Enhancing Performance Models with Intelligent Configuration Space Coverage

NEGAR MOHAMMADI KOUSHKI, Computer and Information Sciences, Temple University, USA
SANJEEV SONDUR, Computer and Information Sciences, Temple University, USA
KRISHNA KANT, Computer and Information Sciences, Temple University, USA

Accurate modeling of system performance as a function of various configuration variables (CVs) requires performance measurement for a diverse set of configurations. However, the available data is often limited and covers only a small portion of the configuration space, making it insufficient for robust modeling. Collecting additional data is both expensive and challenging, particularly in production environments where measurements are time-consuming and may inconvenience users. In this paper, we introduce an Intelligent Configuration Space Coverage (ICSC) methodology that identifies the regions of the configuration space where additional performance measurements would be most beneficial for accurate modeling, while explicitly limiting the number of such measurements required. We demonstrate that our methodology substantially enhances the accuracy of performance predictions compared to methods that choose the data points randomly or via simple considerations of gaps in the CV values. Furthermore, We show that the methodology is highly valuable even for semi-supervised learning scenarios where no new measurement campaign is needed.

CCS Concepts: • **Mathematics of computing → Probabilistic reasoning algorithms**; **Machine learning**; • **Software and its engineering → Software performance**; • **Hardware** → *Performance evaluation*; • **Applied computing** → *Enterprise computing*.

ACM Reference Format:
Negar Mohammadi Koushki, Sanjeev Sondur, and Krishna Kant. 2025. Enhancing Performance Models with Intelligent Configuration Space Coverage. 1, 1 (April 2025), 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A fundamental challenge in data-centric modeling is ensuring the adequacy and diversity of available data, as well as addressing the difficulty of collecting additional diverse data. For example, data collection may require specialized equipment, setup time, human participation (as in biomedical experiments), or interruptions in normal functioning, especially when it involves human body measurements. Therefore, careful consideration is needed when determining both the number of cases to examine and their specific characteristics before conducting experiments. A prominent example, which we focus on in this paper, is evaluating the performance of an enterprise system across diverse configurations of its parameters.

The achievable performance of an enterprise system invariably depends on the specific values of numerous hardware and software configuration variables (CVs)[1]. Adequately characterizing the performance (denoted as y) as a function of configuration vector (x) requires a diverse set of (x, y) pairs. This is particularly true for machine learning-based

This research was supported by NSF grant CNS-2011252.
Authors' addresses: Negar Mohammadi Koushki, koushki@temple.edu, Computer and Information Sciences, Temple University, 1925 N 12th St, Philadelphia, PA, USA, 19122; Sanjeev Sondur, sanjeev.sondur@temple.edu, Computer and Information Sciences, Temple University, 1925 N 12th St, Philadelphia, PA, USA, 19122; Krishna Kant, kkant@temple.edu, Computer and Information Sciences, Temple University, 1925 N 12th St, Philadelphia, PA, USA, 19122.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

characterization [2–4], which is often necessitated due to the complexity of the behavior of y wrt x, and extensive but poorly understood dependencies across CVs (i.e., the components of x).

Unfortunately, the available data is often not very diverse, since it largely concerns a small number of configurations that were tried out on the production system [5]. *This remains true despite the existence of a large amount of such data – most of it concerns dynamic factors like the utilization levels of different resources, rather than varied values of CVs.* Furthermore, production systems do not allow arbitrary configuration changes, and translating configuration from a test system to a production system is often difficult or even impossible [6, 7].

In view of these difficulties, the key question that we answer in this paper is as follows: Given an existing set of data points $(x_i, y_i), i = 1, \ldots, N$ and the willingness to collect at most K additional data points, where should these be located in the configuration space? That is, we want to seek the most informative additional configurations $x_j, j = N+1, \ldots, N+K$ for which the measurements should be performed to improve the accuracy of system performance models.

To address this challenge, we propose Intelligent Configuration Space Coverage (ICSC)—a methodology that systematically identifies the most informative regions of the configuration space for additional data collection. ICSC aims to optimize data selection by leveraging structured exploration rather than random selection or simple space-filling techniques. The approach applies broadly to any field where data collection is expensive, such as biomedical research, industrial experimentation, and enterprise system configuration. The coverage aspect has not received much attention in the literature, which is primarily focused on analyzing the available data. Often, the x_i's are chosen randomly in the context of generating additional artificial data, such as in a semi-supervised learning (SSL) context [8, 9], which too are focused on generating reliable labels (y_i's) for the added data. We henceforth denote the random addition as RAND, and show that ICSC works substantially better than RAND in the context of improving the label (or y) predictions *with the help of additional data.* When faced with the issue of choosing additional x_i's to conduct additional experiments, people might also use simple space-filling techniques, such as looking for the largest gaps in the x_i values in the existing labeled data and choosing points to reduce them. This can be considered as a greedy algorithm, henceforth called GREED. We show that ICSC works substantially better than GREED as well.

A true evaluation of such a methodology would require access to an enterprise system to obtain real y_j's for the intelligently chosen x_j's. Instead, we use the publicly available datasets that record performance against configuration parameter values. Such datasets themselves are difficult to come by since most of them simply record performance as a function of resource utilization and the like, rather than true configuration variable values. Any such dataset will not provide the y value for arbitrary x; doing so requires a model. We experimented with a large number of standard machine learning methods to predict y as a function of x and chose the best-performing model for each dataset. This quest did include methods like Kriging (Gaussian regression) [10, 11] and Bayesian optimization [12–14], which are popularly used to estimate y's from x's by explicitly taking into account the correlations in the x domain. We also explored standard SSL methods, whose goal is again to improve the estimate of y based on limited labeled data. Typically, SSL methods choose the x values for new data points arbitrarily, but in our case, we employ ICSC to determine the x for the unlabeled data points. We show that this strategy helps SSL algorithms as well.

In summary, the contributions of this paper are as follows:

- We introduce a novel mechanism called ICSC to fill the gaps in the configuration space, and thereby provide valuable guidance for planning the collection of additional experimental data subject to the practical limitations and domain-specific constraints of such an endeavor.

- We develop a quantitative metric to assess the coverage and effectiveness of the generated configurations.
- Through comprehensive experimental evaluation on multiple datasets, we demonstrate significant improvements in the accuracy of performance predictions when ICSC is used for generating x_i's as compared to the random or greedy selection of data points.

To the best of our knowledge, this is the first contribution that systematically addresses the limitations of data availability in configuration studies by intelligently generating and validating a targeted selection of new configuration data points, ensuring a comprehensive and efficient exploration of the configuration space. The code for our ICSC methodology is available online[1].

The rest of the paper is organized as follows: Section 2 discusses the background. Section 3 outlines the steps in the ICSC methodology. Section 4 covers the detailed experimental evaluation. Section 5 then concludes the discussion.

2 BACKGROUND

In an enterprise IT infrastructure, optimizing system performance by accurately configuring various system parameters is challenging due to their complexity and interdependencies. These CVs span hardware, OS, software, and workload parameters. With the widespread use of virtualization, standard hardware parameters (e.g., CPU cores, memory size, storage, and network BW) can be set dynamically. The non-linear and unpredictable interactions among CVs create a complex landscape, making it difficult to find the optimal configuration for a given workload [1]. The correct configuration is crucial for good system performance, but the task is daunting due to the many possible configurations and their unpredictable performance impacts [5, 15]. System administrators often rely on trial and error or rule-of-thumb approaches, which can be time-consuming and suboptimal [5, 15–18].

Machine learning has recently been used to address the difficulty of characterizing the impact of CVs on performance analytically [3, 4]. However, successful models that do not suffer from overfitting and are accurate require a substantial amount of diverse training data [15]. The data in this case would be in the form of the pairs $(\vec{x}, y)$ where $\vec{x}$ is the configuration vector and y is the corresponding performance. Unfortunately, the configurations $\vec{x}$ for which the y value is available are generally quite limited, largely corresponding to the "good" configurations that have been tried out [6, 7].

Experimentation with new configurations is further constrained by operational risks and resource limitations in production environments. Test systems, while useful, often differ significantly in scale and hardware, making data collection challenging and expensive [19, 20]. Given these constraints, identifying a small set of additional configurations that can maximize the diversity of the training dataset is critical. A careful choice of configurations can substantially improve the accuracy of Machine learning models compared to arbitrary or random data augmentation strategies. **We show that a careful choice of $(\vec{x}, y)$ pairs can substantially enhance the accuracy of the performance prediction models compared to an uninformed data augmentation.**

Even when additional configurations are carefully chosen, they may not provide sufficient data for training a standard Supervised Learning model. SSL [8, 9] can address data inadequacies by augmenting the real data with artificially generated $\vec{x}$ values for which the y values are not obtained via experimentation but estimated from the labeled data itself. There are many SSL techniques, however, their success depends on how the artificial data is generated and used [8, 21, 22].

[1] https://github.com/AnonUser-sudo-sys/ICSC

Various SSL techniques such as self-training and co-training have been explored in [21]. Self-training [23] involves further training the original supervised learning classifier using both original and pseudo-labeled data. An expectation-maximization (EM) approach can be applied, where all unlabeled data is included at once, iteratively adjusting parameters to maximize the likelihood that the data fits the assumed label distribution. For example, Wu et al. [24] use Naive Bayes for classification, assuming each dimension independently contributes to the label and estimating the mean and variance to fit a normal distribution for continuous variables or a binomial model for discrete values. Pseudo-labeled data may be given a lower weight than original data, either directly or through a hyperparameter.

Co-training involves training $K \geq 2$ distinct classifiers on the labeled dataset and generating pseudo-labeled data for each other. Data with a confidence level above a threshold is used for further training. If classifiers are strongly correlated, they may produce similar labels, negating the benefit of multiple classifiers. Diversity can be natural (e.g., multi-modal datasets) or induced by focusing on different CVs [25]. However, the selected CVs for each classifier must be sufficient for accurate prediction; otherwise, Co-training could reduce accuracy.

Ref [21] shows that a domain-knowledge-assisted SSL method works substantially better than traditional techniques that depend on available data for generating pseudo-labels. However, Ref [21] chooses the x vectors to label randomly. Instead, we show that by choosing the $\vec{x}$ vectors intelligently to fill the state space, we can do significantly better.

Workloads in enterprise environments can be highly variable, adding another layer of complexity to the configuration challenge. Different applications and services have distinct resource demands, and a configuration that is optimal for one workload may not be suitable for another [26, 27]. This variability makes it difficult to maintain consistent performance across different scenarios [28]. Moreover, the interdependencies among CVs can lead to unexpected interactions, complicating the task of identifying optimal configuration settings [29].

To systematically address the CV sparsity and optimize system configurations, we propose the ICSC method. This approach is illustrated in Fig. 1(b), demonstrating its application in a 3-D configuration space example shown in Fig. 1(a). In this example, blue dots • are the original data points, green triangles ▲ mark valid gaps identified by ICSC for potential new configurations, and red crosses × represent invalid gaps that do not meet the criteria for valid configurations. Together, the red crosses and green triangles make up the complement data. ICSC strategically generates and validates new configurations to overcome the limitations of existing datasets. This approach involves deliberate and systematic augmentation of the configuration dataset, either through direct experimental data collection or indirect estimation via simulation or modeling, ensuring a comprehensive exploration of the configuration space.

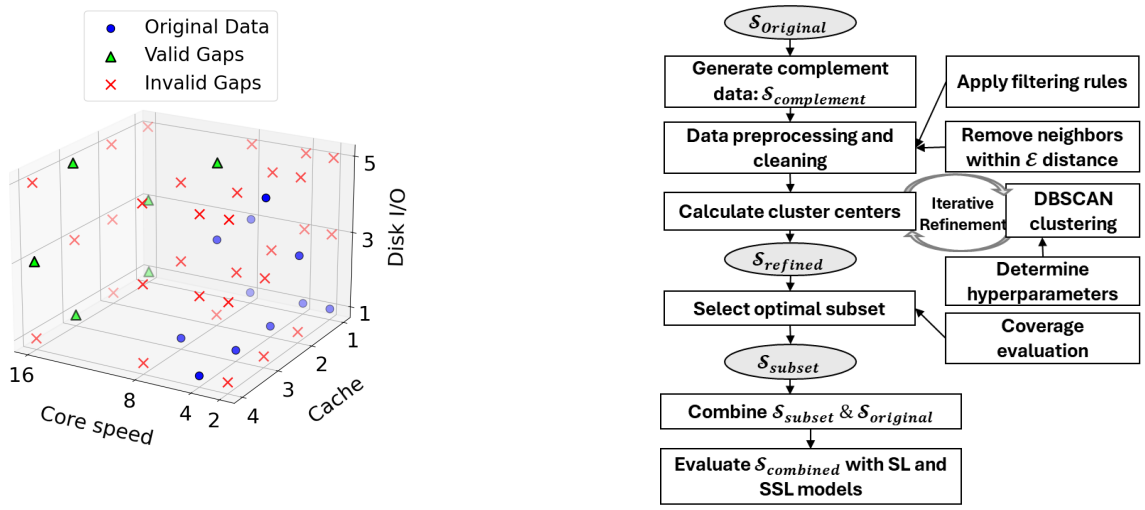
3 INTELLIGENT CONFIGURATION SPACE COVERAGE (ICSC) METHODOLOGY

In this section we describe our ICSC approach to generate, filter, cluster, and iteratively refine the additional data points.

3.1 Overview of the ICSC Methodology

ICSC consists of several key steps designed to address the challenges of optimizing system configurations in enterprise IT environments:

(1) We first identify the uncovered part of the configuration space by calculating the *complement set of configurations* missing from the original dataset.
(2) To refine the complement set of configurations into a manageable set, we apply domain-specific *filtering rules* that eliminate impractical configurations.



(a) (b)

Fig. 1. (a) Potential Gaps in the Configuration Space. (b) Overview of the ICSC Design Approach.

(3) We *identify* and remove configurations that closely resemble existing ones to ensure the dataset retains only the most representative configurations.
(4) We iteratively apply the *clustering algorithm*, compute cluster centers, and remove points within a specified distance from these centers until convergence. The cluster centers represent the potential new data points.
(5) We *employ a novel* evaluation method to determine whether a cluster center fills a new gap in the multidimensional data space. This step uses a greedy selection strategy to maximize the diversity and representativeness of the dataset.
(6) We introduce a *measure of coverage* to quantitatively assess how well the generated configurations span the configuration space, providing a concrete metric to evaluate the effectiveness of ICSC.

3.2 Configuration Space and Variables

The ICSC methodology begins with an exhaustive generation of configuration data points to comprehensively cover the configuration space. Let $\vec{x} = \{CV_1, CV_2, \ldots, CV_N\}$ denote the set of N CVs, each with a range of possible values. Each CV_j has a current value between the lower and upper bounds $\left[lb_{CV_j}, ub_{CV_j}\right]$. The set of all possible combinations of the CVs' ranges can be represented as, $\mathcal{S}_{\text{all}} = \left\{\vec{x} \mid lb_{CV_j} \leq x_j \leq ub_{CV_j}, \forall j \in \{1, 2, \ldots, N\}\right\}$. Here, $\mathcal{S}_{\text{all}}$ denotes the set of all possible configuration vectors $\vec{x}$ where each element x_j is constrained by its respective lower and upper bounds.

3.3 Measure of Coverage

To quantify the effectiveness of ICSC, we define the measure of coverage based on unique combinations of CV pairs within the configuration space. A unique combination refers to a specific pairing of values from two different CVs. For example, if the configuration space includes variables such as CPU cores and disk IO, a unique combination could be a pairing of a particular value of CPU cores with a particular value of disk IO. The measure of coverage evaluates how

well the generated configurations explore new, previously unrepresented combinations of CV pairs. This is essential for ensuring diversity and representativeness in the dataset.

To calculate the measure of coverage, we start by considering all possible pairs of CVs. For N CVs, there are $\frac{N(N-1)}{2}$ unique pairs. For each pair of CVs, we extract all unique value combinations present in the dataset. Formally, for a dataset $\mathcal{S}_{\text{all}}$ and a pair of CVs (CV_i, CV_j), the set of unique combinations is defined as, $C(\mathcal{S}_{\text{all}}, CV_i, CV_j) = \{(x_{CV_i}, x_{CV_j}) \mid (x_{CV_i}, x_{CV_j}) \in \mathcal{S}_{\text{all}}\}$. Next, we calculate the number of unique combinations for the original dataset $\mathcal{S}_{\text{orig}}$ and the combined dataset $\mathcal{S}_{\text{comb}}$. The combined dataset includes both the original data and the intelligently generated configurations: $\mathcal{S}_{\text{comb}} = \mathcal{S}_{\text{orig}} \cup \mathcal{S}_{\text{opt}}$. We then define the coverage ratios for the original and combined datasets, denoted as OC and CC, respectively. These ratios are given by $\text{OC} = \frac{|C_{\text{orig}}|}{|C_{\text{all}}|}$ and $\text{CC} = \frac{|C_{\text{comb}}|}{|C_{\text{all}}|}$ where C_{orig} and C_{comb} are the sets of unique combinations in the original and combined datasets, respectively, and C_{all} is the set of all possible unique combinations in the configuration space. Finally, we determine the improvement in coverage by comparing the coverage ratios.

3.4 Initial Refinement of Configuration Data

To enhance the coverage of the configuration space, we adopt a multi-step refinement process. This process begins with the identification of complement data, which are the configurations absent in the original dataset. The set of complement configurations, denoted as $\mathcal{S}_{\text{comp}}$, is derived by subtracting the original set of configurations from the universal set of all possible configurations: $\mathcal{S}_{\text{comp}} = \mathcal{S}_{\text{all}} \setminus \mathcal{S}_{\text{orig}}$. This step is crucial for identifying gaps in the configuration space that need to be explored.

Next, we refine $\mathcal{S}_{\text{comp}}$ into a manageable set of configurations by applying a set of filtering rules $\mathcal{F}$ based on domain knowledge in the form of configuration parameter combinations that are considered unreasonable or impractical. Each filtering rule f_i is a boolean function, $f_i : \mathcal{S}_{\text{comp}} \rightarrow \{0, 1\}$, designed to enforce these constraints and ensure feasibility. For example, in a cloud storage environment, domain knowledge might dictate that certain combinations of CPU and memory configurations are impractical due to hardware limitations or performance degradation under specific conditions. By incorporating such rules, we ensure that configurations like these are excluded from $\mathcal{S}_{\text{cf}}$, resulting in a dataset that is both relevant and feasible. This approach is validated through previous studies and methodologies that emphasize the importance of domain-specific constraints in enhancing the reliability and applicability of configuration datasets [30–33]. Thus, a configuration $\vec{x} \in \mathcal{S}_{\text{comp}}$ is considered valid if it satisfies all filtering rules, $\mathcal{S}_{\text{cf}}$.

To enhance uniqueness and reduce redundancy, we further refine the set $\mathcal{S}_{\text{cf}}$ by identifying and removing neighboring configurations. A neighborhood $\mathcal{N}(\vec{x}_i)$ for each configuration $\vec{x}_i$ in $\mathcal{S}_{\text{cf}}$ is defined as the set of configurations within a small Euclidean distance η. Configurations within this distance are considered redundant because they closely resemble existing CVs in terms of their variable values. Such points are unlikely to provide novel insights or significantly enhance

Algorithm 1: Initial Refinement of Configuration Data

Function `InitialRefinement`$(\mathcal{S}_{all}, \mathcal{S}_{orig}, \eta)$:
  $\mathcal{S}_{\text{comp}} \leftarrow \mathcal{S}_{\text{all}} \setminus \mathcal{S}_{\text{orig}}$;
  $\mathcal{S}_{\text{cf}} \leftarrow \left\{\vec{x} \in \mathcal{S}_{\text{comp}} \mid \bigwedge_{i=1}^{K} f_i(\vec{x}) = 1\right\}$;
  $\mathcal{S}_{\text{cvd}} \leftarrow \mathcal{S}_{\text{cf}}$;
  for *each* $\vec{x}_i \in \mathcal{S}_{cf}$ **do**
    $\mathcal{N}(\vec{x}_i) \leftarrow \left\{\vec{x}_j \in \mathcal{S}_{\text{cf}} \mid \|\vec{x}_i - \vec{x}_j\|_2 \leq \eta\right\}$;
    $\mathcal{S}_{\text{cvd}} = \mathcal{S}_{\text{cf}} \setminus \bigcup_{i=1}^{|\mathcal{S}_{\text{cf}}|} \mathcal{N}(\vec{x}_i)$;
  return $\mathcal{S}_{\text{cvd}}$;

the diversity of the configuration dataset. The choice of η is based on the scale and nature of $\vec{x}$, ensuring effective differentiation between distinct configurations. After this step, the refined set of valid configurations is denoted as $\mathcal{S}_{\text{cvd}}$. The detailed process of this refinement is outlined in Algorithm 1. This algorithm provides a structured approach to deriving a comprehensive, relevant, and non-redundant configuration dataset from the initial set of all possible configurations.

3.5 Iterative Clustering and Refinement

The iterative clustering and refinement process is crucial for ensuring that the configuration space is comprehensively covered by representative configurations. This process aims to iteratively reduce redundancy and enhance the diversity of the configuration dataset. The steps involved are as follows:

Step 1: Start with the set of complement valid configurations, $\mathcal{S}_{\text{cvd}}$, obtained after applying the filtering rules. This set represents the initial candidate configurations for further refinement.

Step 2: Apply DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [34] to $\mathcal{S}_{\text{cvd}}$ to identify clusters. Density-based clustering methods [35], such as DBSCAN, are particularly suitable for our problem as they can effectively identify dense regions in the configuration space that correspond to clusters of configurations with similar performance characteristics such as CPU utilization, memory usage, response time, and throughput.

DBSCAN works well in high-dimensional spaces and can find clusters of arbitrary shapes, making it ideal for dealing with complex and non-linear interactions among CVs. Unlike centroid-based methods such as K-means, DBSCAN does not require the number of clusters to be specified in advance and handles noise effectively, providing robust and flexible clustering results. Let the clusters be denoted as $\{C_1, C_2, \ldots, C_K\}$.

Step 3: For each cluster C_k, compute the center μ_k as the mean of all configurations in the cluster, representing the central configuration that summarizes the cluster.

Step 4: To ensure diversity and reduce redundancy, remove configurations within a specified ε-distance from each cluster center μ_k. This step refines $\mathcal{S}_{\text{cvd}}$ by eliminating configurations that are too similar to the cluster centers, maintaining a diverse set of candidates.

Step 5: Iteratively repeat the clustering, center calculation, and neighbor removal steps. After each iteration, check if the number of configurations in $\mathcal{S}_{\text{cvd}}$ has stabilized. Convergence is achieved when the number of valid configurations remains consistent across consecutive iterations, indicating that further refinement does not significantly alter the dataset.

Step 6: Continue the iterative process until convergence. This ensures that the final set of configurations is both diverse and representative, comprehensively covering the configuration space without redundancy.

The iterative clustering and refinement process is formally described in Algorithm 2.

3.6 Selecting an Optimal Subset of $\mathcal{S}_{\text{refined}}$

The final step in the ICSC methodology involves selecting an optimal subset from the refined set of configurations $\mathcal{S}_{\text{refined}}$. This subset should maintain the diversity and representativeness of the original refined set while minimizing redundancy and ensuring that the number of additional configurations remains optimal. To achieve this, we define the optimization problem with a clear objective and constraints. The primary objective of the optimization problem is to maximize the coverage of new combinations of CV pairs in the configuration space, thus filling gaps not previously covered by the original dataset $\mathcal{S}_{\text{orig}}$. The objective function can be expressed as maximizing the sum of new CV value combinations introduced by each configuration in $\mathcal{S}_{\text{refined}}$. Formally, this is represented as,

Algorithm 2: Iterative Clustering and Refinement

Function `IterCluster`($\mathcal{S}_{cvd}, \varepsilon, \delta$):
  Initialize $\mathcal{S}_{\text{current}} \leftarrow \mathcal{S}_{\text{cvd}}$;
  Initialize converged $\leftarrow$ false ;
  while *not converged* **do**
    Apply DBSCAN to $\mathcal{S}_{\text{current}}$ to identify clusters $\{C_1, C_2, \ldots, C_K\}$;
    Initialize $\mathcal{S}_{\text{next}} \leftarrow \mathcal{S}_{\text{current}}$;
    for *each cluster* $C_k \in \{C_1, C_2, \ldots, C_K\}$ **do**
      Compute cluster center $\mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$;
      Remove points within ε-distance from μ_k in $\mathcal{S}_{\text{next}}$;
      $\mathcal{S}_{\text{next}} \leftarrow \mathcal{S}_{\text{next}} \setminus \{x \in C_k \mid \|x - \mu_k\|_2 \leq \varepsilon\}$
    if $|\mathcal{S}_{current}| - |\mathcal{S}_{next}| \leq \delta$ **then**
      Set converged $\leftarrow$ true ;
    else
      Set $\mathcal{S}_{\text{current}} \leftarrow \mathcal{S}_{\text{next}}$;
  Set $\mathcal{S}_{\text{refined}} \leftarrow \mathcal{S}_{\text{current}}$;
  return $\mathcal{S}_{\text{refined}}$;

Maximize $\sum_{\vec{x}_k \in \mathcal{S}_{\text{refined}}} \text{NewCombinations}(\vec{x}_k)$, where $\text{NewCombinations}(\vec{x}_k)$ denotes the number of new CV value combinations introduced by configuration $\vec{x}_k$. The optimization problem is subject to two main constraints. First, the cardinality constraint ensures that the size of the selected subset $\mathcal{S}_{\text{opt}}$ does not exceed a predefined limit L, $|\mathcal{S}_{\text{opt}}| \leq L$. Second, the diversity constraint requires that the configurations in $\mathcal{S}_{\text{opt}}$ introduce new combinations not present in $\mathcal{S}_{\text{orig}}$, $C(\mathcal{S}_{\text{opt}}) \cap C(\mathcal{S}_{\text{orig}}) = \emptyset$.

The problem can be easily seen to be NP-hard via a reduction from the standard set-cover problem. Consequently, we employ a heuristic algorithm to find a near-optimal solution within a reasonable time frame. The heuristic selection process begins by initializing an empty subset $\mathcal{S}_{\text{opt}}$ and a set of existing combinations for each CV combination in the original dataset $\mathcal{S}_{\text{orig}}$. For each configuration $\vec{x}_k \in \mathcal{S}_{\text{refined}}$, we evaluate whether it introduces new combinations for any configuration dimensions. Let $C(\vec{x}_k, CV_i, CV_j)$ represent the combination of CV values (CV_i, CV_j) for configuration $\vec{x}_k$. A configuration $\vec{x}_k$ is considered valuable if it introduces at least one new combination, $\text{is\_new}(\vec{x}_k)$.

To implement the selection process, we use a greedy algorithm that iteratively selects configurations from $\mathcal{S}_{\text{refined}}$ that introduce the most new combinations. For each selected configuration $\vec{x}_k$, we update the subset $\mathcal{S}_{\text{opt}} \leftarrow \mathcal{S}_{\text{opt}} \cup \{\vec{x}_k\}$ and the dictionary $C(\mathcal{S}_{\text{orig}}$ accordingly, $C(\mathcal{S}_{\text{orig}}(CV_i, CV_j) \leftarrow C(\mathcal{S}_{\text{orig}}(CV_i, CV_j) \cup \{C(\vec{x}_k, CV_i, CV_j)\}$. The process continues until no more configurations in $\mathcal{S}_{\text{refined}}$ can introduce new combinations or the predefined limit L is reached. The detailed steps of the algorithm are outlined in Algorithm 3.

Algorithm 3: Greedy Algorithm for Selecting $\mathcal{S}_{\text{opt}}$

Function `SelectOptimalSubset`($\mathcal{S}_{refined}, L$):
  Initialize $\mathcal{S}_{\text{opt}} \leftarrow \emptyset$;
  Initialize $C_{\text{original}} \leftarrow C(\mathcal{S}_{\text{orig}})$ based on dataset $\mathcal{S}_{\text{orig}}$;
  while $|\mathcal{S}_{opt}| \leq L$ **do**
    for *each* $\vec{x}_k \in \mathcal{S}_{refined} \setminus \mathcal{S}_{opt}$ **do**
      Evaluate if $\vec{x}_k$ introduces new combinations: ;
      $\text{is\_new}(\vec{x}_k) = \bigvee_{(CV_i, CV_j) \in \mathcal{CV}} \left(C(\vec{x}_k, CV_i, CV_j) \notin C(\mathcal{S}_{\text{orig}}, CV_i, CV_j)\right)$
      if *is_new*$(\vec{x}_k)$ **then**
        Add $\vec{x}_k$ to $\mathcal{S}_{\text{opt}}$;
        Update C_{original} with new combinations from $\vec{x}_k$;
  return $\mathcal{S}_{\text{opt}}$

3.7 DBSCAN Hyperparameters

DBSCAN relies on two key hyperparameters that must be set accurately to obtain the desired clustering results.

3.7.1 Maximum Distance Threshold (ε). The maximum distance threshold, denoted as ε, defines the radius of the hypersphere within which the points are considered neighbors. For a dataset with N dimensions, a practical approach to choosing ε is to ensure that it represents a significant but not overly broad neighborhood in the configuration space. To standardize ε across different datasets, a dimensionless measure is employed by dividing ε by the number of dimensions N, $\varepsilon_{\text{dimensionless}} = \frac{\varepsilon}{N}$. This ensures ε is appropriately scaled with respect to the dimensionality, facilitating a balanced clustering process. We then set $\varepsilon = 0.3\sqrt{N}$ as a baseline value. This represents 30% of the maximum Euclidean distance in the configuration space. For a dataset with 7 CVs, this gives $\varepsilon_{\text{initial}} \approx 0.8$.

3.7.2 Minimum number of Points in a Cluster (MinPts). The minimum number of points within an ε-neighborhood, denoted as *MinPts*, is crucial for defining a dense region. The value of *MinPts* should be large enough to avoid detecting stray points (or "noise") as clusters, but small enough to identify meaningful groups. To estimate a suitable *MinPts*, we consider the total number of points P in the configuration space, where $P = \prod_{i=1}^{N} |F_i|$ and $|F_i|$ represents the number of distinct values for the i-th feature across N dimensions. The average density per dimension $\overline{d}$ is $P^{1/N}$, and the density within the ε-neighborhood is $d_\varepsilon = (\varepsilon \cdot \overline{d})^N$. This provides an initial *MinPts* estimate, refined through practical testing and silhouette scores. For example, with $P = 2592$, $N = 7$, and $\varepsilon \approx 0.8$, $d_\varepsilon \approx 19$, suggesting an initial *MinPts* of around 19. Practical experimentation and evaluation using silhouette scores refine this initial estimate.

3.7.3 Silhouette Score for Finding Optimal ε and MinPts. The silhouette score [36] evaluates the clustering quality by measuring how similar a point is to its own cluster (cohesion) versus other clusters (separation). For a point i in cluster A, it is defined as $s(i) = \frac{b(i)-a(i)}{\max(a(i),b(i))}$, where $a(i)$ is the average distance from i to other points in A, and $b(i)$ is the minimum average distance from i to any other cluster B. The score ranges from -1 to 1, with higher values indicating better clustering. The average score across all points reflects overall clustering quality.

To find the optimal values for ε and *MinPts*, we perform a grid search over a range of values for these parameters. The grid search evaluates the silhouette score for each combination of ε and *MinPts* values to determine the pair that maximizes the overall silhouette score by evaluating $\arg\max_{\varepsilon, MinPts} \frac{1}{N} \sum_{i=1}^{N} s(i)$ where N is the total number of points.

Given an initial estimate for ε and *MinPts*, we define their ranges as $\varepsilon \in [\varepsilon_{\text{initial}} \pm \Delta\varepsilon]$ and $MinPts \in [MinPts_{\text{initial}} \pm \Delta MinPts]$, where $\Delta\varepsilon$ and $\Delta MinPts$ are the step sizes for the grid search. To explore a reasonable range around $\varepsilon_{\text{initial}}$, we define $\varepsilon \in [0.6 \cdot \varepsilon_{\text{initial}}, 1.4 \cdot \varepsilon_{\text{initial}}]$. For $\varepsilon_{\text{initial}} \approx 0.8$, this gives $\varepsilon \approx [0.48, 1.12]$. This range is chosen to balance sensitivity and robustness in the clustering process, ensuring that ε values are neither too small to miss significant clusters nor too large to merge distinct clusters.

In addition, to explore a practical range around $MinPts_{\text{initial}}$, we can define $MinPts \in [10, 30]$. This range is chosen based on empirical observations of typical point densities in configuration spaces, ensuring that clusters are meaningful and dense enough to be statistically significant while avoiding the inclusion of noise. For each pair $(\varepsilon, MinPts)$ in these ranges, we calculate the silhouette score as follows: (1) perform clustering with the given ε and *MinPts*; (2) calculate the average silhouette score for the resulting clusters; (3) identify the pair $(\varepsilon, MinPts)$ that yields the highest average silhouette score; (4) stop the iteration when the silhouette score does not improve significantly over several iterations.

Table 1. Datasets: configuration variables and output.

Dataset	Domain	Configuration Variables (CVs) $\vec{x}$	Output (Label y)
ES [37]	Cloud Storage	CPU Cores, Disk IO Rate, Cache, Metadata Size, No. of Files, File Size	Performance
GB [38]	CPU Benchmarking	Clock Speed, L2, L3, Memory, Proc. Cap., Num. of Threads	Throughput
BB [39]	Virtual Machines	CPU Cores, Core Speed, Memory, Network (Rx/Tx), Disk R/W	CPU Usage (%)

4 EXPERIMENTAL EVALUATION

4.1 Dataset Overview: A Closer Look at the Data

We applied ICSC to the datasets listed in Table 1 and described below. A complete list of acronyms used in this paper is provided in Table 2.

Table 2. Table of Acronyms.

Term	Full Form	Term	Full Form
ES	Edge Storage Dataset	GB	Geekbench Dataset
BB	BitBrains Dataset	CVs	Configuration Variables
KDE	Kernel Density Estimate	MSE	Mean Squared Error
RF	Random Forest	SSL	Semi-Supervised Learning

Cloud/Edge Storage Dataset (ES): Cloud or Edge Storage (ES)[2] enables fast local storage access at an edge node by caching a portion of remote cloud storage. This approach optimizes limited local compute and storage resources for improved performance [37, 40]. ES systems involve numerous hardware and software configuration parameters that must be carefully set to avoid IO timeouts and high latencies caused by fluctuating data transfer rates to the cloud. The dataset includes performance metrics for 991 configurations.

Geekbench Dataset (GB): Geekbench dataset[3] contains benchmarking results focused on CPU performance under various workloads. Collected using the Geekbench 3 suite, a tool widely used by consumers and researchers, the dataset measures performance across different microarchitectures, considering factors like frequency, core count, cache size, and memory bandwidth [38]. It includes results for hundreds of Intel CPUs, providing insights into how different specifications impact single-core and multi-core performance across diverse computational tasks.

BitBrains Dataset (BB): BitBrains dataset[4] contains performance logs from 1,750 virtual machines (VMs) in BitBrains's data center, which offers various cloud services. This dataset is a 5-month time series analyzed at Delft University of Technology [39]. Iosup et al.[39] details the characterization of both requested and used resources, including CPU, memory, disk, and network data, along with the initial VM configurations.

We have explored many publicly available datasets, but unfortunately, almost all are in the form of time-series data rather than performance as an explicit function of different configurations. Only the ES dataset does the latter; the others (GB & BB), although adequate to illustrate the methodology, are not true configuration datasets.

4.2 Experimental Setup

To evaluate the effectiveness of the ICSC methodology, we conducted experiments on three distinct datasets: ES, GB, and BB. Our approach aimed to assess how well the method enhances prediction accuracy and configuration optimization

[2] [ES] https://github.com/AnonUser-sudo-sys/ES-dataset
[3] [GB] https://github.com/Emma926/cpu_selection
[4] [BB] http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains (RND500)

Table 3. Comparison of MSE and R^2 Across Models.

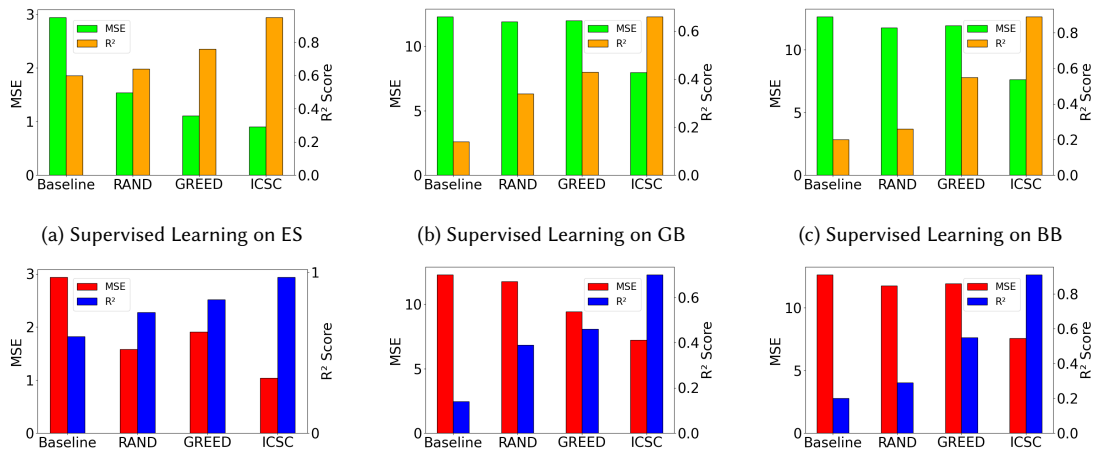
Dataset	Method	MSE	R^2
ES	Gradient Boosting	9.52	0.86
	Random Forest	**1.04**	**0.97**
	Bayesian	2.86	0.76
	Kriging	19.10	0.60
GB	Gradient Boosting	9.20	0.72
	Random Forest	**7.22**	**0.76**
	Bayesian Ridge	8.73	0.68
	Kriging	12.63	0.50
BB	Gradient Boosting	**7.56**	**0.91**
	Random Forest	11.34	0.36
	Bayesian	7.61	0.79
	Kriging	15.03	0.20

by filling the gaps in the original configuration space more comprehensively. For each dataset, we began by splitting the data into training and test sets using a standard 5-fold cross-validation method, reserving approximately 20% of the labeled data for testing while using the remaining 80% for training. We extracted the relevant CVs as features ($\vec{x}$) and the performance metrics as the target variable (y) from each dataset. For instance, in the ES dataset, CVs included CPU cores, disk IO rate, cache, metadata size, no. of files, and file size while the label y was performance. The features ($\vec{x}$) were standardized using a `StandardScaler` to ensure uniform scaling across all CVs.

To augment the space coverage, we generated additional data points using the ICSC methodology, which strategically filled the gaps in the configuration space. For comparison, we also generated random data points (RAND) within the feature ranges of the original data. The labels y for these CVs, both randomly and intelligently generated, were predicted using a model trained on the Baseline. Additionally, we included a comparison using a greedy algorithm (GREED) designed to select new data points by iteratively identifying the largest uncovered gaps in the configuration space. For a fair comparison, we ensure that the number of additional data points added by RAND and GREED is equal to the number of additional points selected by ICSC. This ensures that performance improvements are solely due to the selection strategy rather than the quantity of additional data. In this approach, for each dimension of the configuration space, we calculate the gaps between consecutive points and select the midpoint of the largest gap. This greedy algorithm, which adds one point at a time, ensures that new data points are strategically placed in regions with minimal coverage. The reason to consider such an approach is that it is very intuitive and may be used in the absence of a well-designed intelligent mechanism explored here.

4.3 Performance Prediction Model

Given the complexity of the dependence of performance on the configuration and workload parameters, a simple analytic model is typically inadequate; instead, it is necessary to explore machine learning models. It is well known that no single machine learning model is optimal in all cases; therefore, we evaluated over 15 different supervised learning models, including Linear Regression [41], Random Forest (RF) [42], and Gradient Boosting [43]. The selection of the final model for each dataset was based on its performance in terms of MSE and R^2 score during cross-validation. Table 3 illustrates the results for four key models (out of 15) that we tried for all three datasets. It is seen that Random Forest (RF) works the best for ES and GB datasets, whereas Gradient Boosting works the best for BB. Kriging and Bayesian optimization did not generally perform well for any of the datasets.



(a) Supervised Learning on ES (b) Supervised Learning on GB (c) Supervised Learning on BB

(d) SSL on ES (e) SSL on GB (f) SSL on BB

Fig. 2. Supervised Learning and SSL Model Performance on Different Datasets.

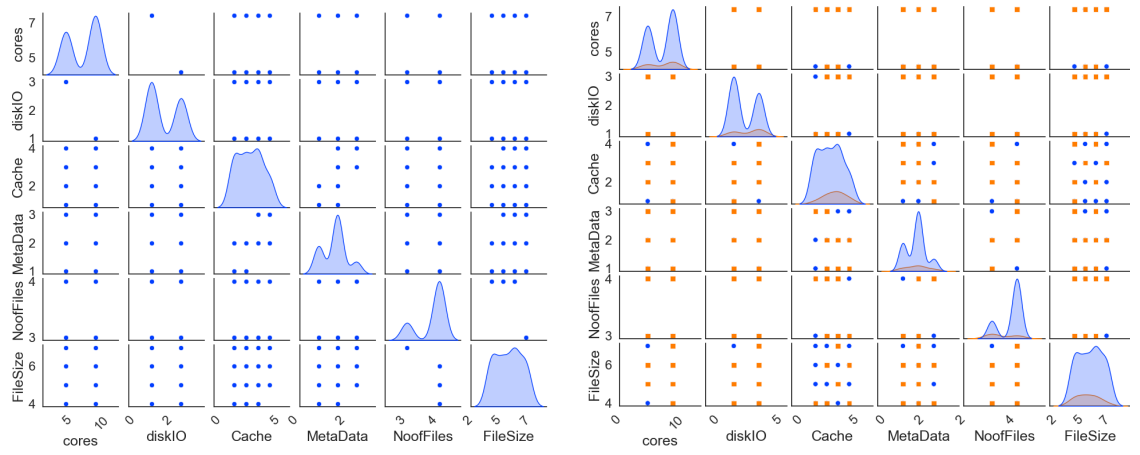
4.4 Influence of ICSC on Performance Prediction

In this section, we explore how the performance prediction is influenced by the selection of x_i's for additional data used in the model. In all cases, the corresponding y_i's are obtained from the best performing model trained on the original dataset. In particular, we compare the performance of ICSC against the RAND and GREED mechanisms for selecting x_i's. We also show the performance of the *Baseline* model trained only on the original data.

The results are shown in Fig. 2a for the ES dataset. In each plot, the left y-axis shows the MSE using green bars, where lower values are better. The right y-axis shows the R^2 score using orange bars indicating the R^2 score, where higher values are better. It is seen that the model trained on ICSC achieves the best performance, with an MSE of 0.90 and R^2 of 0.95. This corresponds to a 69.4% improvement in MSE and a 58.3% improvement in R^2 compared to the Baseline. In comparison, RAND achieves an MSE of 1.54 (47.6% improvement) and R^2 of 0.64 (6.7% improvement), while GREED achieves an MSE of 1.11 (62.2% improvement) and R^2 of 0.76 (26.7% improvement). However, neither method matches the performance of ICSC. These results demonstrate that ICSC significantly outperforms both RAND and GREED methods by effectively targeting the most informative gaps in the configuration space.

Another way to examine the benefit of ICSC in choosing x_i's to examine its influence on the accuracy of performance prediction using SSL models. Such models do not require the effort of conducting experiments to obtain the corresponding y_i's; however, the number of artificial data points added and how they are generated is still crucial. It is shown, for example, in [21] that adding too many artificial data points can hurt the accuracy. It is also shown in [21] that the self-training-based SSL is not only simple and intuitive but also generally performs the best. Therefore, we use that in this paper.

Fig. 2d shows the SSL performance on the ES dataset. It is seen that ICSC again delivers superior performance. It achieves an MSE of 1.04 and R^2 of 0.97, representing a 64.6% and 61.7% improvement over the Baseline, respectively. It may be noted that randomly generated data can occasionally achieve better MSE than GREED, as for example, is the case here. This is because its broader and unstructured exploration of the space can occasionally result in a more diverse and representative configuration set, enhancing model generalization.



(a) ES Configuration Space with Original Data Points.

(b) ES Configuration Space with Filled Gaps.

Fig. 3. Comparison of ES Configuration Space Before and After Applying ICSC.

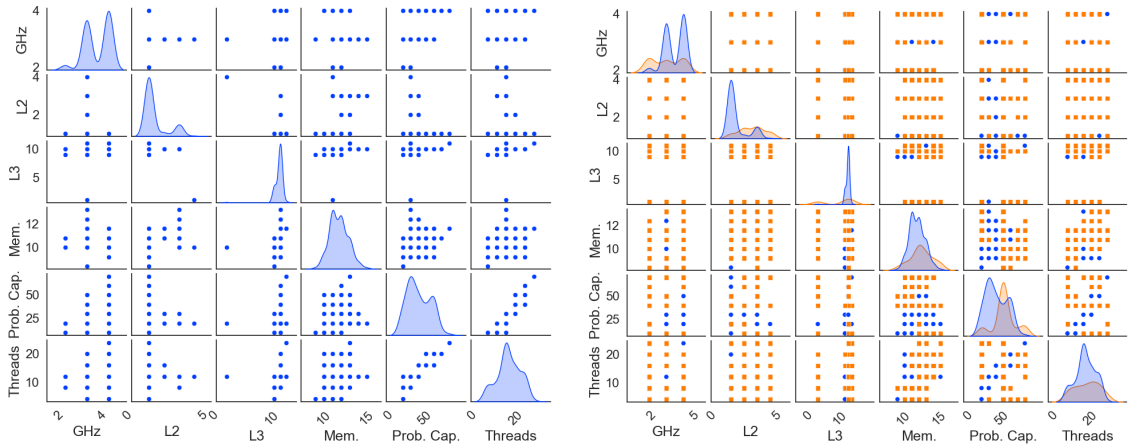
For the GB dataset (Fig. 2b), the differences between the methods become more apparent. The model trained on the Baseline results in an MSE of 12.29 and an R^2 score of 0.14. RAND results in an MSE of 11.90 (3.2% improvement) and R^2 of 0.34 (142.86% improvement), while GREED shows modest improvement with an MSE of 11.99 (2.5% improvement) and R^2 of 0.43 (207.2% improvement). In contrast, ICSC achieves an MSE of 7.96 and R^2 of 0.66—a 35.2% reduction in MSE and a 371.4% improvement in R^2 compared to the Baseline. Similar trends are observed for the SSL models (Fig. 2e), where ICSC achieves the lowest MSE of 7.22 (41.2% improvement) and the highest R^2 of 0.70 (400% improvement), outperforming both RAND and GREED approaches. Table 3 further confirms RF as the best-performing model for the GB dataset.

Finally, for the BB dataset (Fig.2c), the superiority of ICSC is evident. The Gradient Boosting model trained on the Baseline achieves an MSE of 12.63 and an R^2 score of 0.20. However, training on ICSC results in the lowest MSE of 7.64 (39.5% improvement) and the highest R^2 score of 0.89 (345% improvement), achieved by adding 1.4% of the valid complement data. RAND achieves an MSE of 11.77 (6.8% improvement) and R^2 of 0.26 (30% improvement), while GREED shows an MSE of 11.94 (5.5% improvement) and R^2 of 0.55 (175% improvement). The SSL models follow the same pattern (Fig.2f), where ICSC achieves an MSE of 7.56 (40.2% improvement) and R^2 of 0.91 (355% improvement), outperforming all other methods. As highlighted in Table 3, Gradient Boosting was identified as the optimal model for the BB dataset.

4.5 Quality of Coverage by ICSC

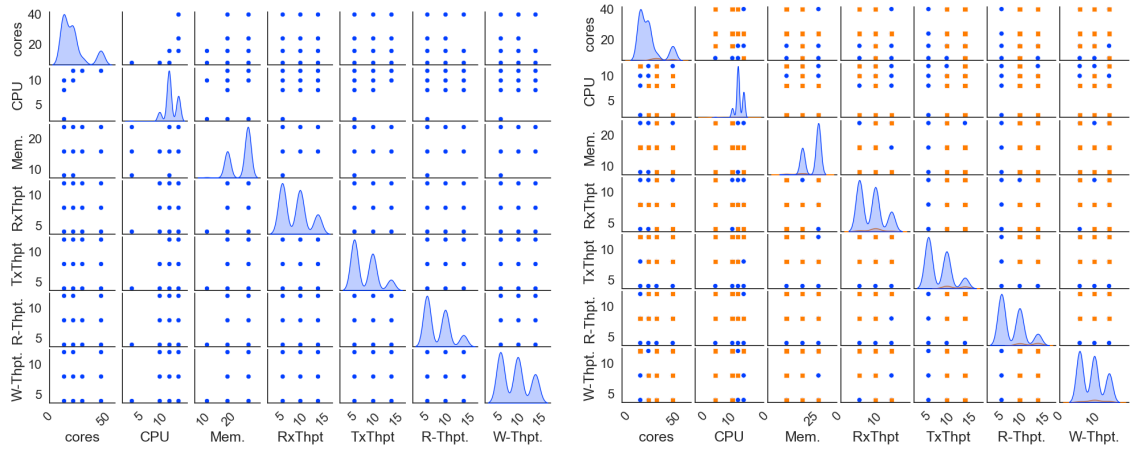
In this section, we present results on how the ICSC mechanism helps to cover the configuration space. We show this both via a series of visualization plots for different datasets and by evaluation of a coverage metric.

Fig. 3a provides a comprehensive visualization of the CV relationships within the dataset. Each subplot in the pair plots represents a unique combination of two CVs, allowing for a detailed examination of their interactions. The diagonal subplots display kernel density estimates (KDE) for each CV, highlighting the distribution patterns within the data. The off-diagonal subplots depict the pairwise relationships through scatter plots, with each point corresponding to an individual data entry. We focused on pairwise combinations of CVs to maintain clarity and readability, as including



(a) GB Configuration Space with Original Data Points. (b) GB Configuration Space with Filled Gaps.

Fig. 4. Comparison of GB Configuration Space Before and After Applying ICSC.



(a) BB Configuration Space with Original Data Points. (b) BB Configuration Space with Filled Gaps.

Fig. 5. Comparison of BB Configuration Space Before and After Applying ICSC.

higher-order combinations would result in overly complex visualizations, making it difficult to identify meaningful patterns within the data.

Fig. 3a illustrates the original ES configuration space, where distinct regions are evident, but there are also noticeable gaps. These gaps indicate unexplored areas that limit the dataset's ability to fully capture the relationships between CVs. In contrast, Fig. 3b shows the configuration space after these gaps have been filled with intelligently generated data points. This additional data enhances the dataset's coverage by 11%, leading to a more complete and robust representation of the configuration space.

Fig. 4a and Fig. 4b illustrate the GB dataset's configuration space before and after applying the ICSC methodology. Fig. 4a shows significant gaps in the original data points, with only 48% coverage. In contrast, Fig. 4b demonstrates how

Table 4. Coverage Metrics: Original vs. Combined Data.

Dataset	$\|\mathcal{S}_{\mathbf{orig}}\|$	$\|\mathcal{S}_{\mathbf{opt}}\|$	*RS%*	$\|C_{\mathbf{orig}}\|$	$\|C_{\mathbf{comb}}\|$	$\|C_{\mathbf{all}}\|$
ES	58	10	0.4	106	118	118
GB	121	72	0.78	191	340	393
BB	380	23	1.4	193	226	226

intelligently generated data points fill these gaps, achieving 86% coverage. Fig. 5a and Fig. 5b illustrate the configuration space of the BB dataset before and after applying ICSC. The original data, shown in Fig. 5a, has significant gaps, with only 48% coverage. After applying ICSC, Fig. 5b shows these gaps filled, achieving 100% coverage.

Table 4 shows the number of data points added to each dataset and the resulting increase in coverage. Here ($|\mathcal{S}_{\text{orig}}|$) is the number of points in the original dataset and ($|\mathcal{S}_{\text{opt}}|$) the number of data points added. Notably, only a relatively small number of new data points were added, achieving the desired outcome. The column (*RS%*) expresses these as a percentage of the total number of valid uncovered configurations, or $|\mathcal{S}_{\text{cvd}}|$. The table also shows the number of unique CV combinations in the original dataset ($|C_{\text{orig}}|$) and after adding the new data points ($|C_{\text{comb}}|$) datasets. The last column expresses the total possible combinations ($|C_{\text{all}}|$). Fig. 6 shows the original coverage (OC) and coverage of the combined dataset (CC) pictorially. It is seen that the coverage increased substantially in all cases, becoming 100% for ES and BB.

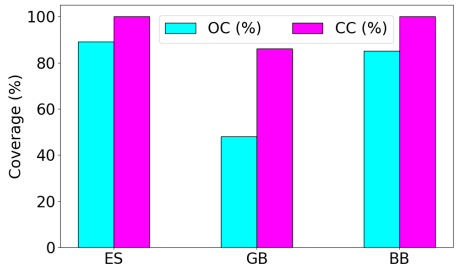


Fig. 6. Comparison of Original Coverage (OC) and Combined Coverage (CC) of Different Datasets.

5 CONCLUSIONS

In this paper, we introduced the ICSC methodology to select additional data points (or x values) from the configuration space to augment the existing data points. The purpose of this is to identify a small set of strategically chosen configurations for which additional experimentation needs to be done to maximize the coverage of the configuration space and thereby enhance the accuracy of the predictions made by machine learning models trained on the data.

Experimental evaluations show that ICSC significantly enhances model performance across multiple datasets, even with a small number of additional data points for which measurements need to be done. We also show that a prediction model trained on ICSC-augmented data exhibits significantly lower MSE and higher R^2 scores. For example, in the GB dataset, the random forest model improved prediction accuracy by 35.2%, achieving an MSE of 7.96 and an R^2 of 0.66. The methodology is also useful in the context of semi-supervised learning (SSL), where a careful selection of x values reduces the MSE to 7.22 and increases the R^2 score to 0.70. Compared to the Baseline, RAND, and GREED models, ICSC shows an MSE reduction of 41.2%, 32.2%, and 33.6%, respectively, and an R^2 improvement of 400%, 105.9%, and 62.8%, respectively.

REFERENCES

[1] O. Alipourfard, et al., "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th Symp. NSDI*, 2017, pp. 469–482.

[2] V. A. Farias and et al., "Machine learning approach for cloud nosql databases performance modeling," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016, pp. 617–620.

[3] B. Mahesh, "Machine learning algorithms-a review," *IJSR*, vol. 9, no. 1, pp. 381–386, 2020.

[4] M. Velez, et al., "Configcrusher: towards white-box performance analysis for configurable systems," *Automated Software Engineering*, pp. 1–36, 2020.

[5] T. Xu and Y. Zhou, "Systems approaches to tackling configuration errors: A survey," *CSUR*, vol. 47, no. 4, pp. 1–41, 2015.

[6] M. Iqbal, et al., "Unicorn: Reasoning about Config. Sys. Perf. through lens of Causality," in *EuroSys*. ACM, 2022.

[7] S. Venkataraman, et al., "Ernest: Efficient performance prediction for large-scale advanced analytics," in *NSDI*, 2016, pp. 363–378.

[8] E. Van, et al., "A survey on semi-supervised learning," *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.

[9] X. J. Zhu, "Semi-supervised learning literature survey," *CS Technical Reports*, 2005.

[10] N. Cressie, "The origins of kriging," *Mathematical geology*, vol. 22, pp. 239–252, 1990.

[11] M. A. Oliver and R. Webster, "Kriging: a method of interpolation for geographical information systems," *IJGIS*, vol. 4, no. 3, pp. 313–332, 1990.

[12] D. J. MacKay, "Bayesian interpolation," *Neural computation*, vol. 4, no. 3, pp. 415–447, 1992.

[13] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of machine learning research*, vol. 1, no. Jun, pp. 211–244, 2001.

[14] W. F. Ogilvie, P. Petoumenos, Z. Wang, and H. Leather, "Minimizing the cost of iterative compilation with active learning," in *2017 IEEE/ACM international symposium on code generation and optimization (CGO)*. IEEE, 2017, pp. 245–256.

[15] S. Sondur and K. Kant, "Performance health index for complex cyber infrastructures," *ACM ToMPECS*, vol. 7, no. 1, pp. 1–32, March 2022.

[16] S. Sondur, et al., "Optimal configuration of high performance systems," in *Conf. HPCS*, 2020.

[17] A. Fogel, et al., "A general approach to network configuration analysis," in *Symp. NSDI*, 2015, pp. 469–483.

[18] Q. Wang, et al., "Optimizing n-tier application scalability in the cloud: A study of soft resource allocation," *ACM TOMPECS*, vol. 4, no. 2, pp. 1–27, 2019.

[19] J. Guo, et al., "Variability-aware performance prediction: A statistical learning approach," in *Conf. ASE*. IEEE, 2013, pp. 301–311.

[20] H. Ha, et al., "Deepperf: performance prediction for configurable software with deep sparse neural network," in *IEEE/ACM ICSE*. IEEE, 2019, pp. 1095–1106.

[21] N. M. Koushki, S. Sondur, and K. Kant, "Improving system configurations using domain knowledge assisted semi-supervised learning," *Conference on Network and Service Management (CNSM)*, 2023.

[22] M. S. Sorower, "A literature survey on algorithms for multi-label learning," *OSU, Corvallis*, vol. 18, no. 1, p. 25, 2010.

[23] D. Yarowsky, "Unsupervised word sense disambiguation rivaling sl methods," in *ACL*, 1995, pp. 189–196.

[24] Z. Wu, et al., "Hysad: A semi-supervised hybrid shilling attack detector for trustworthy product recommendation," in *ACM SIGKDD*, 2012, pp. 985–993.

[25] Z.-H. Zhou and M. Li, "Semi-supervised learning by disagreement," *KAIS*, vol. 24, pp. 415–439, 2010.

[26] V. Nair, et al., "Finding faster configurations using flash," *IEEE Trans. on Soft. Eng.*, vol. 46, no. 7, pp. 794–811, 2018.

[27] ——, "Using bad learners to find good configurations," in *Proc. of FSE*, 2017, pp. 257–267.

[28] R. Krishna, et al., "Conex: Efficient exploration of big-data system configurations for better performance," *IEEE Trans. on Software Eng.*, 2020.

[29] N. Mohammadi Koushki, et al., "Automated configuration for agile software environments," in *CLOUD*. IEEE, 2022, pp. 511–521.

[30] I. Kopanas, et al., "The role of domain knowledge in a large scale data mining project," in *Methods & App. of Artif. Intell.* Springer, 2002, pp. 288–299.

[31] D. Kerrigan, et al., "A survey of domain knowledge elicitation in applied machine learning," *Multimodal Technologies and Interaction*, vol. 5, no. 12, p. 73, 2021.

[32] T. Dash, et al., "A review of some techniques for inclusion of domain-knowledge into deep neural networks," *Scientific Reports*, vol. 12, no. 1, p. 1040, 2022.

[33] S. Zhang, et al., "How to invest my time: Lessons from human-in-the-loop entity extraction," in *SIGKDD*, 2019.

[34] E. Schubert, et al., "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM TODS*, vol. 42, no. 3, pp. 1–21, 2017.

[35] H. Kriegel, et al., "Density-based clustering," *WIREs Data*, vol. 1, no. 3, pp. 231–240, 2011.

[36] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, 1987.

[37] S. Sondur and K. Kant, "Towards automated configuration of cloud storage gateways: A data driven approach," in *CLOUD*. Springer, 2019, pp. 192–207.

[38] Y. Wang, et al., "Predicting new workload or cpu performance by analyzing public datasets," *ACM TACO*, vol. 15, no. 4, pp. 1–21, 2019.

[39] A. Iosup, et al., "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[40] P. Soumplis, et al., "Resource Allocation Challenges in the Cloud and Edge Continuum," in *ACINC*. Springer, 2022, pp. 443–464.

[41] D. Montgomery, et al., *Introduction to linear regression analysis*. JWS, 2021.

[42] L. Breiman, "Random forests," *ML*, vol. 45, pp. 5–32, 2001.

[43] A. Natekin, et al., "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.