Evaluating Anomaly Detection Algorithms: The Role of Hyperparameters and Standardized Benchmarks

Simon Klüttermann
TU Dortmund University
Dortmund, Germany
Simon.Kluettermann@cs.tu-dortmund.de

Shubham Gupta
TU Dortmund University
Dortmund, Germany
shubham.gupta@tu-dortmund.de

Emmanuel Müller
TU Dortmund University
Dortmund, Germany
emmanuel.mueller@cs.tu-dortmund.de

***Abstract*—Anomaly detection is a cornerstone of machine learning with applications spanning healthcare, fraud detection, and scientific discovery. Despite extensive research, fair benchmarking remains a significant challenge due to the unsupervised nature of anomaly detection. Hyperparameter selection, a crucial determinant of algorithm performance, is often overlooked or biased, leading to inflated or misleading results. Current practices, including reliance on default configurations, random choices, or limited optimization, hinder reproducibility and impede progress. This work presents a novel pipeline for standardized hyperparameter optimization in anomaly detection. Leveraging a curated collection of nearly 500 datasets, the largest of its kind, our approach systematically optimizes over 80 hyperparameters for 13 widely used anomaly detection algorithms. Our comparison reveals that the performance variance from hyperparameters often surpasses inter-algorithm differences, emphasizing the need for hyperparameter-specific evaluations. We establish a reproducible foundation for anomaly detection research by providing open-access datasets and code. Our findings not only challenge existing evaluation norms but also pave the way for more robust and reliable comparisons toward better anomaly detection research.**

***Index Terms*—anomaly detection, hyperparameter optimization, benchmarking**

I. Introduction

Anomaly detection is a diverse and widely applicable field, with applications ranging from healthcare [1], [2] and fraud detection [3], [4] to scientific research [5], [6]. The breadth of these applications has driven the development of numerous algorithms, highlighting the critical importance of benchmarking to evaluate and compare their performance. Benchmarking serves two key purposes: helping practitioners select suitable algorithms [7]–[9] and assessing the value of newly proposed methods, which are typically judged by their performance relative to existing approaches.

However, anomaly detection poses unique challenges due to its unsupervised nature. Although unsupervised algorithms are designed to function without labeled data, evaluating their performance still requires labeled data. Previous research has demonstrated that even limited access to labeled anomalies can significantly improve an algorithm's performance [7]. This introduces a potential bias when such supervised feedback is used to guide algorithm design choices like hyperparameters. Consequently, it becomes challenging to determine whether such choices were made impartially or specifically optimized to favor a proposed algorithm, which might penalize such algorithms that do not use supervised feedback to optimize their hyperparameters.

A promising solution to these challenges is standardization. By standardizing evaluation methodologies, the reliability of performance metrics improves, and researchers benefit from reduced effort in conducting evaluations. ADBench [8], for instance, has proposed a standardized set of 121 datasets for anomaly detection research. This initiative has gained traction in the community, mitigating the risk of cherry-picking datasets where a particular algorithm excels. Moreover, it enables more robust evaluations, as few researchers independently would compile such extensive datasets.

Despite the progress in dataset standardization, hyperparameter selection remains a critical and underexplored source of variability in anomaly detection research. As our results show, hyperparameters can be deliberately optimized to favor one algorithm over others, even when their average performance is comparable. This practice introduces significant bias, potentially distorting conclusions drawn from benchmarks.

A fair comparison of algorithms in an unsupervised setting presents additional challenges. Although hyperparameters can be optimized on specific benchmarking datasets, their performance often fails to generalize to other datasets, where thanks to a lack of labeled data, unsupervised optimization is inherently infeasible. Consequently, many studies [9] default to using predefined hyperparameters from the original papers or popular libraries such as PyOD [10]. However, as we demonstrate, the impact of suboptimal hyperparameters can rival or exceed the differences between algorithms, thus constraining the field's progress.

In this paper, we propose a novel pipeline for selecting anomaly detection hyperparameters. Our approach optimizes hyperparameters using one set of anomaly detection datasets

This work was supported by the Lamarr-Institute for ML and AI, the Research Center Trustworthy Data Science and Security, the Federal Ministry of Education and Research of Germany and the German federal state of NRW. The Linux HPC cluster at TU Dortmund University, a project of the German Research Foundation, provided the computing power.

and evaluates performance on a disjoint set of datasets to ensure generalizability. To support this, we curate and publish the most extensive anomaly detection dataset collection to date, comprising thousands of datasets, of which 498 are used in this study because of computational limitations. We optimize over 80 hyperparameters across 13 widely used anomaly detection algorithms, including classical and more modern deep learning approaches with complex hyperparameter spaces.

To enhance reproducibility and support future research, we make our code and our evaluation results publicly available at github.com/psorus/Hyperparam4Anomaly.

II. Related Work

A. Anomaly Detection

Anomaly detection, defined as the identification of rare or exceptional objects, is a longstanding area of research [11]. Despite decades of progress, no universally optimal solution for reliably identifying anomalies exists. This is primarily due to the diverse challenges posed by the field. These challenges include a wide variety of applications [12], [13], each with distinct requirements, as well as complications arising from the scarcity of labeled anomalies. The lack of labels significantly hinders the optimization of anomaly detection algorithms, as feedback for fine-tuning a given setup is unavailable.

This issue is particularly critical in the context of hyperparameter optimization. Without labeled anomalies, it is impractical to systematically optimize hyperparameters, despite their substantial impact on algorithm performance. Consequently, while numerous anomaly detection algorithms exist, along with studies that compare their effectiveness [7], [9], [14], these comparisons are inherently limited. They must often assume the use of suboptimal hyperparameters, which can skew performance evaluations.

The most practical approach to date has been to rely on expert-selected hyperparameters during algorithm implementation, often guided by recommendations in the original research papers. For example, the popular anomaly detection library PyOD [10] adopts this approach by using default hyperparameters based on the original papers. However, as we will demonstrate, these default settings are far from optimal. In this paper, we address this gap by systematically optimizing hyperparameters for unsupervised anomaly detection and proposing improved configurations that significantly enhance anomaly detection performance.

B. Hyperparameter Optimization

Hyperparameter optimization is a crucial component in the development and performance tuning of machine learning models. Although extensively studied in supervised learning, leading to advancements such as AutoML frameworks [15] like AutoSklearn [16], this remains an underexplored area in unsupervised tasks like anomaly detection. AutoML methods typically optimize a well-defined metric, such as accuracy, that quantifies model performance. However, in the absence of labeled data and an unsupervised quality metric for anomaly detection, most existing methods are inapplicable.

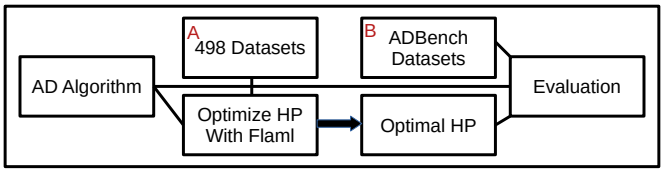

Fig. 1. Overview of our methodology. We optimize the hyperparameters of an algorithm on a large set of anomaly detection datasets and evaluate them on the unrelated ADBench datasets.

For unsupervised tasks, one viable strategy is one-shot optimization, where an optimal set of hyperparameters is suggested outright rather than iteratively optimized. MetaOD [17] represents an approach in this direction, using neural networks to recommend the best combination of hyperparameters and algorithms from a limited predefined set for a specific task. While promising, this method has notable limitations. Its complexity, restricted hyperparameter search space, and reliance on retraining for new algorithms make it less suitable for comprehensive benchmarking studies. Furthermore, its adaptability to new algorithms is constrained by the need for extensive retraining.

In contrast, we advocate for a simpler and more versatile approach: identifying a single robust set of hyperparameters for each algorithm. This approach offers several advantages: it simplifies the benchmarking process, minimizes the risk of overfitting hyperparameters to specific datasets, and allows direct optimization for individual algorithms. Although a different set of hyperparameters might perform better on specific datasets, a singular configuration provides a reproducible and broadly applicable standard for evaluation and is significantly easier to use.

Previous studies have explored hyperparameter optimization for specific algorithms. For example, there is work on optimizing the KNN-based anomaly detection algorithm [18] and selecting the kernel function for OCSVM [19]. However, these studies focus on a small number of hyperparameters or specific algorithms. In contrast, our work systematically explores over 80 hyperparameters spanning 13 widely used algorithms, providing a more comprehensive perspective on hyperparameter optimization for anomaly detection.

III. Methodology

Our methodology is summarized in Figure 1. We employ two sets of datasets: one set (**A**) is used to optimize hyperparameters, and the other set (**B**) is used to evaluate the performance of these hyperparameters. To optimize hyperparameters on set **A**, we employ `flaml` [20], a lightweight framework that supports custom evaluation functions for hyperparameter optimization. The optimization objective we choose is to maximize the average AUC-ROC across all datasets in set **A**.

Some hyperparameter configurations may be erroneous on certain datasets (e.g., a learning rate that is too high may cause a neural network to produce NaN weights). To address this, we assign a score of -1 to the AUC-ROC of any dataset where an algorithm fails with a given hyperparameter configuration.

Since `flaml` uses an evolutionary approach to optimization, this feedback helps it avoid non-functional configurations in subsequent iterations.

We allocate ten CPU cores for each algorithm's hyperparameter optimization, running for approximately one week of CPU time per algorithm to ensure a fair comparison. The hyperparameter search space is defined in Section IV, with a focus on incorporating as many relevant hyperparameters as possible, alongside a wide but reasonable range for each. Categorical variables (e.g., activation functions in neural networks) are handled separately, while numeric values are sampled uniformly or logarithmically, depending on the parameter. Logarithmic sampling is particularly beneficial for parameters like the number of training epochs, where differences are more pronounced at smaller scales (e.g., 1 vs. 2 epochs) than at larger scales (e.g., 100 vs. 101 epochs).

`flaml` prioritizes evaluating faster configurations early in the optimization process, which introduces a slight bias toward speedier hyperparameter settings. We consider this a feature rather than a limitation, as it aligns with our goal of identifying efficient and effective configurations.

After completing the optimization, we evaluate the optimized hyperparameters on the 121 datasets proposed by ADBench [8] as set B. This evaluation compares the optimized performance against the performance achieved using default parameters. For default settings, we rely on the parameters provided by the respective implementations.

A. Datasets

For our analysis, we require a large and diverse set of anomaly detection datasets to ensure that our results effectively generalize to new datasets. Moreover, these datasets must be independent of those used for evaluation (ADBench), as reusing evaluation datasets during optimization could unfairly bias the results. We prioritize real-world datasets over synthetic ones, as the latter often introduce human biases in their creation. However, constructing a substantial collection of real-world datasets is challenging and costly, given the rarity of anomalies and the fact that most widely used datasets are already part of ADBench.

To address this, we generate new anomaly detection datasets from classification datasets by treating one class as normal and another class as anomalous. This approach is commonly used in the literature [21], for instance, in image-based anomaly detection using datasets such as CIFAR-10 [22], and such datasets are also part of the ADBench evaluation set. It can be argued that classification datasets exhibit different distributions compared to true anomaly detection datasets since anomalies may not naturally form clusters. We somewhat mitigate this by using a large and diverse set of datasets, but it would likely be possible to construct a better set of datasets given unlimited resources.

We retrieve datasets using the Kaggle API [23], resulting in an initial collection of approximately $10,000$ datasets. The datasets range from 36 to 88 features and 121 to 566,602 samples. These are filtered to retain only classification datasets with tabular data. To transform these into anomaly detection datasets, we preprocess them as follows:

- Remove string or categorical features with more than 10 unique values, and one-hot encode the remaining categorical features.
- Handle missing values by removing samples with rare missing values ($< 1\%$), and otherwise removing the affected features.
- Split datasets based on their last categorical feature, treating one group as the normal class and each remaining group as anomalous.

After preprocessing, we validate each generated dataset by ensuring it achieves an AUC-ROC score of at least 0.55 using a simple isolation forest [24], guaranteeing that each dataset contains meaningful information. This process results in $3,174$ datasets of various sizes.

For hyperparameter optimization, we further filter the datasets:

- Restrict to datasets with no more than 100 features and 1 million samples.
- To avoid redundancy, select only one random subset per original dataset when multiple subsets are generated.

This yields a final pool of 498 datasets for hyperparameter optimization. While this large number of datasets is useful to generate very general results, using more datasets also limits how many different hyperparameter configurations can be evaluated.

To investigate this trade-off, we conduct our optimizations twice: once using all 498 datasets (**L**arge) and once using only the 170 datasets with a size smaller than 100KB (**S**mall).

All datasets are available through a python library (pip install kyano).

IV. Results

This section presents the outcomes of our hyperparameter optimization strategy. We detail the optimized hyperparameter configurations for the 13 analyzed algorithms and provide a comprehensive statistical comparison of their performance both before and after optimization. By systematically evaluating the impact of hyperparameter tuning, we highlight its critical role in improving algorithmic efficacy and ensuring fair benchmarking practices.

First, we summarize the performance of both the baseline hyperparameter set and the optimized sets for the small and large datasets across all algorithms in Figure 2. For clarity, we highlight hyperparameter sets that outperform the baseline in green and those that perform worse in red. Additionally, columns that do not show statistically significant differences, as determined by a Wilcoxon-Friedman test [25], are struck through.

In total, 14 out of 26 hyperparameter sets significantly outperform the baseline, while an additional 6 sets show improvement, albeit without statistical significance. Furthermore, on 12 out of the 13 algorithms, at least one optimized hyperparameter set outperforms the baseline, and on 8, both

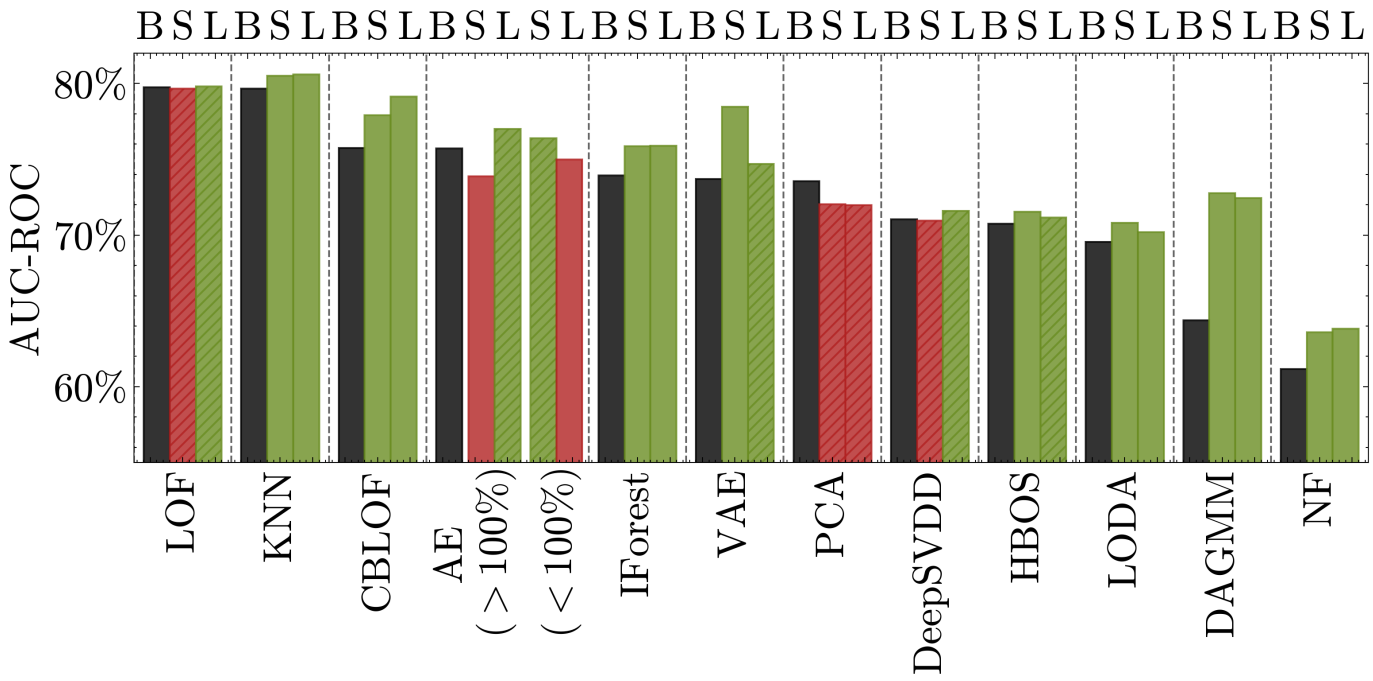

Fig. 2. Comparison of average performance before and after optimization. Each algorithm is represented by three columns corresponding to the **B**ase performance, and the performance after the **S**mall and **L**arge optimization loops. Hyperparameter configurations that improve upon the base performance are highlighted in green, while those that do not are marked in red. Additionally, a Wilcoxon-Friedman test is conducted, and columns that do not exhibit statistically significant differences to the baseline performance at $p = 0.05$ are struck through.

optimized sets achieve this. Unfortunately, the PCA algorithm does not benefit from hyperparameter optimization, a topic we will discuss further below, as the algorithm also behaves quite strangely overall. While optimization is not perfect for every algorithm, our optimization generally improves an algorithm's performance. The three highest observed performances were achieved after optimization, and the differences between hyperparameter sets often exceed those between algorithms, even when comparing fundamentally different methods. For example, the improvement of IForest and VAE is drastically larger than the difference between the two, underscoring the importance of hyperparameter optimization over algorithm development in unsupervised learning tasks.

Consistent with recent benchmark studies [8], [26], we find that classical anomaly detection algorithms, such as LOF, KNN, and CBLOF, outperform more recent deep learning approaches when evaluated with baseline parameters. Interestingly, this gap diminishes when considering optimized hyperparameters, with three variants of Autoencoders outperforming CBLOF. This trend is expected, as deep learning algorithms typically involve many more hyperparameters that require tuning and have less theoretical research to guide the choice of optimal values [18]. However, the larger number of hyperparameters also results in less efficient optimization. In this study, we allocated an equal amount of optimization time for each algorithm to ensure fairness and to limit computational demands. However, since neural network-based algorithms are often slower, this results in fewer hyperparameter combinations being considered here.

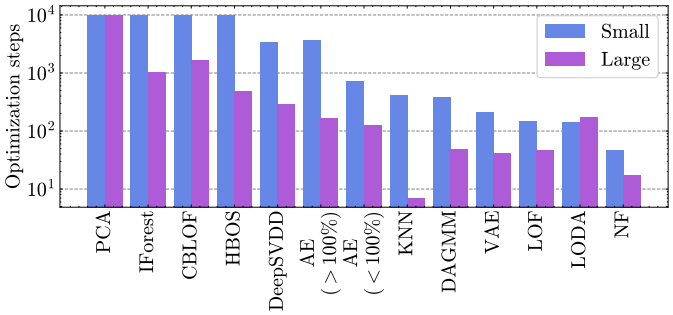

Fig. 3. Number of hyperparameter combinations explored during optimization. Due to the imposed maximum time cost of one week, a smaller number of optimization steps corresponds to a slower algorithm. Additionally, a maximum of 10,000 optimization steps was set, as visible with the first four algorithms.

The number of hyperparameter combinations evaluated by each algorithm is visualized in Figure 3. While the hyperparameter space of some algorithms, such as PCA and IForest, is well explored, others, like NF (Normalizing Flows) and SOD (Subspace Outlier Detection), are approximately 100 times slower, leading to less thorough exploration of their respective hyperparameter spaces. Allowing for a maximum number of optimization steps could also result in interesting results, but computational constraints make this difficult.

Next, we discuss the optimized hyperparameters for each algorithm, beginning with LOF [27], as shown in Table I. The performance of both optimization strategies is comparable

TABLE I
HYPERPARAMETERS LOF ALGORITHM

Hyperparam	Options	Base	Small	Large
n_neighbors	1 - 1000	20	5	8
p	1 - 6	2	1	1
algorithm	auto / kd_tree / brute / ball_tree	auto	ball-tree	ball-tree
leaf_size	10 - 50	30	13	10
AUC-ROC		0.7976	0.7967	0.798

TABLE II
HYPERPARAMETERS KNN ALGORITHM

Hyperparam	Options	Base	Small	Large
n_neighbors	1 - 1000	5	1	1
method	mean / max / median	max	max	max
p	1 - 6	2	5	1
AUC-ROC		0.7967	0.8052	0.8061

to the base performance. However, the optimized hyperparameters exhibit notable similarity, with a slightly reduced number of neighbors, a Manhattan distance metric, the Ball Tree algorithm, and a lower value for the leaf_size parameter.

For KNN [28], as shown in Table II, the optimized hyperparameters are quite similar across both optimization strategies, particularly for the number of neighbors and the method used. Notably, the number of neighbors aligns with theoretical research [18] for uncontaminated training sets (it is important to note that we use ADBench default parameters as baselines). However, there is some discrepancy between the optimizations regarding the choice of distance metric (p). Despite this, the performance remains largely unaffected, with both optimizations outperforming the baseline. The Large optimization variant overall achieves the highest average ADBench anomaly score observed in this study.

Continuing with CBLOF [29], as shown in Table III, most of the optimized parameters also show consistent results across both optimization strategies. A smaller β value than the baseline is preferable, and increasing the number of clusters also improves performance. This second observation is perhaps not surprising, as with a high number of clusters, possibly approaching the number of samples, the cluster distances calculated by CBLOF become very similar to those used in LOF and KNN, which, as shown in Figure 2, are more effective algorithms for the tasks studied here.

Our first deep learning algorithm, the Autoencoder [30], presented in Table IV, involves significantly more hyperparameters than the previously studied algorithms. This likely accounts for the considerable variation in optimized hyperparameters. Additionally, inspired by a paper [31], we consider two separate cases: an undercomplete autoencoder, where we restrict the latent size to be smaller than the number of features, and an overcomplete autoencoder, where the latent size needs to exceed the number of features. In both cases, the latent size is represented as a factor relative to the number of features. Since the ADBench implementation does not support the overcomplete case, we implemented our own version. We use here the abbreviation "(act)" to refer to a set of common activation functions (ReLU, tanh, sigmoid, softmax, linear).

TABLE III
HYPERPARAMETERS CBLOF ALGORITHM

Hyperparam	Options	Base	Small	Large
n_clusters	1 - 100	8	87	99
α	0.1 - 0.9	0.9	0.709	0.9
β	1.0 - 5.0	5	1.4782	1.4395
use_weights	✓/ ✗	✗	✗	✗
AUC-ROC		0.7576	0.779	0.7914

Although not every hyperparameter set outperforms the baseline, and significant disagreement exists between the optimized parameters, some conclusions can still be drawn from the optimization process.

First, it remains inconclusive whether an overcomplete or undercomplete autoencoder is preferable, supporting the claim made in Reference [31]. The best and worst performance are both observed with an undercomplete autoencoder, while the overcomplete autoencoder tends to be slightly faster to evaluate (see Figure 3). In both cases, two dense layers for both the encoder and decoder appear optimal, and the training should be extended well beyond the ten epochs suggested by the ADBench parameters, although the improvement is marginal. Additionally, regularization does not seem beneficial.

The Isolation Forest [24], as shown in Table V, appears to have a high degree of agreement between the optimized hyperparameters, with the number of estimators being roughly three times higher as the only notable change. This raises the question: why are more estimators not considered? It is well-established [32], [33] that for unsupervised ensembles like Isolation Forest, more submodels generally lead to better performance. During optimization, similar hyperparameter configurations with more submodels have been considered, and they would slightly increase the ADBench performance, as shown in the last row of Table V.

We believe the observed results are due to overfitting of our optimization algorithm. Since the Isolation Forest is non-deterministic, performance can vary slightly depending on random choices. Although there are only a few parameters to optimize, each leads to a new evaluation with a different random seed, allowing for slightly better or worse-performing runs. Thus, increasing the number of submodels improves performance, albeit with diminishing returns, while also reducing uncertainty and thus allowing the optimizer to extract performance from random changes less efficiently. A smaller number of estimators represents the optimal trade-off between the benefits of larger ensembles and the potential overfitting caused by the random seed. This suggests that maybe a smaller number of optimization steps might have been beneficial. Nevertheless, our results also indicate that the overfitting effects are minimal.

The results for PCA [34], presented in Table VI, are intriguing. The optimized performance is actually worse than the baseline configuration. However, the baseline parameters themselves seem questionable: a PCA model, which has access

TABLE IV
HYPERPARAMETERS AE ALGORITHM

Hyperparam latent_size	Options	Base	Small $< 100\%$	Large $< 100\%$	Small $> 100\%$	Large $< 100\%$
latent_size_factor	0.1 - 1.0 - 10.0	(32)	0.2041	0.463	3.1757	1.5514
layers	1 - 7	3	1	2	2	1
batch_size	16 - 128	32	23	29	123	103
learning_rate	1e-5 - 0.01	1e-3	3.92e-3	2.16e-3	7.55e-3	2.07e-3
epochs	100 - 500	10	496	141	191	157
encoder_activation	(act)	relu	tanh	tanh	relu	relu
decoder_activation	(act)	relu	softmax	relu	sigmoid	relu
output_activation	(act)	linear	linear	linear	linear	linear
encoder_dropout	0.05 - 0.99	None	0.05	0.11	0.05	0.05
decoder_dropout	0.05 - 0.99	None	0.99	0.2155	0.05	0.4799
evaluation_metric	1 - 6	2	2	4	1	4
loss	mse / bce	mse	mse	mse	mse	mse
regularisation	l1/l2	l1	l2	l2	l2	l2
regularizer	0.0001 - 0.5	0	6.29e-4	1e-4	1e-4	2.08e-4
AUC-ROC		0.7572	0.7387	0.7700	0.7640	0.7500

TABLE V
HYPERPARAMETERS IFOR ALGORITHM

Hyperparam	Options	Base	Small	Large
n_estimators	1 - 1000	100	356	230
bootstrap	✓/ ✗	✗	✗	✗
max_features	0.1 - 1.0	1.0	0.9699	0.831
max_samples	0.1 - 1.0	auto	1.0	1.0
AUC-ROC		0.7393	0.7586	0.7591
AUC ($n = 1k$)		0.7412	0.7598	0.7592

TABLE VI
HYPERPARAMETERS PCA ALGORITHM

Hyperparam	Options	Base	Small	Large
n_components	0.1 - 1.0	1.0	0.1523	0.1661
n_selected	0.1 - 1.0	1.0	0.3558	0.3877
AUC-ROC		0.7356	0.7205	0.7198

to all features and utilizes the full reconstruction, should theoretically achieve a perfect reconstruction, leading to a reconstruction error of 0 when used as an anomaly score. The performance of PCA can vary significantly depending on the fraction of features selected. The distribution of features differs between sets A and B. Thus, the optimal fraction may shift, leading to suboptimal results. We do not have a satisfactory explanation for this unexpected performance, but it may be that, similar to CBLOF, this limitation becomes comparable to the performance of another, more effective algorithm.

DeepSVDD [21], as presented in Table VII, shares many hyperparameters with the previously discussed autoencoder. In addition to the same shortcut "act" for activation functions, we also introduce a shortcut for the number of hidden neurons, denoted as "neurons," which we optimize. To ensure reasonable optimization time, we restrict the possible neural network architectures to a fixed set of configurations: $20-20$, $10-10$, $64-32$, $20-10-5$, $20-10-3$, $30-20-10-3$, $128-64-32$, and $20-10-3-10-20$. These relatively small configurations were chosen to maintain optimization efficiency and to align with the typically low-dimensional datasets used in this study. Upon reviewing the results, it appears that the

TABLE VII
HYPERPARAMETERS DEEPSVDD ALGORITHM

Hyperparam	Options	Base	Small	Large
batch_size	16 - 128	32	25	91
epochs	50 - 200	100	51	109
dropout_rate	0.05 - 1	0.2	0.05	0.05
h._neurons	(neurons)	64-32	64-32	20-10-3
h._act	(act)	relu	relu	relu
output_act	(act)	sigmoid	linear	sigmoid
l2_regularizer	1e-4 - 0.5	0.1	8.64e-4	7.44e-3
optimizer	sgd / adam	adam	adam	adam
preprocessing	✓/ ✗	✓	✗	✓
use_ae	✓/ ✗	✗	✗	✓
AUC-ROC		0.7103	0.7161	0.7095

TABLE VIII
HYPERPARAMETERS LODA ALGORITHM

Hyperparam	Options	Base	Small	Large
n_bins	auto / 5 / 10 / 20	10	10	10
n_random_cuts	10 - 1000	100	541	726
AUC-ROC		0.6954	0.708	0.7019

baseline parameters are already quite well-chosen, with the only notable differences being a lower dropout rate and regularization rate. However, these adjustments do not significantly affect the overall performance.

The LODA [35] algorithm, as shown in Table VIII, has two primary parameters. While the number of bins appears to be well-chosen, increasing the number of random cuts slightly enhances performance.

The HBOS [36] algorithm in Table IX behaves similarly, with a smaller alpha and tol(erance) value leading to a slight

TABLE IX
HYPERPARAMETERS HBOS ALGORITHM

Hyperparam	Options	Base	Small	Large
alpha	0.01 - 1.0	0.1	0.01	0.01
n_bins	0 / 5 / 10 / 20	10	5	20
tol	0.1 - 1.0	0.5	0.1	0.1016
AUC-ROC		0.7076	0.7154	0.7116

TABLE X
HYPERPARAMETERS NF ALGORITHM

Hyperparam	Options	Base	Small	Large
K	2 - 100	10	15	15
batch_size	16 - 128	64	16	16
epochs	10 - 500	200	499	165
lr	1e-05 - 0.01	2e-3	4.05e-4	2.29e-4
AUC-ROC		0.6116	0.6359	0.6382

TABLE XI
HYPERPARAMETERS VAE ALGORITHM

Hyperparam	Options	Base	Small	Large
batch_size	16 - 128	32	18	26
4 dropout_rate	0.05 - 0.99	0.2	0.0537	0.0593
epochs	100 - 500	100	266	142
hidden_act	(act)	relu	relu	tanh
l2_regularizer	0.0001 - 0.5	0.1	0.5	0.1012
latent_dim	0.1 - 10.0	(2)	1.2028	6.6883
num_features	1 - 100	4	49	2
output_act	(act)	sigmoid	tanh	tanh
AUC-ROC		0.7371	0.7846	0.7471

performance improvement.

The Normalizing Flow [37] algorithm, presented in Table X, is less straightforward. Despite being a neural network-based method, the constraints of normalizing flows limit the number of tunable parameters. In our experiments, adjustments such as a lower learning rate and smaller batch size result in more careful training, which, when combined with a more complex distribution (higher K), leads to a modest performance improvement. However, the performance is quite low when compared to other algorithms, aligning with some theoretical findings limiting the value of normalizing flows for such tasks [38].

The Variational Autoencoder [39], shown in Table XI, exhibits one of the largest performance improvements after optimization. The difference of 4.75% between the Small and Base versions is notably larger than the difference between LOF and AE, our best and fourth-best algorithms. This improvement could be attributed to the fact that the base version uses a fixed latent dimension of 2, whereas our approach employs a latent dimension relative to the number of features, similar to the autoencoder optimization. While we no longer distinguish between overcomplete and undercomplete versions, the results clearly indicate that an overcomplete version is preferable. Moreover, our optimization suggests a lower dropout rate, a significantly increased number of features, and strong regularization. We believe that further investigation is warranted, as this relatively strange set of hyperparameters leads to the highest performance among all deep learning algorithms.

The largest improvement of 8.38% is observed for the DAGMM [40] algorithm, as shown in Table XII. However, the performance of DAGMM is less remarkable overall, as the baseline performance is not particularly impressive. Similar to the VAE, this may be due to the suboptimal choice of baseline parameters, with nearly every hyperparameter undergoing drastic changes after optimization.

TABLE XII
HYPERPARAMETERS DAGMM ALGORITHM

Hyperparam	Options	Base	Small	Large
batch_size	16 - 128	256	35	53
lambda_cov	0.01 - 1.0	0.005	0.2004	0.1617
lambda_energy	0.01 - 1.0	0.1	0.0662	0.01
latent_dim	1 - 25	1	7	8
lr	1e-5 - 0.01	1e-4	2.69e-3	1.42e-4
lr_milestones	0 - 1	(50)	61.93%	92.66%
n_gmm	1 - 25	4	6	24
num_epochs	100 - 500	200	107	142
patience	5 - 200	50	35	35
AUC-ROC		0.6438	0.7276	0.7243

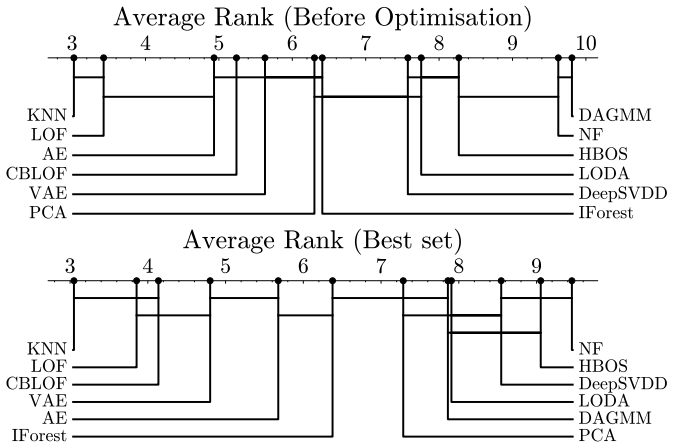

Fig. 4. Critical difference plot on ADBench data comparing the algorithms used in this study. The top plot shows performance with baseline parameters, and the bottom plot shows performance with the best set of hyperparameters found in this paper.

With these optimized parameters, we can now compare the algorithms more fairly. To do this, we use critical difference plots to visualize the results of a Wilcoxon test [25]. We consider p-values below $p \leq 5\%$, after applying the Bonferroni-Holm correction [41], as significant.

Figure 4 shows two different variations. The top plot compares the performance of the algorithms with baseline parameters, while the bottom plot compares performance using the best set of hyperparameters identified in this study. With these optimized parameters, we observe a notable change in the ordering of the algorithms, with some, such as DAGMM and VAE, experiencing significant shifts in their rankings.

V. CONCLUSION

In this paper, we introduce a novel and effective methodology for optimizing hyperparameters in unsupervised anomaly detection. We apply this methodology to 13 commonly used anomaly detection algorithms and observe performance improvements in the majority of cases.

Particularly for deep learning algorithms, we see notable gains when using the optimal hyperparameters, underscoring the importance of careful hyperparameter tuning. Furthermore, our work provides valuable insights into how these algorithms

can be better utilized and, in some cases, might serve as a foundation for the development of new algorithms.

We publish both our code as well as our evaluation results to hopefully make hyperparameter-fair comparisons more common in anomaly detection research.

REFERENCES

[1] C. Medrano, R. Igual, I. Plaza, and M. Castro, "Detecting falls as novelties in acceleration patterns acquired with smartphones," *PLoS ONE*, vol. 9, p. None, 2014.

[2] J. Zhang, Y. Xie, Y. Li, C. Shen, and Y. Xia, "Covid-19 screening on chest x-ray images using deep learning based anomaly detection," *ArXiv*, vol. abs/2003.12338, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:214693235

[3] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: a review of anomaly detection techniques and recent advances," *Expert systems With applications*, vol. 193, 2022.

[4] M. Zhang, R. Alvarez, and I. Levin, "Election forensics: Using machine learning and synthetic data for possible election anomaly detection," *PLoS ONE*, vol. 14, p. None, 2019.

[5] V. Mikuni, B. Nachman, and D. Shih, "Online-compatible unsupervised nonresonant anomaly detection," *Phys. Rev. D*, vol. 105, p. 055006, Mar 2022. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.105.055006

[6] M. Archana, N. M. Student, M. S. S. Pawar, and A. Prof, "Periodicity detection of outlier sequences using constraint based pattern tree with mad," *ArXiv*, vol. abs/1507.01685, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:1083169

[7] L. Ruff, J. Kauffmann, R. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. Dietterich, and K.-R. Müller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. PP, pp. 1–40, 02 2021.

[8] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," in *NeurIPS*, 2022.

[9] M. Olteanu, F. Rossi, and F. Yger, "Meta-survey on outlier and anomaly detection," *Neurocomputing*, vol. 555, p. 126634, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223007579

[10] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019. [Online]. Available: http://jmlr.org/papers/v20/19-011.html

[11] D. Hawkins, *Identification of outliers*, ser. Monographs on applied probability and statistics. London [u.a.]: Chapman and Hall, 1980.

[12] A. Robles-Durazno, N. Moradpoor, J. McWhinnie, and G. Russell, "A supervised energy monitoring-based machine learning approach for anomaly detection in a clean water supply system," *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, vol. None, pp. 1–8, 2018.

[13] F. Seraj, B. V. D. Zwaag, A. Dilo, T. Luarasi, and P. Havinga, "Roads: A road pavement monitoring system for anomaly detection using smart phones," *None*, vol. None, pp. 128–146, 2015.

[14] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Artificial Intelligence and Cognitive Science*, L. Coyle and J. Freyne, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 188–197.

[15] S. S. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, "Automl to date and beyond: Challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, pp. 1 – 36, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:235078786

[16] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28 (2015)*, 2015, pp. 2962–2970.

[17] Y. Zhao, R. A. Rossi, and L. Akoglu, "Automating outlier detection via meta-learning," 2021. [Online]. Available: https://arxiv.org/abs/2009.10606

[18] P. Cunningham and S. Delany, "k-nearest neighbour classifiers," *Mult Classif Syst*, vol. 54, 04 2007.

[19] A. Budynkov and S. Masolkin, "The problem of choosing the kernel for one-class support vector machines," vol. 78, 2017, p. 138–145.

[20] C. Wang, Q. Wu, M. Weimer, and E. E. Zhu, "Flaml: A fast and lightweight automl library," in *Fourth Conference on Machine Learning and Systems (MLSys 2021)*, April 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/flaml-a-fast-and-lightweight-automl-library/

[21] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *ICML*, 2018.

[22] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[23] "Kaggle," https://www.kaggle.com.

[24] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*, 2008.

[25] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[26] V. Livernoche, V. Jain, Y. Hezaveh, and S. Ravanbakhsh, "On diffusion modeling for anomaly detection," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=lR3rk7ysXz

[27] M. Breunig, P. Kröger, R. Ng, and J. Sander, "Lof: Identifying density-based local outliers." vol. 29, 06 2000.

[28] X. Gu, L. Akoglu, and A. Rinaldo, "Statistical analysis of nearest neighbor methods for anomaly detection." in *NeurIPS*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 10 921–10 931. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2019.html#GuAR19

[29] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9, pp. 1641–1650, 2003.

[30] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, ser. MLSDA'14. New York, NY, USA: Association for Computing Machinery, 2014, p. 4–11.

[31] B. X. Yong and A. Brintrup, "Do autoencoders need a bottleneck for anomaly detection?" *IEEE Access*, vol. 10, pp. 1–1, 01 2022.

[32] C. Aggarwal, "Outlier ensembles: Position paper," *SIGKDD Explorations*, vol. 14, pp. 49–58, 01 2012.

[33] S. Klüttermann and E. Müller, "Evaluating and comparing heterogeneous ensemble methods for unsupervised anomaly detection," in *IJCNN*, 2023.

[34] C. Callegari, L. Gazzarrini, S. Giordano, M. Pagano, and T. Pepe, "A novel pca-based network anomaly detection," 07 2011, pp. 1 – 5.

[35] T. Pevný, "Loda: Lightweight on-line detector of anomalies," *Mach. Learn.*, vol. 102, no. 2, p. 275–304, feb 2016. [Online]. Available: https://doi.org/10.1007/s10994-015-5521-0

[36] M. Goldstein and A. R. Dengel, "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm," 2012.

[37] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1530–1538.

[38] P. Kirichenko, P. Izmailov, and A. G. Wilson, "Why normalizing flows fail to detect out-of-distribution data," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[39] D. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[40] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. ki Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:51805340

[41] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979. [Online]. Available: http://www.jstor.org/stable/4615733