Journal of VLSI circuits and systems

RESEARCH ARTICLE

Journal of VLSI Circuits and Systems, ISSN: 2582-1458 Vol. 8, No. 2, 2026 (pp. 23-35)
WWW.VLSIJOURNAL.COM

Generalization-Aware Evaluation of HLS QoR Prediction: In-Distribution Performance versus Cross-Algorithm Generalization Using ForgeHLS

Ameen Shaheen*, Wael Alzyadat, Aysh Alhroob
Department of Software Engineering, Al-Zaytoonah University of Jordan, Amman, Jordan.

KEYWORDS:
Cross-Algorithm Generalization;
Evaluation Protocols;
ForgeHLS;
High-Level Synthesis;
Machine Learning;
Quality-of-Results Prediction

ARTICLE HISTORY:
Received: 07.07.2026
Revised: 10.07.2026
Accepted: 20.07.2026

DOI:
https://doi.org/10.31838/jvcs/08.02.03

Abstract

High-Level Synthesis (HLS) enables efficient FPGA-based accelerator design, but quality-of-results (QoR) exploration remains expensive because accurate latency and resource estimates typically require full synthesis flows. This has motivated the use of machine learning (ML) for early QoR prediction. However, it remains unclear whether strong performance under conventional random train/test splits truly reflects generalization to unseen algorithms. In this work, we study this question using the ForgeHLS design-level dataset. We build a reproducible multitarget QoR prediction pipeline to estimate six metrics: BRAM, LUT, DSP, FF, best-case latency, and worst-case latency. To isolate the effect of the evaluation protocol, we adopt a simple and interpretable Random Forest baseline and evaluate it under two settings: a standard random split and a stricter group split based on algorithm identity. In addition, a boosting-based baseline (XGBoost) is evaluated to verify that the observed behavior is not model-specific. We further repeat the group-based evaluation across multiple random seeds. The results show that the baseline performs strongly under random splitting, especially for BRAM and latency-related targets, but that its performance degrades sharply under group-based evaluation. Repeated group-split experiments confirm that this degradation is systematic rather than incidental. Feature importance analysis further indicates that the strongest predictive signals are closely tied to algorithm identity and source-related information. Overall, the study shows that random-split evaluation can substantially overestimate the true capability of HLS QoR predictors and underscores the importance of generalization-aware evaluation protocols for future ML-based HLS QoR prediction research.

Authors' e-mail ID: a.shaheen@zuj.edu.jo, Wael.alzyadat@zuj.edu.jo, aysh@zuj.edu.jo

Authors' ORCID ID: 0009-0003-3120-1892; 0000-0002-0068-1526; 0000-0002-4653-7386

How to cite this article: Ameen Shaheen et al., Generalization-Aware Evaluation of HLS QoR Prediction: In-Distribution Performance versus Cross-Algorithm Generalization Using ForgeHLS, Journal of VLSI Circuits and System, Vol. 8, No. 2, 2026 (pp. 23-35).

Introduction

High-Level Synthesis (HLS) has become an increasingly important design methodology for FPGA-based acceleration because it raises the abstraction level from RTL to C/C++ and significantly reduces development effort [1]. However, generating high-quality designs requires extensive exploration of trade-offs between latency and clock-related metrics. This process remains expensive in practice since each candidate design must undergo synthesis and report generation before its quality of results (QoR) can be assessed. Consequently, machine learning (ML)-based QoR prediction has emerged as a promising approach for accelerating design-space exploration in HLS by estimating design metrics before executing the complete synthesis flow [2].

Recent work has advanced this area along two complementary directions. The first has focused on creating publicly available, open, and reproducible datasets for machine learning (ML) applications in HLS. For example, HLSDataset introduced one of the first public collections of FPGA designs generated through HLS, drawn from benchmark suites such as PolyBench, MachSuite, CHStone, and Rosetta [3]. HLSFactory further advanced this direction by introducing a scalable framework for constructing, aggregating, and reusing datasets in

ML-based studies of HLS [4]. Most recently, the ForgeHLS dataset significantly expanded the number and diversity of available designs, providing greater scale than previous datasets. It was developed specifically to support downstream tasks, including QoR prediction and automated optimization in HLS [5].

The second direction has focused on improving the predictive models used for QoR estimation. Recent studies include graph-based approaches that predict QoR from a hierarchical source-to-post-route perspective by modeling program structure and pragmatics using Graph Neural Networks [6]. Other studies have explored language-level modeling, demonstrating the feasibility of using transformer-based architectures to estimate resource utilization and performance directly from source code and HLS pragmas [7]. Together, these research efforts demonstrate that both high-quality datasets and robust model development are essential for building accurate HLS QoR predictors [8].

Despite this progress, a key methodological limitation remains underexplored: whether predictive performance under traditional random train/test splits truly reflects a model's ability to generalize. Because most HLS datasets contain numerous samples generated from the same underlying kernel through different combinations of pragmas and synthesis strategies, related design variants often appear in both the training and test sets, potentially leading to overly optimistic performance estimates [9]. A more stringent and representative evaluation setting is one in which a model is trained on a set of several known algorithms and then tested on algorithms that were unseen during training [10]. This distinction is particularly important for modern large-scale HLS datasets, where having abundant data does not necessarily imply abundant diversity in the underlying distribution of algorithms represented in the dataset.

This paper evaluates the differences between in-distribution QoR prediction and cross-algorithm generalization in HLS using the ForgeHLS dataset. A reproducible multitarget ML pipeline was developed to predict six QoR metrics: BRAM, LUT, DSP, FF, best-case latency, and worst-case latency. Two evaluation procedures are compared: a standard random train/test split and a stricter group-based split based on algorithm identity. Although the primary objective of this study is to evaluate QoR prediction performance, it also investigates whether the performance typically reported under conventional evaluation protocols remains robust when the test set consists entirely of algorithms unseen during training. Consequently, this work serves not only as a study of QoR prediction but also as a methodological investigation into how predictive performance should be interpreted in HLS.

The study results demonstrate a clear and consistent pattern. Under a random train/test split, the baseline model achieved high-quality performance results for many target metrics (especially latency-related). However, when the evaluation protocol was changed from a random split to a group split by *algo_name*, the model performance declined significantly for almost all targets. Furthermore, repeated group-split trials confirmed that this decline was systematic rather than incidental. Overall, these findings indicate that much of the baseline model's success in predicting HLS QoR under an in-distribution setting can be attributed to within-algorithm similarity rather than robust cross-algorithm generalization. Consequently, these observations are methodologically significant because they demonstrate that evaluating models using random sampling may lead to a misleading impression of their capabilities for predicting HLS QoR quality.

Beyond their methodological contribution, these findings also have practical implications for industrial FPGA and VLSI design automation. More reliable evaluation protocols can help developers better assess ML-based QoR predictors before deployment, reducing the risk of overly optimistic performance estimates during early design-space exploration and enabling more dependable automated HLS optimization workflows.

The main contributions of this study are as follows:

- A reproducible, multitarget QoR prediction pipeline on the ForgeHLS dataset, using practical design- and code-level features and a simple, interpretable baseline.
- A systematic evaluation of how the choice of evaluation protocol affects the interpretation of QoR prediction performance, through a direct comparison between a traditional random split and a stricter group-based split defined by algorithm identity.
- Robust empirical evidence, supported by repeated group-split experiments, demonstrating that the observed performance degradation is systematic rather than incidental.
- Additional experimental validation using a stronger boosting-based baseline (XGBoost) and a single-target setting, confirming that the generalization gap is not limited to a specific model or to the multitarget formulation.

In summary, the results show that random-split evaluation can lead to overly optimistic conclusions about model performance and that generalization-aware evaluation protocols are essential for the reliable assessment of HLS QoR predictors.

Related Work

Recent studies in data-driven optimization and automated design space exploration in HLS have increasingly moved beyond traditional manual tuning methods. One major line of research uses prior optimization knowledge to support decision-making for directive selection and improve design quality. Ferikoglou et al.[11] presented a data-driven HLS optimization method for reconfigurable accelerators, demonstrating that knowledge from previously optimized designs can be used to identify viable configuration options without relying on explicit QoR models or manually crafted search algorithms. Similarly, CollectiveHLS: Ultrafast Knowledge-Based HLS Design Optimization demonstrated how previously optimized knowledge could be utilized to improve directive selection for new design requirements [12]. The concept was later extended into a collaborative framework, CollectiveHLS: A Collaborative Approach to High-Level Synthesis Design Optimization, which introduced a reusable source-code-based optimization methodology applicable across multiple applications [13].

A second major area of research addresses the automation of pragma placement and code transformation, both of which are critical aspects of HLS optimization. The study by Pouget et al.[14] presented an automated framework for pragma placement using nonlinear programming. The authors demonstrated that selecting pragmas to satisfy latency and resource constraints can be formulated as an optimization problem rather than explored heuristically. The work by Pouget et al.[15] was later extended in Sisyphus, a framework that unifies three previously independent flows, including pragma insertion, code transformation, and tile size selection, into a single optimization flow. The results of both studies demonstrate that the primary challenge in HLS is not only predicting the QoR during synthesis but also effectively exploring the large combinatorial design space of possible pragma placements and program transformations.

A recent line of research has also investigated how Large Language Models (LLMs) and Graph Neural Networks (GNNs) could be integrated into HLS optimization frameworks. Shang and Huang [16] introduced an LLM-based pragma insertion framework that utilizes GNNs to supervise pragma generation quality, demonstrating that structural code understanding and language modeling can be combined to produce higher-quality pragmas. Similarly, Xu et al.[17] developed a hybrid that combines GNNs and LLMs to enhance evolutionary search in HLS design-space exploration. These studies reflect a growing trend toward integrating predictive modeling with search strategies rather than viewing them as two distinct stages. Together, they suggest that future HLS automation will increasingly rely on joint reasoning over prior optimization knowledge, learned search policies, and structural code understanding.

Compared with these HLS optimization studies, the present work has a narrower scope but a stronger methodological focus. It does not propose a new optimizer, pragma insertion framework, or LLM-based exploration strategy. Instead, it investigates the influence of the evaluation protocol on the assessment of QoR prediction performance. Although many recent HLS studies have reported significant predictive or optimization results, the distinction between random-split performance and generalization to previously unseen algorithms has rarely been examined as a primary research question.[11–17] Accordingly, the novelty of this work lies in treating the evaluation methodology itself as the primary object of analysis. Specifically, this study complements the existing literature by demonstrating that, even when using a practical baseline on a large, modern HLS dataset, the observed performance of QoR prediction differs significantly when the split protocol is aware of the algorithms used [18–22]. This methodological emphasis distinguishes the present study from prior work, which primarily focuses on model or optimization design.

Methodology

The entire experimental pipeline used in this study is presented in this section. Because the primary objective of this paper is to determine how well in-distribution QoRs predictions correlate with predictions that generalize across unseen algorithms, the methodology was designed both to develop practical QoR predictors and to assess how strongly their reported performance depends on the choice of evaluation protocol. Accordingly, this study uses the ForgeHLS Design-Level Dataset, formulates QoR prediction as a multitarget regression problem, and compares two train/test splitting methods: a conventional random split and a more stringent group-based split based on algorithm identity. A simple, interpretable baseline model was intentionally selected instead of a highly specialized architecture so that the effect of the evaluation protocol could be

isolated without being confounded by model complexity. This design choice aligns the methodology with the central objective of the paper, which is methodological evaluation rather than architectural innovation.

Dataset and Target Metrics

The experiments were conducted using the design-level ForgeHLS JSON file, which contains 184,717 design entries. Each entry includes 24 design-related attributes that directly represent an HLS-generated design, along with additional metadata describing both the design and the source code used to generate it. In this work, a subset of relevant features was selected and preprocessed to construct a clean tabular dataset suitable for QoR prediction. Specifically, attributes that were directly and consistently relevant to QoR prediction were selected from the complete set of design-level features.

The prediction task in this study involves six target variables: BRAM_18K, LUT, DSP, FF, Best-caseLatency, and Worst-caseLatency. Together, these variables represent the two primary QoR attributes that characterize HLS designs: resource utilization and performance-related timing. After selecting the required input and output variables and removing records with missing values, the cleaned dataset contained 184,717 samples and 15 columns. Table 1 summarizes the selected input features and prediction targets.

Problem Formulation

Let $xi \in R$ denote the encoded input representation of the $i\text{-}th$ design, and let $yi \in R^6$ denote the corresponding vector of QoR targets. The prediction problem is formulated as learning a function:

$$f : x_i \mapsto y_i \tag{1}$$

where the output vector is defined as:

$$yi = \left[y_i^{BRAM}, y_i^{LUT}, y_i^{DSP}, y_i^{FF}, y_i^{BCL}, y_i^{WCL} \right] \tag{2}$$

Table 1: Features and prediction targets used in this work

Type	Variables
Categorical input features	Part, algo_name, source_name, top_function_name
Numerical input features	TargetClockPeriod, code_length, pragma_number, is_pareto, is_kernel
Prediction targets	BRAM_18K, LUT, DSP, FF, Best-caseLatency, Worst-caseLatency

In Equation (1), $f(\cdot)$ denotes the regression model that maps each encoded design representation to a multitarget QoR estimate. In Equation (2), $y_i^{BRAM}, y_i^{LUT}, y_i^{DSP}, y_i^{FF}$ represent the resource-related targets, while y_i^{BCL}, y_i^{WCL} denote the best-case and worst-case latency values, respectively. This formulation allows us to treat QoR prediction as a structured multitarget regression problem rather than as a set of isolated single-output tasks.

Feature Construction and Preprocessing

The retained input variables represent the three aspects of each design that were identified as relevant: the design's implementation context, the identity of its algorithm, and its code-level complexity. The implementation context is described by the target FPGA design and the target clock period. The identity of the algorithm and its source provenance is captured by algo_name, source_name, and top_function_name. The code-level complexity is estimated using code_length, while the characteristics of the design space are represented by is_pareto, is_kernel, and pragma_number.

Categorical data are encoded using one-hot encoding, whereas numerical variables are passed directly to the regression model. This approach maintains the simplicity, transparency, and replicability of the baseline while still allowing the predictor to exploit design identity and source-level regularities. Most importantly, the need for expensive feature engineering processes, such as graph extraction or intermediate representation analysis, has been deliberately avoided, because the objective of this paper is not to achieve the highest possible prediction accuracy through feature engineering. Instead, the goal is to determine whether the choice of train/split affect the interpretation of QoR prediction performance.

Prediction Models

To ensure that the conclusions of this study are driven primarily by the evaluation setting rather than by model-specific complexities, a baseline family of tree-based ensemble models was employed. The primary model used throughout the analysis was a multi-output Random Forest regressor. This well-recognized regression algorithm has strong capabilities for non-linear tabular regression and is particularly effective at handling heterogeneous feature spaces [23]. In addition to Random Forest, this study evaluated Extra Trees as a comparative ensemble baseline and HistGradientBoostingRegressor as a single-target baseline [24]. Furthermore, XGBoost model was included as a stronger boosting-based

baseline to further validate the experimental findings. However, because the Random Forest model provided the most stable and interpretableable results, it was selected as the primary reference model for all comparisons and methodological analyses. Table 2 illustrates the baseline models used in the experiments.

All experiments were implemented in Python 3.13.9 using scikit-learn 1.7.2 and XGBoost 3.2.0. The experiments were conducted on an ASUS TUF Gaming F15 laptop equipped with a 13th Gen Intel Core i7-13620H processor, 32 GB of RAM, and an NVIDIA GeForce RTX 4050 Laptop GPU with 6 GB of dedicated memory.

Evaluation Protocols

The central question of this paper is whether strong predictive performance under a standard random train-test split reflects true generalization to previously unseen algorithms. To investigate this question, two evaluation protocols are employed.

In the first protocol, the data are divided using a standard random split, with 80% of the samples allocated to the training set and the remaining to the test set. This protocol reflects the conventional in-distribution evaluation setting commonly adopted in predictive modeling and enables direct comparison with the evaluation methodology used in most previous studies.

The second protocol employs a group-based split based on `algo_name`. Under this protocol, each design is assigned completely to either the training set or the testing set for each of the algorithms that it represents. This prevents algorithm-level overlap between the training and test sets and provides a stronger and practically meaningful evaluation of a model's ability to generalize to previously unseen algorithms. In other words, all design instances derived from the same algorithm are assigned exclusively to either the training or the test set, but never to both. Formally, the split satisfies the following constraint, where denotes the algorithm identification number for sample *i*.

$$g_i \in G_{train} \Rightarrow g_i \notin G_{test}, \tag{3}$$

and equivalently:

$$G_{train} \cap G_{test} = \varnothing. \tag{4}$$

In Equation (3), G_{train} and G_{test} denote the sets of algorithm identities assigned to the training and test partitions, respectively. Equation (4) explicitly states that these two sets have no overlap. This protocol resulted in 376 training algorithms and 95 test algorithms that had no commonality. This distinction is particularly important in HLS because multiple implementations of the same algorithm can be generated using different pragma combinations or synthesis strategies. Under a standard random split, such implementations could appear in both the training and test sets, leading to information leakage between the two partitions [25].

Table 2: Baseline models used in the experiments

Model	Output setting	Role in the study
Random Forest	Multitarget	Main baseline
Extra Trees	Target-wise comparison	Secondary baseline
HistGradientBoosting	Single-target comparison	Exploratory model
XGBoost	Target-wise comparison	Strong boosting-based baseline

To establish that this performance degradation under group-based evaluation is not an artifact of an unfavorable data partition, the group-split experiment was repeated across multiple random seeds, and the mean and variance of the resulting performance metrics were reported. These additional robustness checks enable us to determine whether the observed generalization gap persists consistently across multiple distinct algorithm-level groupings.

Evaluation Metrics

To quantify prediction quality, the Mean Absolute Error (MAE) [26] and the coefficient of determination (R^2) [27] were reported for each target variable. MAE is defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{5}$$

Where y_i is the ground-truth value of the *i*-th test sample, $\hat{y}_i$ is the corresponding prediction, and *n* is the number of test samples. MAE provides an interpretable absolute error scale and is therefore useful for understanding the practical magnitude of prediction error. The R^2 score is computed as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{6}$$

Where $\bar{y}_i$ denotes the mean of the true target values in the test set. The numerator measures the residual prediction error, while the denominator measures the total variance of the target. Consequently, R^2 reflects how much of the target variance is explained by the predictor and is particularly useful for comparing predictive strength across different QoR metrics.

Feature Importance Analysis

To examine the model behavior from a target perspective, an importance analysis of the features using the random forest was conducted for the Worst-Case Latency target. The purpose of this analysis was to identify the features that have the highest impact on the model's prediction process under the baseline setting, as these features are expected to contribute most significantly to the model's performance.[28] This type of analysis is often used in ensemble-based predictive modeling efforts to identify the variables that most significantly influence the model's output. It helps determine whether the observed performance is driven by transferable QoR relationships or is dominated by algorithm-specific signaling.

An overview of the overall QoR prediction workflow is shown in Figure 1. The entire process begins with the ForgeHLS JSON dataset, followed by feature selection, data cleaning, preprocessing, and data encoding. After model training, the workflow branches in to two evaluation protocols: a random split and an evaluation split based on the algorithms used. The outputs of these two branches are then used to perform robustness analyses, feature importance analyses, and generate the final tables and figures.

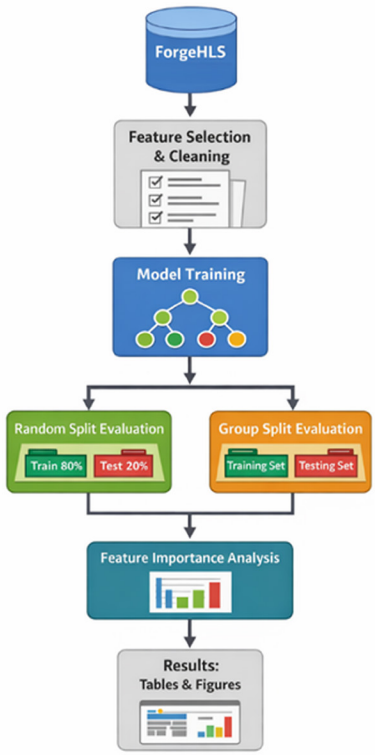

Fig. 1: Overview of the proposed QoR prediction methodology.

Algorithm 1 summarizes the full experimental procedure used in this study, from data preparation and feature encoding to random-split evaluation, algorithm-aware group-split evaluation, repeated robustness analysis, and feature-importance extraction.

Algorithm 1. QoR prediction workflow.

Input:

ForgeHLS design-level dataset D

Output:

Random-split results, group-split results, repeated group-split statistics, and feature importance

1: Load dataset D
2: Select the retained features and QoR targets
3: Remove records with missing values
4: Encode categorical features using one-hot encoding
5: Construct feature matrix X and target matrix Y
6: Train and evaluate the Random Forest model under a random split.
7: Train and evaluate the same model under a group split based on .
8: Repeat the group-split experiment across multiple seeds.
9: Aggregate the repeated results as mean ± standard deviation.
10: Extract feature importance for Worst-caseLatency.
11: Report the final tables and figures.

Methodological Rationale

The methodology in this paper intentionally focuses on clarity, reproducibility, and rigorous evaluation rather than architecture-based innovation. The objective of this study is not to develop the most advanced QoR predictor, but rather to determine whether the perceived success of HLS QoR prediction remains valid when the evaluation becomes sensitive to algorithm-level generalization. Therefore, a simple, interpretable table-based baseline with controlled preprocessing is more

appropriate than a highly optimized model, where improvements in model architecture may obscure the key methodological issue. By making this design choice, the study can identify the effects of the evaluation protocol, maintain focus on the primary research question, and provide a clear foundation for the results and discussion presented below.

Results and Discussion

In addition to presenting the empirical results of the proposed QoR prediction pipeline, this section emphasizes the central methodological question of the paper: whether strong performance under a conventional random split represents true generalization or merely overly optimistic predictions for in-distribution data. Therefore, the analysis focuses on comparing random-split evaluation with the more stringent group-based split approach using algorithm identity and determining whether the performance gap between these two methods persists across different group-based partitions of the dataset. Baseline comparisons and feature importance analyses are included to support the interpretation of the observed differences between the two evaluation methods without detracting from the primary claim of the paper. Thus, the results section serves both to present predictive performance and to demonstrate that commonly applied evaluation methods may significantly overestimate the true capabilities of HLS QoR predictors.

Performance Under Random Split

First, the proposed multi-output Random Forest baseline under an 80/20 random split was evaluated. Table 3 reports the MAE and R^2 values for all six QoR targets.

The findings reveal a clear performance imbalance among the prediction objectives. BRAM_18K and both latency objectives, Best-case Latency and Worst-case Latency, achieved extremely high R^2 values of 0.93 under the random split evaluation. The timing-related objectives demonstrated particularly high predictive accuracy, while the BRAM objective also achieved a reasonable level of accuracy. These results indicate substantial regularity between the retained code- and design-level attributes and the timing-related outputs, while also suggesting that BRAM usage is relatively stable and reproducible.

Table 3: Random-split performance of the main Random Forest baseline

Target	MAE	R^2
BRAM_18K	2.4438	0.9363
LUT	11954.5101	0.2713
DSP	97.5387	0.1556
FF	9518.0625	0.1963
Best-caseLatency	153560576.0931	0.9503
Worst-caseLatency	699180943.2904	0.9634

On the other hand, the model performs better on BRAM and latency than on LUT, DSP, and FF. Although the R^2 values for all three resource categories remain positive, the results indicate that the selected baseline features and model do not explain variations in fine-grained logic and arithmetic resource usage as effectively as they explain latency and memory block utilization. However, while the random-split results establish a strong baseline, they also demonstrate that at least part of the QoR prediction problem can be explained using the selected feature set.

Comparative Baseline Analysis

To confirm that the observed behavior is not specific to a single model but instead generalizes across different models, the primary Random Forest model was compared with two additional models: Extra Trees and HistGradientBoosting. Table 4 summarizes the results.

Random Forest and Extra Trees exhibit similar behavior across all test objectives. This similarity is particularly evident for BRAM and latency, where both models achieve nearly equivalent results. However, Extra Trees provides a slightly better BRAM predictor than Random Forest, although this improvement is minimal and does not alter the overall interpretation of the results. Neither ensemble method provides a meaningful advantage over the baseline results for LUT, DSP, and FF.

To further validate that the observed behavior is not specific to a single model remains consistent across

Table 4: Comparison of baseline models under random split

Target	Random Forest R^2	Extra Trees R^2	HistGradientBoosting R^2
BRAM 18K	0.9363	0.9365	0.5760
LUT	0.2713	0.2616	0.0938
DSP	0.1556	0.1437	0.0470
FF	0.1963	0.1848	0.0483
Best-caseLatency	0.9503	0.9442	0.8669
Worst-case Latency	0.9634	0.9629	0.9264

different model configurations, we evaluated a stronger boosting-based baseline (XGBoost) under both random and group-based evaluation protocols. The results are summarized in Tables 5 and 6.

Under the random split setting, XGBoost achieves strong predictive performance, particularly for BRAM and latency targets, with R^2 values comparable to Random Forest. However, under the group-based split, performance drops significantly across all targets. This confirms that the generalization gap is not specific to Random Forest, but is a consistent phenomenon across different model families.

Random Forest performed significantly better than the Gradient Boosting baseline when predicting all four target variables, particularly the resource-related target variables. Although it performed almost equally well as Random Forest for latency prediction, it did not match the Random Forest model in terms of predictive stability or overall performance. Therefore, Random Forest is treated as the primary reference model for the remainder of the paper. These results provide additional methodological value by supporting the conclusion that the main findings of this study are not simply the result of an unsuitable model choice, but instead reflect more general characteristics of the evaluation environment and the prediction problem being addressed.

Feature Importance Analysis

To better understand what drives the baseline predictions, we extracted feature importance scores from the Random Forest model trained for the Worst-case Latency target. The top-ranked features are listed in Table 7.

In addition to providing insight into model behavior, the feature importance analysis identifies the key features contributing to the model's predictive capability as being primarily related to algorithm identity and the algorithm's overall top-level function. A smaller proportion of the feature importance can be attributed to more general complexity indicators, such as code length. Therefore, these findings support the idea that the strong performance of the Random-Split model, particularly for the latency targets, is largely due to the algorithm-specific characteristics that enable a degree of pattern recognition. It appears that the predictor identifies both generalizable code-level patterns and specific algorithmic signatures encountered during the training phase. These factors are important to consider evaluating the new protocol for model assessment presented in the next subsection.

Table 5: XGBoost performance under random split

Target	MAE	R^2
BRAM_18K	3.0667	0.9242
LUT	14189.4951	0.1802
DSP	117.8532	0.1353
FF	11003.2744	0.1923
Best-caseLatency	167144784	0.9480
Worst-caseLatency	745726336	0.9616

Table 6: XGBoost performance under group split

Target	MAE	R^2
BRAM_18K	7.9402	−0.0007
LUT	30700.6836	−0.0244
DSP	185.6959	−0.0013
FF	16433.9492	0.0026
Best-caseLatency	304737696	0.2274
Worst-caseLatency	2464116480	0.2705

Table 7: Top feature importance values for Worst-caseLatency prediction

Rank	Feature	Importance
1	top_function_name_kernel_lu	0.211904
2	algo_name_lu	0.210586
3	algo_name_ludcmp	0.192448
4	top_function_name_kernel_ludcmp	0.185230
5	source_name_PolyBench	0.083159
6	top_function_name_kernel_cholesky	0.020130
7	algo_name_cholesky	0.018426
8	code_length	0.013797
9	algo_name_seidel-2d	0.011119
10	top_function_name_kernel_seidel_2d	0.010836

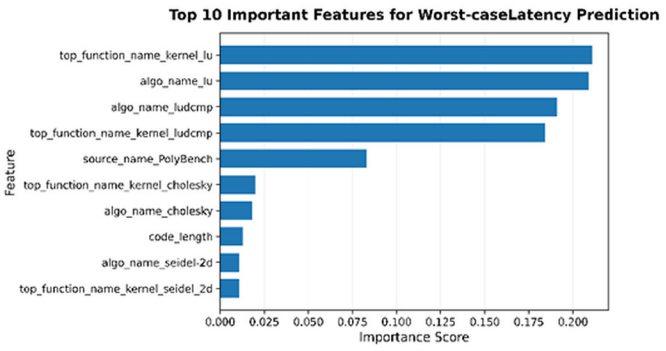

Fig. 2: Top feature importance values for Worst-case Latency prediction under the Random Forest baseline.

Random Split Versus Group Split

The primary comparison in this study is between the standard random split and the split based on *algo_name*, in which the training and test sets comprise distinct algorithm identities. The group-split protocol used in this implementation resulted in 376 algorithms in the training set and 95 algorithms in the test set, with no overlap between the two partitions. Table 8 illustrates the performance results.

The comparison with the random-split results reveals an enormous difference. While these same targets exhibited the highest predictive capabilities under the random-split protocol, they now produce the lowest R^2 values, with some being close to zero or even negative. Although the latency-based metrics remain the best performing targets under the in-distribution condition, they also have been reduced significantly under the group split. Worst-CaseLatency was still positive; however, the margin is much smaller than that observed under the random-split protocol. This result directly confirms the main hypothesis of this research: the generalization performance of a model when predicting data of a model when predicting data from previously unseen algorithms is significantly lower than its performance estimated using a random-split evaluation.

The severity of this degradation is easily interpreted through a direct visual comparison. Figure 3 contrasts the R^2 scores obtained under the random-split and group-split evaluation protocols.

Figure 3 shows a clear and systematic difference between the two evaluation methods. The random-split protocol produces high predictive performance for BRAM and latency targets, as well as moderate performance for LUT, DSP, and FF. On the other hand, under the group-split protocol, the predictor's performance declines across all evaluated targets. This indicates that much of the performance observed in the random-split evaluation is due to similarities between the training and test sets at the algorithm-family level. Once this similarity is eliminated, the predictor has difficulty making accurate predictions for algorithms outside of the set on which it was trained.

Table 8: Group-split performance of the Random Forest baseline

Target	MAE	R^2
BRAM_18K	7.3355	−0.0012
LUT	42581.9675	−0.1680
DSP	186.3864	−0.0015
FF	18589.1061	−0.0082
Best-caseLatency	309302884.2145	−0.0166
Worst-caseLatency	2934632571.3226	0.1057

The primary methodological contribution of this paper is the magnitude of the performance differences observed between the two different methods. Although it was assumed that the group-split protocol would be more challenging than the random-split protocol, the magnitude of these differences was large enough to change how the quality of the models is interpreted. A model that appears to perform well under random-split conditions, in fact, may perform poorly when predicting the performance of new algorithms. Therefore, generalization-aware evaluation protocols should not be considered optional in studies of HLS QoR prediction.

Single-Target Validation

To further examine whether the observed degradation is caused by the multitarget formulation, we conducted an additional experiment using a single-target Random Forest model for Worst-caseLatency.

Under the random-split protocol, the model achieved a high R^2 score of 0.963. However, under the group-split protocol, the performance dropped significantly, with the R^2 score decreasing to 0.103.

This result indicates that the generalization gap persists even in the single-target setting, confirming that the limitation is not caused by multitarget modeling but is instead inherent to cross-algorithm generalization.

Robustness across Multiple Group Splits

To verify that the observed generalization gap is not an artifact of a single train/test partition, the group-based

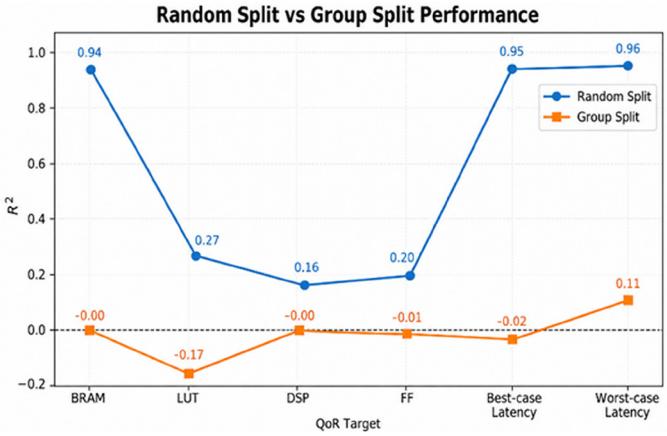

Fig. 3: Comparison of Random Forest performance under random split and group split by algorithm.

split experiment was repeated using five different random seeds. Table 9 reports the mean and standard deviation of the Random Forest results across these repeated group splits.

The group-split results are presented in a concise graphical form in Figure 4. The height of each bar represents the mean R^2 value, while the error bars represent the corresponding standard deviation. First, it can be seen that the overall average performance is at zero or below for most targets. This indicates that the degradation that occurs when algorithms are evaluated in groups and compared, as opposed to individually, does not occur due to the placement of a single algorithmic split in an unfavorable position. Second, there is considerable variation in the performance of some targets, particularly Best-caseLatency, DSP, and FF, depending on how the algorithms are grouped. This suggest that cross-algorithm prediction is both unstable and weak on average.

The results from the repeated evaluations of these algorithms are consistent with those obtained during the initial groups-split evaluation. Most target algorithms exhibited mean R^2 values that were close to zero or even negative, suggesting that the performance degradation observed under the algorithm-aware evaluation protocol is systematic rather than incidental. Specifically, LUT, DSP, and FF consistently exhibited weak predictive performance across all repeated dataset splits, while BRAM_18K and Worst-caseLatency generally achieved values at or close to zero. Although Best-caseLatency had a slightly higher mean R^2 value than the other three, it exhibited a much larger standard deviation, indicating that its performance varied considerably depending on how the dataset was partitioned into algorithm groups.

Table 9: Random Forest performance across repeated group splits (mean ± SD)

Target	R^2 (mean ± SD)	MAE (mean ± SD)
BRAM_18K	0.0034 ± 0.0263	23.3001 ± 24.5678
Best-caseLatency	0.0811 ± 0.2178	424860834.5758 ± 532763987.8675
DSP	−0.5260 ± 1.0148	94.3128 ± 91.8712
FF	−0.8431 ± 0.9319	14773.2560 ± 4838.7439
LUT	−0.2675 ± 0.3290	19962.8407 ± 6556.3177
Worst-caseLatency	0.0059 ± 0.0405	5051448633.4849 ± 2052608806.3089

These repeated experiments provide further support for the main claim of this paper. The gap between the accuracy that can be achieved with random-split protocol and the generalization performance under the group-split protocol does not arise from an unfavorable split alone; it arises from the consistent limitations of the baseline model when evaluated on previously unseen algorithms. As such, the high performance observed under random-split evaluation should be interpreted as in-distribution performance and do not indicate the model's cross-algorithm predictive ability.

Additional Ablation Observations

An additional ablation experiment was performed by removing the metadata fields algo_name, top_function_name, and source_name, as well as by using only general features. The results of the random split evaluation did not show a significant difference after removing each identity-related feature individually. However, using only generic features resulted in a clear reduction in performance for LUT, DSP, and FF compared with the other two methods. Therefore, the high accuracy observed under random-split evaluation is not attributable to a single metadata field but rather to multiple correlated signals related to design characteristics and algorithm identity. Thus, this study finds that the primary evidence of overestimation does not from a single-variable ablation, but instead from the large and consistent gap between the results of random-split and algorithm-aware group-split evaluations.

This study produced several key findings. First, the selected design-level and code-level parameters are clearly useful, as they enable accurate estimation of

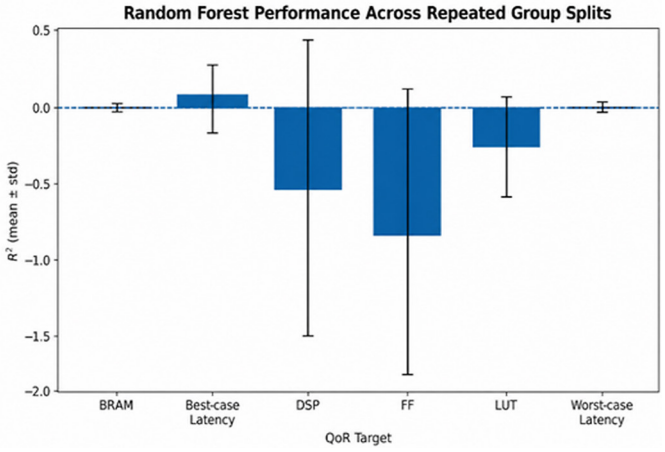

Fig. 4: Random Forest performance across repeated group splits based on algorithm identity, reported as mean R^2 with standard deviation over five random seeds.

both BRAM and Latency when using randomly divided data. Second, the fact that the predictions produced by both Random Forest and Extra Trees are very similar indicates that the primary findings of this work are likely applicable regardless of the ensemble method used. Third, and most importantly, the significant decrease in HLS QoR prediction performance under the group split method demonstrates that the primary challenge is achieving effective generalization beyond the specific algorithms used during training.

This finding explains why the importance of feature results for each algorithm was so similar. The reason the random-split evaluation is so successful is that it identifies patterns that are shared across the algorithms and familiar to both datasets, thereby enabling a high number of correct predictions. However, when the predictor is required to make predictions based on the two different algorithms separately, it does not have access to these common patterns, and its performance is severely reduced. Thus, while current tabular baselines may work well as in-distribution estimators, they remain limited in their ability to predict across algorithms [29].

At the same time, it would be incorrect to interpret this limitation as a failure of either the dataset or the baseline. Instead, these are valuable empirical findings demonstrating that potential future improvements in HLS QoR prediction will likely require either more sophisticated representations capable of identifying transferable structural features, more powerful training methodologies designed to improve the robustness of predictions for out-of-distribution cases, or both. Overall, the study results show that the method used to evaluate a QoR predictor plays a critical role in determining what constitutes a good-quality QoR predictor.

From a practical perspective, these findings are relevant to CAD tools and automated HLS optimization frameworks that rely on machine learning models to guide design decisions. A QoR predictor that performs well under random splitting but fails to generalize across algorithms may provide unreliable estimates when applied to previously unseen designs. Therefore, generalization-aware evaluation should be considered before integrating such predictors into automated design-space exploration frameworks, where inaccurate predictions may lead the optimization process toward unsuitable implementation choices.

This study has several limitations. First, the experiments were based on design-level tabular features from ForgeHLS and did not utilize richer structural representations, such as graphs or intermediate representations. Therefore, some degree of a generalization gap was expected as a result of the selected feature space and baseline models [30]. However, the results indicate that generalizing across algorithms with rich structural representations will remain challenging for a while longer. Second, the models were intentionally designed to be simple and interpretable. As such, they satisfy the methodological goals of this paper but could have been designed to achieve different levels of absolute performance and possibly different cross-algorithm behavior. Third, the strict evaluation setting was limited to evaluating transfer only across algorithm-name groupings. Although this provides an important prospect on unseen-algorithm generalization, it does not cover other potential real-world distribution shifts, such as cross-source, cross-device, or cross-benchmark transfer. Previous work in other application domains has likewise shown that model performance may change under distribution shifts, supporting the need for evaluation beyond a single distribution setting [31]. Finally, our findings depend on the specific retained features, preprocessing pipeline, and target selection used in this study.

Despite these limitations, the results demonstrate that random-split evaluations significantly overestimate the ability of HLS QoR predictors to predict quality of results in practical settings.

Conclusion

This paper examined HLS QoR prediction using the ForgeHLS design-level dataset, with a specific focus on the gap between in-distribution performance under random splits and cross-algorithm generalization under stricter evaluation. A reproducible multitarget prediction pipeline was developed to estimate six QoR objectives: BRAM_18K, LUT, DSP, FF, Best-caseLatency, and Worst-caseLatency. The results revealed a clear and consistent pattern. Under a conventional random split, the Random Forest baseline achieved strong performance, particularly for BRAM_18K and the two latency targets, indicating that the selected design- and code-level features are informative when the training and test data share similar algorithmic structure. However, when the evaluation was performed using a group split based on algo_name, performance declined substantially across most targets. Repeated group-split experiments confirmed that this decline was systematic rather than incidental. Feature importance analysis helps explain this behavior. The most influential predictors for latency were strongly associated with algorithm identity, top-level function information, and

source provenance, suggesting that the baseline model relies heavily on algorithm-specific regularities under random-split evaluation. As a result, strong performance under random-split evaluation should not be interpreted as evidence of robust generalization to unseen algorithms. Instead, the findings show that the evaluation protocol plays a decisive role in how HLS QoR prediction performance should be interpreted. In this sense, the main contribution of this work is both methodological and predictive: it demonstrates that random-split evaluation alone can substantially overestimate the practical capability of QoR predictors in realistic deployment settings. Several directions remain open for future work. Richer input representations, such as graph-based program encodings, intermediate-representation features, or hybrid structural-code features, may help improve transfer across algorithms. It is also important to investigate learning strategies that are explicitly designed for out-of-distribution robustness, as well as broader evaluation settings involving cross-source, cross-family, or cross-device prediction. Overall, future progress in HLS QoR prediction should be assessed not only by higher predictive accuracy but also by stronger evidence of genuine generalization. These findings highlight the importance of adopting generalization-aware evaluation protocols in future HLS QoR prediction studies to ensure reliable and practically meaningful performance assessment.

Funding

This research received no external funding.

Author Contributions

Ameen Shaheen: Conceptualization, methodology, experiments, analysis, and writing—original draft. Wael Alzyadat: Validation, review, and editing. Aysh Alhroob: Supervision, review, and editing. All authors reviewed and approved the final manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Nane, R., Sima, V.-M., Pilato, C., Choi, J., Fort, B., Canis, A., et al. (2016). A survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35*(10), 1591–1604. https://doi.org/10.1109/TCAD.2015.2513673
2. Dai, S., Zhou, Y., Zhang, H., Ustun, E., Young, E. F. Y., & Zhang, Z. (2018). Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (pp. 129–132). https://doi.org/10.1109/fccm.2018.00029
3. Wei, Z., Arora, A., Li, R., & John, L. (2023). HLSDataset: Open-source dataset for ML-assisted FPGA design using high level synthesis. *ArXiv (Cornell University)*. https://doi.org/10.1109/asap57973.2023.00040
4. Abi-Karam, S., Sarkar, R., Seigler, A., Lowe, S., Wei, Z., Chen, H., Rao, N., John, L., Arora, A., & Hao, C. (2024). HLSFactory: A framework empowering high-level synthesis datasets for machine learning and beyond. *MLCAD '24: Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD* (pp. 1–9). https://doi.org/10.1145/3670474.3685961
5. Peng, Z., Li, Z., Gao, M., Xu, Q., Zhang, C., & Zhao, J. (2025). ForgeHLS: A large-scale, open-source dataset for high-level synthesis. *arXiv preprint arXiv:2507.03255*. https://doi.org/10.48550/arXiv.2507.03255
6. Gao, M., Zhao, J., Lin, Z., & Guo, M. (2024). Hierarchical source-to-post-route QoR prediction in high-level synthesis with GNNs. *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1–6). https://doi.org/10.23919/date58400.2024.10546555
7. Masouros, D., Ferikoglou, A., Zervakis, G., Xydis, S., & Soudris, D. (2024). Late breaking results: Language-level QoR modeling for high-level synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (pp. 1–2). https://doi.org/10.1145/3649329.3663500
8. Aljebory, K., Jwmah, Y., Mohammed, T., & Al Mamarid, A. (2026). Comparative study of LSTM, GRU, and BRNN performance: Large-scale data analytics (EMG signal classification). *International Journal of Advances in Soft Computing and Its Applications, 18*(1), 99–117. https://doi.org/10.15849/ijasca.v18i1.28
9. Aliferis, C., & Simon, G. (2024). Overfitting, underfitting and general model overconfidence and under-performance pitfalls and best practices in machine learning and AI. In: Simon, G. J., & Aliferis, C., (Eds.). *Artificial Intelligence and Machine Learning in Health Care and Medical Sciences: Best Practices and Pitfalls*. Cham, Springer, (pp. 477–524). https://doi.org/10.1007/978-3-031-39355-6_10
10. Ma, W., Papadakis, M., Tsakmalis, A., Cordy, M., & Traon, Y. L. (2021). Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology, 30*(2), 1–22. https://doi.org/10.1145/3417330
11. Ferikoglou, A., Kakolyris, A., Kypriotis, V., Masouros, D., Soudris, D., & Xydis, S. (2024). Data-driven HLS optimization for reconfigurable accelerators. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (pp. 1–6). https://doi.org/10.1145/3649329.3658471
12. Ferikoglou, A., Kakolyris, A., Kypriotis, V., Masouros, D., Soudris, D., & Xydis, S. (2024). CollectiveHLS: Ultrafast knowledge-based HLS design optimization. *IEEE Embedded Systems Letters, 16*(2), 235–238. https://doi.org/10.1109/les.2023.3330610
13. Berrazueta-Mena, D., & Navas, B. (2025). AHA: Design and evaluation of compute-intensive hardware accelerators for AMD-Xilinx Zynq SoCs using HLS IP flow. *Computers, 14*(5), 189. https://doi.org/10.3390/computers14050189
14. Pouget, S., Pouchet, L.-N., & Cong, J. (2025). Automatic hardware pragma insertion in high-level synthesis: A non-linear programming approach. *ACM Transactions on*

Design Automation of Electronic Systems, 30(2), 1–44. https://doi.org/10.1145/3711847

15. Pouget, S., Pouchet, L.-N., & Cong, J. (2025). A unified framework for automated code transformation and pragma insertion. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (pp. 187–198). https://doi.org/10.1145/3706628.3708873
16. Shang, W., & Huang, X. (2025). A survey of large language models on generative graph analytics: Query, learning, and applications. *IEEE Transactions on Knowledge and Data Engineering, 37*(12), 6799–6819. https://doi.org/10.1109/tkde.2025.3609877
17. Xu, L., Wang, S., Casseau, E., & Xiao, C. (2025). Intelligent4DSE: Optimizing high-level synthesis design space exploration with graph neural networks and large language models. *arXiv preprint arXiv:2504.19649*. https://doi.org/10.48550/arXiv.2504.19649
18. Surendar, A. (2026). Urban sustainability through smart technologies: A review of global practices and research progress. *Journal of Smart Infrastructure and Environmental Sustainability*, *3*(1), 34–42.
19. Abdullah, D. (2026). Physics-informed machine learning for real-time stability control in renewable-dominant smart grids. *National Journal of Renewable Energy Systems and Innovation*, *2*(1) 1–9.
20. Jagadhabi, N. (2025). Machine learning models for automated testing prioritization in continuous integration systems. *SECITS Journal of Scalable Distributed Computing and Pipeline Automation*, *3*(1), 58–62.
21. Tandi, M. R., & Kumar, M. S. (2025). Quantum-inspired reconfigurable computing for accelerated matrix computations in high-performance embedded systems. *SCCTS Journal of Embedded Systems Design and Applications*, *3*(2), 48–58.
22. Kavitha, M. (2025). Energy-efficient Edge-AI accelerator design using reconfigurable FPGA-based VLSI architecture. *Journal of VLSI and Embedded System Design*, *1*, 26–33.
23. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324
24. Hammad, R., & AL-Slamat, A. (2026). A deep learning approach for predicting hospital length of stay for people with diabetes using electronic health records (EHRs). *International Journal of Advances in Soft Computing and Its Applications, 18*(1), 192–206. https://doi.org/10.15849/ijasca.v18i1.24
25. Durdy, S., Gaultois, M. W., Gusev, V. V., Bollegala, D., & Rosseinsky, M. J. (2022). Random projections and kernelised leave one cluster out cross validation: Universal baselines and evaluation tools for supervised machine learning of material properties. *Digital Discovery, 1*(6), 763–778. https://doi.org/10.1039/d2dd00039c
26. Altork, Y. (2025). Comparative analysis of machine learning models for wind speed forecasting: Support vector machines, fine tree, and linear regression approaches. *International Journal of Thermofluids, 27*, 101217. https://doi.org/10.1016/j.ijft.2025.101217
27. Hayajneh, A. M., Aldalahmeh, S., Alasali, F., Al-Obiedollah, H., Raza, A., & McLernon, D. (2023). Tiny machine learning on the edge: A framework for transfer learning empowered unmanned aerial vehicle assisted smart farming. *IET Smart Cities, 6*(1), 10–26. https://doi.org/10.1049/smc2.12072
28. Thalji, N., & Alwadi, M. (2026). Predicting apartment prices in Jordan using ensemble machine learning algorithms to support investment and urban planning decisions. *International Journal of Advances in Soft Computing and Its Applications, 18*(1), 321–338. https://doi.org/10.15849/ijasca.v18i1.42
29. Duan, H., Yang, Y., Abbasi, A., & Tam, K. Y. (2026). Robust predictive modeling under unseen data distribution shifts: A methodological commentary. *Information Systems Research*. https://doi.org/10.1287/isre.2022.0537
30. Verkerken, M., D'hooge, L., Wauters, T., Volckaert, B., & De Turck, F. (2022). Towards model generalization for intrusion detection: Unsupervised machine learning techniques. *Journal of Network and Systems Management, 30*(1), 12. https://doi.org/10.1007/s10922-021-09615-7
31. Ktena, I., Wiles, O., Albuquerque, I., Rebuffi, S.-A., Tanno, R., Roy, A. G., Azizi, S., Belgrave, D., Kohli, P., Cemgil, T., Karthikesalingam, A., & Gowal, S. (2024). Generative models improve fairness of medical classifiers under distribution shifts. *Nature Medicine, 30*(4), 1166–1173. https://doi.org/10.1038/s41591-024-02838-6