocomputing* 503 (2022), 129–139.