Efficient Understanding of Machine Learning Model Mispredictions

Martin Eberlein
Humboldt-Universität zu Berlin
Berlin, Germany
martin.eberlein@hu-berlin.de

Jürgen Cito
TU Wien
Vienna, Austria
juergen.cito@tuwien.ac.at

Lars Grunske
Humboldt-Universität zu Berlin
Berlin, Germany
grunske@informatik.hu-berlin.de

***Abstract*—Mispredictions by machine learning components can have severe consequences, especially in safety-critical and mission-critical software systems. Therefore, understanding and debugging these mispredictions is a crucial part of the development process for systems that use machine learning components. Previous research has successfully applied methods that identify when a model's predictions may be unreliable by generating a rule set that links feature values to prediction errors. However, current state-of-the-art rule set approaches require significant computational resources, particularly for large data sets.**

To address these high computational demands, we propose a strategy to identify and focus only on the most influential features that lead to mispredictions. Additionally, to improve the accuracy of mispredictions diagnosis, we replace traditional rule-based approaches with decision tree learning. We evaluate our tool MMDFast across 11 diverse real-world data sets. The results show that focusing on influential features with decision trees improves the accuracy of misprediction explanations, while significantly reducing computational demands in all scenarios. Thus, MMDFast produces better results much faster, making it more efficient and effective for generating misprediction explanations.

I. Introduction

Machine learning is applied to a wide range of real-world problems. As models grow in complexity, their decision processes become opaque. In high-stakes domains such as traffic management [1], medical diagnostics [2], and autonomous driving [3], this opacity raises serious risks. To ensure reliability, it is necessary to rigorously test and interpret these "black box" systems.

Testing machine learning models requires techniques that assess reliability and generalization. Even with rigorous validation, mispredictions remain common. Causes include poor data quality [4], such as noisy or imbalanced training sets; models that are too simplistic [5]; or concept drift [6], where target distributions shift over time and degrade performance. Although machine learning components often achieve strong results, understanding why they fail remains difficult. Mispredictions arise from diverse factors, which complicates efforts to analyze model behavior. Recent research has concentrated on two directions: making the decision process more transparent and applying debugging techniques, such as rule induction, to explain mispredictions [7]–[9].

Explainable machine learning aims to make the decisions of these models more transparent. Common approaches include attributing importance to input features [7] or constructing surrogate models that approximate the original system. These methods seek to improve both interpretability and debuggability [8]. Classical techniques, such as LIME [8] and Partial Dependence Plots [10], primarily analyze relationships between inputs and outputs, but they do not account for whether the predictions themselves are correct.

In contrast, Machine Learning Model Diagnosis (MMD) [9] constructs an interpretable set of rules that link specific feature values to increased likelihood of mispredictions. By iteratively refining these rules against a ground truth, MMD produces concise conditions that highlight when a model is systematically unreliable. These rules act as concrete "red-flag" indicators. When triggered, developers know to treat the corresponding prediction with caution. Such rules can be integrated into CI pipelines to flag risky predictions, guide manual reviews, or inform targeted data collection—thereby supporting debugging and increasing practitioner trust. The usefulness of misprediction explanations has been demonstrated in practice [9]. At Facebook, for instance, MMD revealed that long iOS crash traces were a major source of errors in the *Bug2Commit* model, guiding feature augmentation that reduced mispredictions. In another case, applied to *Oncall Recommendation*, MMD exposed how reliance on a legacy feature introduced noise into rank assignments, helping engineers identify and suppress problematic conditions. These case studies show how simple rules can uncover hidden failure modes and provide actionable guidance to improve real-world model usage [9].

While valuable, MMD has a critical limitation: scalability. Because it constructs rules from every feature in the data set, runtime grows quickly with feature count. On moderately sized data (e.g., 50,000 instances, ∼50 features), generating explanations can take up to an hour—too slow for iterative debugging or feedback loops. To illustrate, consider the Python merge-conflict data set [11] (approx. 50,000 commits, 28 features such as number of files added/removed, active developers, and commit frequency). Sophisticated predictive models achieve around 95% accuracy here, yet they still exhibit systematic blind spots. MMD surfaces explanations such as $\mathit{parallel\_files\_changed\_num} > 0 \wedge \mathit{files\_removed} > 2$—that is, commits that change files simultaneously and delete more than two files are precisely the conditions under which the model

tends to err, providing a characterization of its mispredictions. However, producing such a rule with MMD is computationally intensive and yields limited diagnostic quality: on our setup, MMD required about 37 minutes and achieved only 0.186 precision and 0.443 recall.

We therefore introduce MMDFAST, a diagnosis approach that retains the rule-level interpretability of MMD while (i) drastically reducing runtime and (ii) improving the quality of misprediction explanations. With MMDFAST, the same task completes in under two seconds and improves to 0.344 precision and 0.617 recall, roughly doubling precision and substantially increasing recall compared to MMD.

MMDFAST extends the MMD framework in two significant ways. First, to address the computational demands of rule induction, we focus the search space by prioritizing only the most influential features that correlate with mispredictions. Concretely, we train an auxiliary model on the (correct vs. mispredicted) labels to rank and retain only the top-k features. This reduces the search space significantly and accelerates rule learning while preserving the most informative properties. Second, we enhance the rule construction process by utilizing decision trees, aiming to improve the effectiveness of the generated rule sets. Decision trees, known for their interpretability and robustness, can create more precise and understandable rules. While enhancing the respective performance and predictive metrics, we also consider the length of the rule set to maintain human interpretability and prevent data overfitting.

In summary, we make the following contributions:

Fast, Focused Explanations. We show that restricting rule learning to a small set of k influential features dramatically reduces computational demand without degrading explanation quality. An ablation over k finds that k=3 offers a strong accuracy–efficiency trade-off, with larger k yielding diminishing returns.

Optimized Rule Construction. We replace costly rule induction with limited-depth decision trees to generate concise, interpretable rule sets. This design improves predictive performance and guarantees readability by limiting rule depth, yielding actionable rules that practitioners can use to flag risky predictions, guide reviews, or trigger fallbacks.

II. Misprediction Explanation Generation

Machine learning models often mispredict, and it is difficult to determine when these models provide wrong predictions. Misprediction explanation generation is a process that systematically produces rule sets that characterize where the model is likely to mispredict [9]. Such rules make a model's limitations explicit, helping developers judge reliability and decide when to trust, double-check, or override a prediction. In practice, these explanations serve as actionable "red-flag" indicators: when triggered, they warn practitioners to exercise caution, request manual review, or collect additional data. The process involves three main steps:

1) **Labeling Mispredictions:** The first step involves labeling each prediction as either correct or a misprediction. This labeling is based on comparing the model's predictions to the known ground truth.
2) **Generating Atomic Predicates:** Atomic predicates are the basic building blocks for rule sets. They are derived from the input data and consist of conditions based on features, comparison operators, and values. These predicates represent conditions such as *files_removed* > 2 that represent a subset of the input space.
3) **Learning Rules:** Using the labeled data and atomic predicates, misprediction explanation generation employs rule induction to iteratively build a rule set. The process involves searching for the best-performing predicates that cover the set of mispredictions. The work by Cito et al. uses beam search to find the optimal rules in the search space while balancing precision, recall, and rule size [9].

The rule construction is model-agnostic: it applies to any classifier and produces concise, interpretable rules that surface recurring mispredicitons patterns. These explanations have been shown to provide developers with tangible debugging insights and improve trust in model predictions. However, scalability is a major obstacle. Because current state-of-the-art approaches, like MMD, generate predicates for every feature, runtime grows quickly with feature count, limiting its usefulness in practical debugging scenarios.

Our goal is to retain the interpretability and actionability of MMD while addressing its computational bottlenecks. Specifically, we aim to reduce runtime by focusing on only the most influential features, while still achieving comparable or better precision, recall, and rule quality.

III. Approach

In this section, we introduce MMDFAST, our approach for constructing misprediction explanations of black box machine learning models. Building on MMD, we aim to preserve its interpretability while overcoming its main limitations: high computational demand and poor explanation quality. MMDFAST addresses this challenge through two complementary design choices: (i) *minimizing computational demand* by focusing only on the most influential features, and (ii) *improving explanation quality* by replacing rule induction with decision tree–based rule construction.

Reducing Computational Demand, Instead of generating predicates for every feature, which quickly becomes infeasible on feature-rich data sets, we identify and retain only the features that most strongly drive mispredictions. This reduces the search space and allows rule generation to scale to larger data sets.

Improving Explanation Quality. Traditional rule induction often yields either weak or overly complex rules. We instead employ limited-depth decision trees that balance precision and recall while explicitly limiting rule length, thereby producing concise and human-readable rules.

A. Overview

Figure 1 shows how MMDFAST works. MMDFAST consists of several key steps, starting with a trained model M

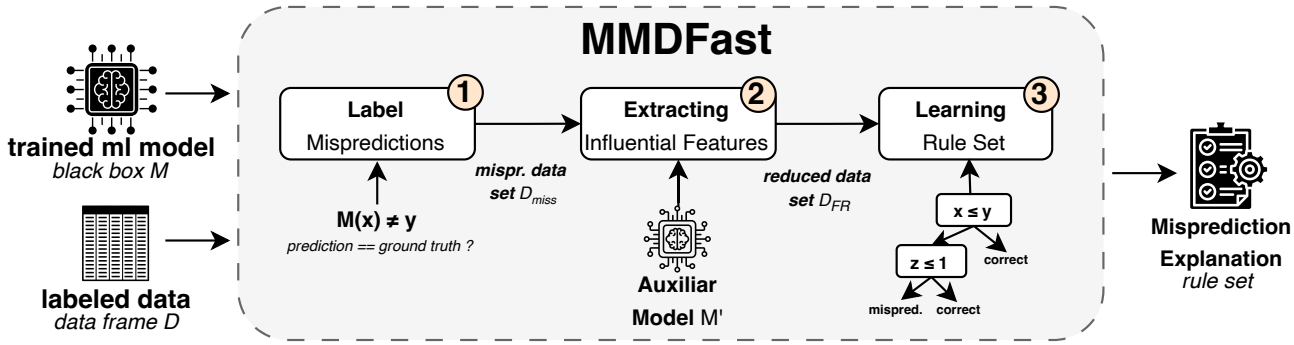


Fig. 1. How MMDFAST works. Starting with a trained model M and labeled data set D, MMDFAST first identifies mispredictions, then extracts the most influential features to reduce computational demand, and finally constructs rule sets using a decision tree. The final output is a rule set that explains the mispredictions of the machine learning model.

and a labeled data set D, and culminating in the generation of rule-based explanations of the mispredictions:

1) **Label Mispredictions**: (1) Similar to MMD, MMDFAST starts with a trained model and a labeled data set to identify instances in the data set where the model's predictions do not match the known ground truth (Section III-B).
2) **Extract Influential Features**: (2) Then, to enhance scalability, MMDFAST identifies the most influential features contributing to the mispredictions. This is achieved by training an auxiliary model to predict mispredictions and using feature importance metrics to determine which features have the highest impact (Section III-C).
3) **Learn Rule Set**: (3) Next, MMDFAST uses a decision tree to construct the misprediction rule sets. Our technique iteratively builds rules that explain mispredictions by covering as many mispredicted instances as possible while minimizing false positives (Section III-D).

Finally, the generated rule sets are formatted to provide a uniform and interpretable output. This step ensures that the explanations are easily understandable and actionable.

B. Labeling Mispredictions

Similar to MMD, given a trained model and labeled data set, the process of labeling the mispredictions is straightforward. Considering a labeled data set $D\colon X \to Y$ and a trained machine learning model $M\colon X \to Y$, where $x \in X$ is a data instance and $y \in Y$, $y \in \{0,1\}$ is the matched target variable, and noting that D actually contains the ground truth for its data instances, we define the mispredicitons function $L\colon X \to \{0,1\}$ as follows:

$$L(x) = \begin{cases} 1 & \text{if } M(x) \neq D(x) \\ 0 & \text{otherwise} \end{cases}$$

In words, the function $L(x)$ is 1 if the prediction given by the machine learning model for a data instance x does not match the label found in the data set, which is the known ground truth. In this way, we build a new data set $D_{\text{mis}}\colon X \to \{0,1\}$ such that the following equation is valid:

$$D_{\text{mis}}(x) = 1 \iff M(x) \neq D(x)$$

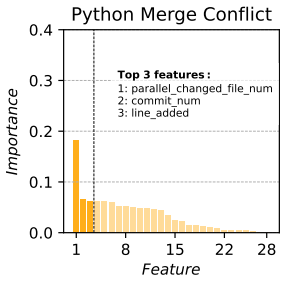
The resulting new misprediction-labeled data set D_{mis} contains each input instance in D but now linked to a new boolean label, indicating whether the given instance is mispredicted by the model. This newly constructed data set is essential for extracting the most influential features and subsequently assembling our misprediction explanations.

C. Extracting Influential Features

One of the major limitations of MMD is its high computational demand for large data sets, especially when the number of features is extensive. This is because the number of potential atomic predicates that need to be considered for rule learning grows rapidly with more features.

To address scalability, MMDFAST focuses on the features most strongly associated with mispredictions and discards less-impactful ones. The idea is that only a handful of features are needed to characterize mispredictions. We therefore train an auxiliary model M' on the newly constructed data set D_{mis} created in the previous step. The sole purpose of this auxiliary model is to predict whether an instance will be mispredicted by the original model M. Formally, the process can be defined as follows:

1) **Input**: A labeled data set $D_{\text{mis}} : X \to 0,1$, where each instance $x \in X$ is labeled as 1 if it is mispredicted by the model and 0 otherwise; a machine learning model M' used to predict the labels for D_{mis}; and a chosen number of influential features k.
2) **Training an Auxiliar Model**: Train a secondary machine learning model M', such as a random forest classifier or neural network, on D_{mis} to predict whether an instance will be mispredicted by the original model M. This captures the relationship between features and misprediction likelihood. For feature importance, we use *Gini Importance*, which measures the average decrease in impurity across all trees. Alternatives such as Permutation Importance, which evaluates performance drops under feature permutation, and SHAP [7], based on Shapley values, were considered, but preliminary experiments showed Gini Importance to be most consistent and reliable.



Feature	Imp. Score
parallel_changed_file_num	**0.182383**
commit_num	**0.070173**
line_added	**0.066021**
duration	0.061488
line_removed	0.061168
file_modified	0.060463
...	...
file_copied	0.000000
file_renamed	0.000000

Fig. 2. Most influential features for mispredictions in the Python merge conflict data set (k=3).

3) **Calculating Feature Importance**: Compute the feature importance of each input feature f_i using the Gini Importance from M'.
4) **Selecting Influential Features**: Ranks the features based on their computed importance scores and select the top k features with the highest importance scores, denoted as $F_{\text{FR}} = \{f_1, f_2, \ldots, f_k\}$.
5) **Remove Unimportant Features**: Remove the features that are not part of F_{FR} from D_{mis}.
6) **Output**: A reduced data set D_{FR} containing only the most influential features F_{FR}.

By focusing on these key features, MMDFAST reduces the number of features considered in subsequent steps, thus lowering computational costs and improving efficiency.

Running Example: To illustrate how we select the most influential features, we revisit the Python merge conflict data set. After training the auxiliary M' on D_{mis}, we compute the feature-importance scores. Figure 2 ranks all 28 features by importance and exhibits a clear long-tail pattern: only a few features are truly dominant for misprediction classification. Setting, for instance, k=3, we retain *parallel_changed_file_num* (0.18), *commit_num* (0.07), and *line_added* (0.06), while discarding the remainder. Guided by this selection, subsequent rule learning focuses on these three features, substantially shrinking the search space and yielding much faster, more interpretable rules.

D. Learning Rule Sets

The core of MMDFAST is the generation of rule sets that explain mispredictions. We detail how MMDFAST constructs these rules with a focus on precision, efficiency, and adequate coverage of mispredicted instances.

1) General Rule Construction: The generall idea of our rule construction approach is that we first construct an initial rule aiming at covering as many mispredicted instances as possible while minimizing false positives. Each rule consists of a series of atomic predicates connected by conjunctions. A predicate typically takes the form of a condition on a feature, such as "feature $>$ value". Formally, the initial rule can be defined as:

$$\text{Rule}_1 = P_1 \wedge P_2 \wedge \ldots \wedge P_n$$

where P_i represents individual predicates.

Once the initial rule is constructed, it is added to the current rule set. Subsequent rules are connected to the existing rule set using disjunctions. The objective is to iteratively expand the rule set until it achieves a predefined desired coverage of mispredictions. The process involves:

1) **Adding New Rules**: After constructing a rule, it is added to the rule set, which now takes the form:

$$\text{Rule Set} = \text{Rule}_1 \vee \text{Rule}_2 \vee \ldots \vee \text{Rule}_m$$

2) **Coverage Calculation**: The recall of the current rule set, measuring the percentage of mispredicted instances it covers, is calculated and compared against the desired coverage.
3) **Iterative Refinement**: If the desired coverage is not met, the process continues by iteratively constructing additional rules. This involves creating a new data set from instances not already covered by the current rule set and generating new rules specifically for these remaining instances.

2) Decision Tree Integration: MMDFAST leverages decision trees to determine the rules for covering mispredictions. The decision tree is trained on the data instances not covered—for the initial rule, we use all data instances—by the current rule set, using misprediction as the target variable. Each iteration involves the following steps:

1) **Training the Decision Tree**: A new decision tree is trained on the uncovered instances.
2) **Extracting Predicates**: Individual predicates are extracted from the decision tree by traversing its structure. Each node in the tree represents a predicate.
3) **Rule Evaluation**: The performance of each rule is assessed based on the number of mispredicted and correctly predicted instances it covers.

3) Rule Selection and Optimization: The best performing rule is selected by calculating *Precision*, *Recall*, and the weighted F_β*-Score*, with the highest F_β*-Score* indicating the best balance between precision and recall. The selected rule is then added to the rule set. If the desired coverage is not reached, the covered instances are removed from D_{mis}, and a new cycle begins. Mathematically, the weighted F_β*-Score* is defined as:

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}}$$

where β is chosen such that recall is considered β times as important as precision.

4) Final Evaluation: The process continues until the desired coverage is achieved. The final *rule set* is then evaluated for *Precision* and *Recall* to ensure it accurately and comprehensively explains the mispredictions.

Running Example: The last step for our running example involves constructing the final rule set with our decision tree approach. Figure 3 shows the initial tree learned on the reduced data set, containing just the most influential features. The initial rule from this tree consists of the two predicates

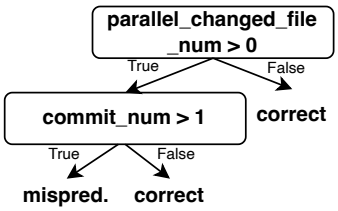


Fig. 3. Initial decision tree for the Python merge conflict data set.

parallel_changed_file_num $>$ 0 and commit_num $>$ 1. Refining this initial rule results in the following misprediction explanation: *(parallel_changed_file_num* $>$ *0* $\wedge$ *commit_num* $>$ *1)* $\vee$ *line_added* $>$ 23. Both MMD and MMDFAST surface *parallel_changed_file_num* as a primary factor of mispredictions. The difference is that MMDFAST, after ranking features, pairs it with *line_added* (rather than *files_removed*) because *line_added* is scored as more informative for misprediction. On our setup, MMDFAST learns this rule in under two second, while MMD required about 37 minutes. The resulting rules are short, easy to read, and, as shown in our evaluation, yield much higher accuracy than the rule set produced by MMD.

Summary: MMDFAST improves the generation of misprediction explanations for black-box machine learning models. By optimizing rule generation and focusing on the most influential features, we enhance the efficiency and quality of explanations, making them more actionable and easier to understand. Our versatile approach can be adapted to various machine learning models and data sets for better model diagnosis and improvement.

IV. Implementation

We have implemented MMDFAST as a Python library, extending the framework of MMD. Similar to MMD, our implementation takes a Pandas dataframe, a target variable, and a set of optional parameters as input, and returns a set of decision lists paired with precision, recall, and coverage metrics. The primary parameters that need to be provided are: the number k for selecting the most influential input features (default is k=3), the desired coverage (default is c=0.6), and the weight for the F_β*-Score* used to select the best-performing rule (default is β=0.5), slightly favoring precision over recall.

As our approach aims to improve computational efficiency, we chose a random forest classifier as the auxiliary model. Although other models, such as neural networks, can perform better, random forests provide very consistent results and are extremely fast. Our implementation supports the use of other auxiliary models.

We utilized the Python library scikit-learn [12] to train both the auxiliary model M' and the decision trees for rule construction. Both machine learning models are trained using default settings. Additionally, for the decision trees, we allow the tree depth to be defined by an input parameter, which directly controls the maximum length of an individual rule (default is d=2). This limits the rules extracted from the trees to two predicates. To ensure transparency and reproducibility, we have made MMDFAST and our complete evaluation publicly available. This includes all means necessary to reproduce the results presented in this paper.

TABLE I
EVALUATION SUBJECTS WITH NUMBER OF FEATURES, INSTANCES (INST.), AND FEATURE DESCRIPTIONS.

Task		Feat.	Inst.	Feature description
Bank Marketing		20	6,797	Bank marketing prediction with client demographics, contact/campaign details, and macroeconomic indicators
Hotel Booking		59	7,135	Hotel booking records with guest demographics, reservation details, booking history, and transactional attributes.
Spam Email Detection		100	9,000	Spam classification with anonymized numerical features (f1–f99); feature semantics are unspecified.
Water Potability		9	5,940	Water quality data set with chemical, physical, and contaminant indicators for potability prediction.
Heart Failure		13	300	Heart failure patient data with demographics, medical conditions, and clinical measurements.
Job Change		13	5,748	Job change prediction data set with candidate demographics, education, experience, and employment history from a data science training program.
Bug Report Close Time		21	1,481	Bug report closing time prediction using report metadata, user activity, project statistics, and process timing indicators.
Merge Conflict	Java	28	26,699	Merge conflict data set with file- and line-level change statistics, developer activity metrics, change type frequencies, and commit message characteristics.
	PHP	28	49,453	
	Ruby	28	40,129	
	Python	28	50,342	

V. Evaluation Setup

In our evaluation, we assess the performance of our approach MMDFAST compared to MMD by addressing the following research questions:

RQ1 How does MMD with the most influential feature reduction (MMD+FR) compare to standard MMD?

RQ2 How does MMDFAST compare to MMD?

For RQ1, we use the same rule induction approach introduced by MMD and extend it with our feature reduction (MMD+FR). For RQ2, we compare MMDFAST (which incorporates both the feature reduction strategy and the decision tree-guided rule construction) to the standard MMD approach.

A. Evaluation Subjects

To answer the research questions, we evaluate our tool's misprediction rules on a set of test subjects introduced by Gesi et al [13]. In total, we evaluate 11 real-world data sets, encompassing both software engineering and non-software engineering tasks. These tasks include: *merge conflict prediction* for four programming languages (*Java*, *PHP*, *Python*, and *Ruby*), *bug report close time prediction*, *job change prediction*, *bank marketing*, *hotel booking*, *water quality assessment*, and *spam email detection*. Unfortunately, we could not use the subjects from the initial MMD evaluation, as they are part of

Fig. 4. Feature importance distributions for all data sets (Python Merge Conflict in Figure 2). In most cases, a few dominant features stand out, while the rest follow a long tail, supporting the choice of focusing on the top-3 features.

internal company data [9]. Table I shows all data sets with their respective number of features and instances. These data sets provide a diverse and general evaluation ground. Preprocessing steps are applied as necessary, including handling missing values, normalizing data, and encoding categorical variables. On these data sets, we trained random forest classifier which serves as the black box model M. Each data set is split into training and test sets with a 80/20 split, maintaining the correct proportions of target labels. Fixed random seeds are used to ensure reproducibility. Models are trained using the default settings of the scikit-learn library. Note that our approach is model-agnostic, as MMDFAST only requires labeled misprediction data. The construction of such a data set can be done for *any* machine learning model.

B. Baseline

To assess the performance of MMD+FR and MMDFAST, we compare them to MMD [9]. We used the standard parameter settings of MMD, including the weights assigned to precision, recall, and rule length when evaluating rule performance. MMD considers rule set length, so the desired coverage is not guaranteed and can be lower if too many rules are needed to reach the specified coverage. Consequently, MMD provides multiple possible rule sets, ranked primarily by precision and secondarily by recall, reflecting the trade-off between these metrics. For our evaluation, we selected the highest-ranked explanation.

C. Research Protocol

To answer our research questions, we proceed in three steps: First, train a random forest model M for each data set D and split the data into training and test sets. Next, using the model M and the training set, construct misprediction diagnoses with MMDFAST and MMD, and record CPU time as a measure of

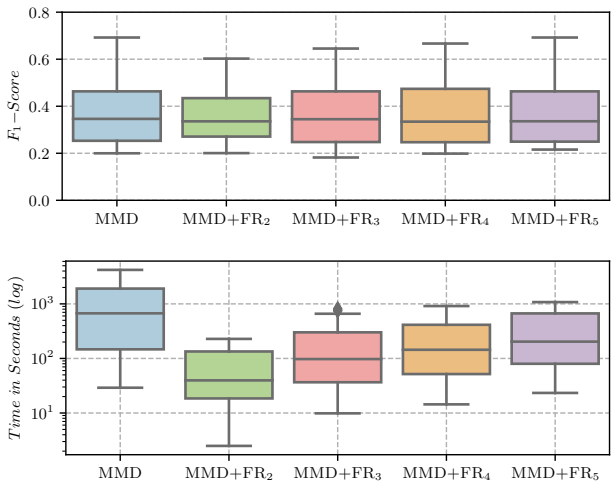


Fig. 5. Aggregated results for MMD+FR$_k$ with different numbers of extracted features k over all subjects.

computational efficiency (Times for MMD+FR and MMDFAST include feature selection). Finally, we used the test set to measure *Precision*, *Recall*, and F_1-*Score*. *Precision* is the ratio of true positives to all predicted positives, *Recall* is the ratio of true positives to all actual positives, and F_1-*Score* is the harmonic mean of precision and recall. These metrics measure how well the constructed rules predict model mispredictions.

To remove the model variance, we used five-fold cross-validation and repeated it five times with random seeds, which is a common approach used by other studies [13]. Thus, this procedure was repeated for each subject 25 times to account for the variability in data splitting and learning of the ML models, as well as variance in CPU time, ensuring reliable conclusions. To answer RQ1, we assessed the reduction in computational demand introduced by selecting the most influ-

TABLE II
COMPARISON OF PRECISION, RECALL, AND F_1-*Score* FOR MMD AND MMD+FR$_3$. SIGNIFICANT RESULTS ARE BOLD.

Subject	MMD			MMD+FR$_3$		
	Prec.	Recall	F_1-*Score*	Prec.	Recall	F_1-*Score*
bank	0.355	0.685	0.459	0.344	0.705	0.456
bugreport	0.227	0.736	0.346	0.227	0.760	0.345
heart	0.465	0.800	0.600	0.450	0.750	0.563
hotel	0.195	0.877	0.319	0.202	0.812	0.324
java	0.195	0.373	0.246	0.179	0.402	0.247
job	0.317	0.625	0.418	0.413	0.456	†**0.432**
php	0.136	0.627	0.226	0.155	0.509	***0.233**
python	0.169	0.476	0.251	0.196	0.382	0.259
ruby	0.144	0.614	0.233	0.146	0.574	0.233
spam	0.379	0.624	†**0.472**	0.383	0.492	0.444
water	0.388	0.849	0.498	0.362	0.892	0.516

TABLE III
COMPARISON OF CPU TIME (IN SECONDS) FOR MMD AND MMD+FR$_3$. SIGNIFICANT RESULTS ARE BOLD.

Subject	MMD	MMD+FR$_3$		Improv. Factor
	Total	Total	FR-Time	
bank	791	†**203.45**	1.33	3.9
bugreport	352	†**37.58**	0.89	9.4
heart	54	†**13.49**	0.33	4.1
hotel	237	†**47.57**	1.45	5.0
java	2098	†**424.38**	1.46	4.9
job	144	†**93.46**	1.24	1.5
php	1877	†**235.21**	1.64	8.0
python	1944	†**316.39**	1.56	6.1
ruby	2277	†**431.73**	1.23	5.3
spam	635	†**64.93**	12.37	9.8
water	94	†**18.31**	0.44	5.1

ential features, comparing the CPU times of both approaches. To answer RQ2, we compared the overall predictive power of MMDFAST's misprediction diagnoses to those of MMD. For both research questions, we applied the Mann–Whitney U test [14] with a significance threshold of $\alpha = 0.05$ to compare the two approaches. Results that are statistically significant are highlighted in *bold* and further annotated to indicate the level of significance: * ($p < 0.05$), ** ($p < 0.01$), and † ($p < 0.001$).

The experiments were conducted on a compute server equipped with an AMD EPYC 7713P processor (64 cores) and 256 GB of system memory.

VI. EVALUATION RESULTS

In this section, we present the results of our evaluation. All our results are detailed in Table II to Table V and Figure 4 to Figure 6. Table II to Table V display the median results across all subjects and runs.

A. RQ1: MMD+FR vs. MMD

The first research question investigates whether our feature selection strategy improves the performance of MMD. The underlying hypothesis is that by focusing only on the most relevant aspects of the misprediction, MMD+FR can produce better rule sets much faster.

Since our feature selection strategy depends on the parameter k, which specifies the maximum number of features to be selected, the first step is to analyze its influence. Figure 5 shows the aggregated results—F_1-*Score* and CPU time—over all 11 subjects for MMD+FR$_k$ with ranging k and MMD. We observe that the F_1-*Score* shows no significant difference when measured across all subjects. However, the time required to construct the misprediction diagnoses decreases with lower values of k. This aligns with our expectations, as focusing on the most influential features and discarding less impactful ones reduces computational demand. For the remainder of RQ1, we examine individual subjects in more detail. We use MMD+FR$_3$ as the default, since it achieves F_1-*Score* comparable to the baseline while substantially reducing compute time. Additionally, setting k=3 mitigates overfitting and keeps rules concise by limiting the learner to at most three features.

Figure 4 plots the feature-importance scores and highlights the top three features MMD+FR$_3$ identified as most influential for each data set. With the exception of *water* and *spam*, the distributions exhibit a long-tailed, power-law shape. A few dominant features stand out, while most others contribute little. This pattern supports our hypothesis that only a small subset of features is truly critical for explaining mispredictions. In the merge-conflict data sets in particular, *commit size* and *parallel file changes* consistently dominate, indicating that these factors are the primary drivers of mispredictions.

Table II presents the achieved *Precision*, *Recall*, and F_1-*Score* for MMD and MMD+FR$_3$ for all 11 subjects. The median F_1-*Score* ranges from 0.22 for *PHP* up to 0.60 for *heart*. Based on this comparison, we observe no consistent outperformer between the two methods. A detailed statistical analysis, supports these observations. Specifically, MMD+FR$_3$ performs better for the *job* and *PHP* subjects, while MMD shows better results for the *spam* subject. For the remaining subjects, no statistically significant differences were observed between MMD and MMD+FR$_3$. Thus, we conclude that there is no overall performance difference between the two approaches.

> *Focusing on the most influential features achieves similar results (**F_1-Score**s) to the baseline.*

One of the primary goals of our feature reduction strategy is to reduce computational demand. By narrowing the focus to the most influential features, we reduce the number of atomic predicates that need to be considered for rule induction, i.e., the creation of the misprediction explanation. While the F_1-*Score* provides insight into the accuracy and balance of *Precision* and *Recall*, it does not capture computational efficiency directly. Hence, we also measured the execution time required for both MMD and MMD+FR$_3$.

Figure 6 displays the required execution time to construct the misprediction explanations, with the results for all 11 subjects and the required execution time in seconds on the y-axis (log scale). This visual representation highlights the significant computational efficiency improvement achieved by our feature selection strategy. We observe that reducing the number of

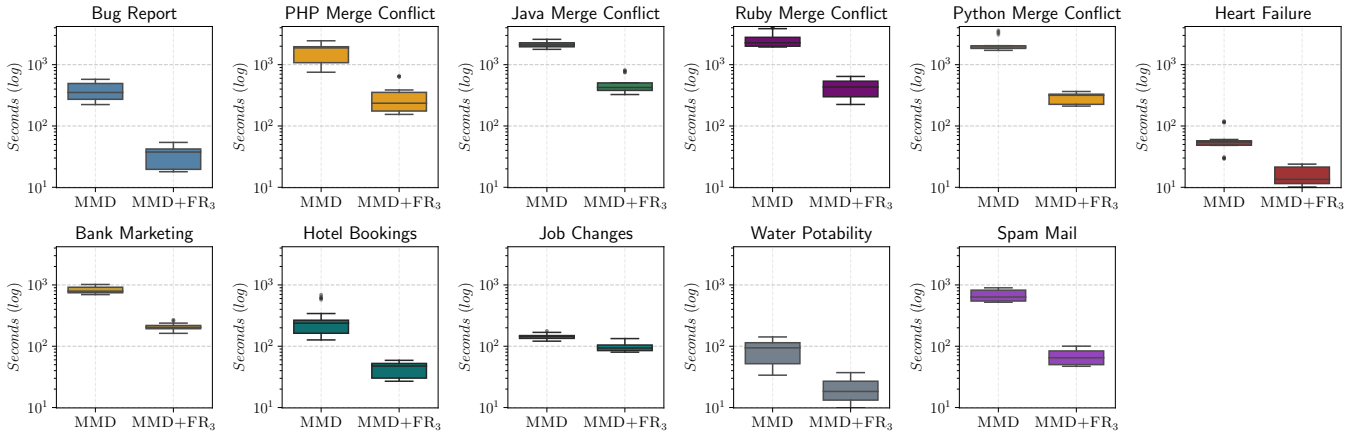


Fig. 6. CPU-time comparison between MMD and MMD+FR$_3$ (log scale). Across all subjects, MMD+FR$_3$ consistently runs faster than MMD.

TABLE IV
COMPARISON OF PRECISION, RECALL, AND F_1-*Score* FOR MMD AND MMDFAST. SIGNIFICANT RESULTS ARE BOLD.

Subject	MMD			MMDFAST		
	Prec.	Recall	F_1-*Score*	Prec.	Recall	F_1-*Score*
bank	0.355	0.685	0.459	0.385	0.643	†**0.486**
bugreport	0.227	0.736	0.346	0.278	0.665	†**0.400**
heart	0.465	0.800	0.600	0.577	0.652	0.588
hotel	0.195	0.877	0.319	0.203	0.850	***0.329**
java	0.195	0.373	0.246	0.300	0.694	†**0.424**
job	0.317	0.625	0.418	0.322	0.687	****0.435**
php	0.136	0.627	0.226	0.326	0.664	†**0.438**
python	0.169	0.476	0.251	0.344	0.624	†**0.451**
ruby	0.144	0.614	0.233	0.349	0.625	†**0.456**
spam	0.379	0.624	0.472	0.355	0.835	†**0.499**
water	0.388	0.849	0.498	0.403	0.821	***0.522**

TABLE V
COMPARISON OF CPU TIME (IN SECONDS) FOR MMD AND MMDFAST. SIGNIFICANT RESULTS ARE BOLD.

Subject	MMD	MMDFAST	Improv. Factor
	Total	Total	
bank	791	†**1.50**	527
bugreport	352	†**1.04**	338
heart	54	†**0.39**	138
hotel	237	†**1.58**	150
java	2098	†**1.54**	1,362
job	144	†**1.29**	111
php	1877	†**1.73**	1,084
python	1944	†**1.62**	1,200
ruby	2277	†**1.39**	1,638
spam	635	†**12.40**	51
water	94	†**0.49**	191

features significantly decreases computational demand across all subjects.

The results in Table III confirm this observation. While the median execution time for MMD ranges from around 54 seconds (*heart*) to 39 minutes (*ruby*), MMD+FR only requires around 13 seconds (*heart*) to 7 minutes (*ruby*). Additionally, Table III includes the time required to extract the most influential features, which contributes minimally to the total time but results in substantial overall improvement. The outlier is *spam*, which has the most features (100). Moreover, Table III shows the speed-up factor of MMD+FR compared to MMD. Our approach achieves significant speed-ups, ranging from 1.5 times (*job*) to nearly 10 times for the large data set *spam*.

Focusing on the most influential features results in a significant speed-up ranging from 1.5 to 9.8 times.

Our investigation into RQ1 shows that the feature selection strategy incorporated in MMD+FR can significantly reduce computational demand without a significant decrease in accuracy. This demonstrates the effectiveness of focusing on the most influential features to streamline the rule induction process and enhance computational efficiency.

B. RQ2: MMDFast vs. MMD

To answer RQ2, we compare MMDFAST—which combines feature selection (k=3) with our decision-tree–based rule construction—to the original MMD.[1] Table IV reports the median *Precision*, *Recall*, and F_1-*Score* across subjects. Overall, MMDFAST consistently outperforms MMD, achieving significant improvements in 10 out of 11 data sets. Significant results are shown in bold. Median F_1-*Score* ranges from 0.32 (*hotel*) to 0.58 (*heart*), with notable gains such as Ruby (0.233 to 0.456, doubling the F_1-*Score* by raising precision while maintaining recall), Python (0.251 to 0.451, improvements in both precision and recall), and Spam (0.472 to 0.499, higher recall at a small cost in precision).

MMDFAST *achieves higher* ***F*$_1$*-Score*** *for almost all data sets.*

Beyond the improvement in F_1-*Score*, we also investigate the execution time of MMDFAST. Table V shows the required CPU time for MMD and MMDFAST—including the feature

[1] Preliminary experiments showed that feature selection is crucial: it restricts rule learning to the most relevant features, reducing overfitting and excluding noisy variables.

reduction learning time. With the integration of both the feature selection strategy and the new rule construction approach, we observe a substantial reduction in computational demand. While MMD+FR (with just the feature reduction) already improved execution time by up to 10 times, MMDFAST further reduces computational demand by up to two orders of magnitude. The statistical analysis confirms that MMDFAST reduces computational demand for all subjects significantly. Whereas MMD and MMD+FR required multiple seconds to half an hour, MMDFAST consistently requires only a few seconds. The exception is the extremely large data set *spam*, where MMDFAST requires an average of only 12 seconds compared to MMD's 635 seconds (10 minutes).

MMDFAST *drastically reduces the time needed for all subjects.*

Summarizing the results of RQ2, we conclude that MMDFAST not only significantly improves the accuracy for most subjects but also drastically reduces the time required to calculate misprediction explanations. Thus, MMDFAST produces better results much faster, making it a more efficient and effective tool for generating misprediction explanations.

Based on our findings, we draw two main conclusions:

Decision Tree Rule Construction Our evaluation shows that MMDFAST consistently improves performance metrics and significantly reduces computational demand compared to MMD. This makes MMDFAST a highly effective and efficient method for generating misprediction explanations.

Feature Selection Strategy Our method of utilizing the most influential features significantly reduces computational demand for generating misprediction explanations. This reduction in computational demand enhances the scalability and feasibility for use with large, feature-rich data sets. This indicates that our feature selection strategy is a crucial component in improving the efficiency of misprediction explanation generation.

VII. Threats to Validity

Internal Validity: The implementation details and parameter settings for both MMD and MMDFAST pose a threat to internal validity. We used default settings for the random forest models and standard parameters for MMD, but different configurations might yield different results. Computational demand was measured by CPU time, excluding other factors such as memory usage and parallel processing capabilities. Including these factors in the future could provide a more comprehensive assessment. Finally, we used impurity-based feature importance for selecting influential features, which has known biases [15]. Exploring alternative techniques could validate the consistency and reliability of our approach.

External Validity: Selection bias may influence our results, as we evaluated MMDFAST using 11 diverse real-world data sets. Although these data sets cover various applications, their specific characteristics could affect the outcomes. To mitigate this risk, we used the same subjects as Gesi et al [13], covering a diverse set of prediction tasks. Our evaluation focused on random forest models, which may limit generalizability to other machine learning models like neural networks or support vector machines. Future research should assess MMDFAST across a broader range of models. Additionally, domain-specific factors may affect MMDFAST's effectiveness. Investigating its performance in specific areas like healthcare or finance could provide valuable insights.

VIII. Discussion

A. Interpretability & Actionability

Across data sets, the learned rules are short, interpretable, and point to concrete "caution zones" where the trained models are prone to error. A consistent pattern emerges in all merge-conflict subjects (Java, Python, Ruby, PHP): rules include *parallel_changed_file_num*, often combined with churn or timing signals. For instance, in *PHP* the rule *parallel_changed_file_num* $> 0.5 \wedge$ *duration* ≤ 1.5 (recall 0.854, precision 0.325) says that fast, multi-file commits are where mispredicitons concentrate. This likely reflects interaction complexity and rushed changes overwhelming simple predictors. Based on this rule, some consequences could be to add CI gates (e.g., require an extra reviewer or run conflict detectors) when the rule fires, train a small specialized sub-model for high-parallelism commits, or prioritize collecting more labeled examples from high-parallelism regions to reduce uncertainty.

For the non-SE tasks the rules translate into equally actionable guidance. For the hotel data set, the rule *lead_time* $> 7.5 \wedge$ *lead_time* ≤ 228.5 (recall 0.846, precision 0.203) captures a broad mid-range of bookings where the model struggles. This again is useful to trigger additional checks, while enriching features with seasonality or customer characteristics to improve prediction. For the job change prediction of data scientists, the rule *(city_development_index* $\leq 0.632 \wedge$ *training_hours* $\leq 99.5) \vee$ *experience* ≤ 5.5 (recall 0.684, precision 0.325) indicates the model is less reliable for low-experience/low-CDI candidate applicants. This could indicate to gather more data from these segments, and run fairness checks to ensure correlated socioeconomic variables are not inducing systematic bias. Finally, for the heart failure data set, *time* $\leq 132 \vee$ *age* $> 68 \vee$ *serum_creatinine* > 2.7 (recall 0.87, precision 0.465) is clinically sensible (early time points, elderly patients, kidney dysfunction) and marks cases where a cautious workflow might be warranted. This could require clinician confirmation, incorporating lab results, and enriching training data for these cohorts.

In summary, the rules constructed by MMDFAST are both interpretable and actionable: they highlight where additional checks are needed, where targeted data collection can reduce uncertainty, and where adapting the model (e.g., via specialized sub-models) may improve robustness.

B. Generalization of k-Influential Features

A central design choice in MMD+FR and MMDFAST is to restrict rule learning to the top-k features that most strongly drive mispredictions. Throughout this paper we have

used k=3, since our ablation study (see Figure 5) shows that this value provides the best trade-off between efficiency and predictive performance: smaller k values (e.g., k=2) reduce recall, while larger k values yield only marginal accuracy gains but increase runtime and rule complexity.

An important question is how well this choice generalizes when data sets contain many potentially influential features. Our experiments across 11 subjects with feature counts ranging from 9 to 100 suggest that the dominant error patterns are usually concentrated in a handful of features. For example, in the SE data sets, rules almost always involve commit size and parallel file changes, regardless of the total number of available features. This stability indicates that focusing on a small k captures the main conditions under which models fail, even in high-dimensional settings. Further, even in subjects without a pronounced long-tail distribution of feature importances (e.g., *water* and *spam*), MMDFAST still achieved significant accuracy gains over MMD, demonstrating that the approach remains effective beyond clear power-law cases.

That said, we do not claim k=3 is universally optimal. In domains with highly entangled or weakly informative features, increasing k may be necessary to cover more nuanced error modes. Our framework is flexible: practitioners can tune k based on their tolerance for runtime overhead versus the need for completeness.

C. Comparison with Other Approaches

For evaluating MMDFAST, we selected MMD as the baseline due to its established methodology and comprehensive framework for generating misprediction explanations, making it a suitable reference for assessing improvements in computational efficiency and rule quality. A closely related approach is the work by Gesi et al. [13], which also leverages feature bias for feature selection. However, a direct comparison was not possible since their research artifact [16] includes only evaluation data and not the necessary tools. Despite this limitation, we evaluated MMDFAST on the data sets provided by Gesi et al., as this ensures consistency with prior work, demonstrates versatility across domains such as software engineering and marketing, and provides a basis for future benchmarking. In summary, while a direct comparison with Gesi et al. was not possible, evaluating MMDFAST on their data sets still demonstrates its applicability across diverse domains and strengthens the credibility of our results.

IX. Limitations

A. Balance Precision, Recall, and Rule Length.

A fundamental challenge in generating misprediction explanations is balancing *Precision*, *Recall*, and explanation length. Focusing heavily on two of these characteristics inevitably worsens the third: (i) Maximizing *Precision* and *Recall* leads to long, overfitted rules that are complex and less interpretable. (ii) Maximizing *Precision* while keeping rule sets short results in low *Recall*, missing many mispredicted instances. (iii) Maximizing *Recall* while keeping rule sets short generally causes low *Precision*, leading to many false positives.

Adjusting the weights of these characteristics based on specific use cases is crucial. Each option has its advantages and disadvantages. MMD consistently provides concise explanations by considering rule length, but this approach can be computationally expensive and may not always satisfy desired coverage. Our approach can produce misprediction explanations quickly and often with better accuracy. However, without fine-tuning, decision trees tend to generate longer rule sets, as they may add high *Precision* but low *Recall* rules.

B. Feature Importance Measure

To identify the most influential input features, an explainable machine learning technique is necessary. We evaluated three methods: (i) Feature importance based on feature permutation, (ii) Feature importance with SHAP [7], and (iii) the impurity-based feature importance (i.e., Gini impurity).

Initial experiments showed that SHAP values performed slightly better than impurity-based importance. However, the significant increase in execution time made SHAP impractical. Impurity-based importance, despite its biases, was the most feasible given our constraints. Permutation importance resulted in poorer performance even after experimenting with different permutations. Therefore, we settled on impurity-based feature importance. Nonetheless, our tools would benefit from a faster and more reliable feature importance measure. Future work could explore advancements in feature importance techniques that balance accuracy and computational efficiency.

X. Related Work

The work presented in this paper relates to approaches that detect, explain, and debug faults in machine-learning systems:

A. Testing of Machine Learning Components

There is a significant body of work related to the testing of machine learning components [17]–[19]. The basic problem of most of these approaches is the construction of test inputs and providing an oracle to check if the test inputs are handled correctly. For the construction of test inputs, traditional software engineering techniques like grammar-based test-case generation [20], fuzz testing [21], [22], concolic testing [23], or search-based testing [24], [25] have been adopted. Another line of research for testing machine learning models is to utilize adversarial examples [26]. Test case generation approaches based on adversarial examples are used in different application domains, such as image recognition [27]–[29], text recognition [29]–[32], and speech recognition [33]–[36].

To tackle the oracle problem, metamorphic relations and metamorphic testing have been adapted for testing machine learning systems [37]–[39]. Besides metamorphic relations, domain-specific oracles can also be used. An example is the critical behavior like exceeding the speed limit or crashing of an Advanced Driver Assistance System (ADAS) [40], [41].

However, despite all the significant advances in testing ML components, merely providing a failed test case does not help the developers of machine learning components understand

and fix the problem. This is why we have developed MMDFAST, which can be used to refine and cluster failed test cases to present suitable debugging information to the developer.

B. Interpreting of Machine Learning Components

Interpretable machine learning is crucial as algorithms impact significant decisions and enter safety-critical systems. Molnar provides a comprehensive overview in his book [42] and paper [43]. Research splits into local and global interpretability: local explains individual predictions, while global addresses model behavior. Key local techniques include LIME [8] by Ribeiro et al. and SHAP [7] by Lundberg and Lee, which integrates LIME with Shapley Values [44]. However, these have drawbacks; LIME's stability issues are noted in [45], and [46] discusses exploiting LIME and SHAP to hide biases. Improvements include stability indices [47], ALIME [48], MeLIME [49], OptiLIME [47], and LIME-SUP [50]. For global interpretability, SHAP, GALE [51], and DENAS [52] identify influential features. Few studies, like MMD [9], address the ground truth and conditions for model mispredictions. Our work with MMDFAST draws from these approaches, particularly from interpretable machine learning, to extract the most impactful features. However, unlike these interpretable machine learning approaches, we do not aim to explain the entire behavior of the machine learning component. Instead, for debugging purposes, we focus specifically on the behavior where the machine learning component exhibits a failure or misprediction.

C. Debugging of Machine Learning Components

More in line with our work to debug machine learning components and identify groups of data causing mispredictions is Errudite [53], proposed by Wu et al., which conducts error analysis for NLP models. Errudite requires significant user involvement in manually formulating hypotheses and evaluating misprediction explanations. In contrast, our approach is mostly automated and can be easily adapted to create misprediction explanations for any underlying machine learner. Ma et al. proposed a technique to automatically repair neural networks called MODE [54]. They attempt to fix bugs by identifying the model's internal features causing incorrect results and then selecting tailored training data to address these bugs.

Cito et al. [9] introduced MMD, which utilizes rule induction to develop an interpretable set of rules linking specific feature values to potential inaccuracies in model predictions. With MMDFAST, we address the high computational demand and demonstrate that we can produce similar explanations faster. Like MMDFAST, Gesi et al. [13] focus on the same problem. They leverage feature bias to select important features to narrow down the feature count, whereas we consider feature importance to predict mispredictions.

A related line of work outside the ML-specific context is AVICENNA [55], [56], a tool to explain program failures. While AVICENNA targets general software debugging rather than ML components specifically, it shares the goal of producing interpretable, rule-like explanations for erroneous behavior.

XI. CONCLUSION

In this paper, we proposed MMDFAST, a efficient and general approach to explain the mispredictions of ML models. MMDFAST addresses two key issues of existing techniques: extensive computational demand and lack of accuracy. By identifying and focusing on the most influential features, MMDFAST streamlines the rule construction process and improves the accuracy and clarity of misprediction diagnoses.

We evaluated MMDFAST on 11 real-world data sets, demonstrating that our feature selection alone can reduce computational demand by up to tenfold without significant loss in accuracy. Additionally, incorporating our improved rule construction algorithm, MMDFAST produces more accurate explanations and further reduces computational demand. With MMDFAST, developers can obtain better diagnoses in seconds compared to hours required by traditional methods.

Overall, MMDFAST enhances the scalability, efficiency, and quality of generating misprediction explanations. Future work can build on our findings to further refine feature selection and optimize rule construction algorithms, ensuring more robust solutions for explaining ML model mispredictions.

Rule Performance Measures Current rule performance measures, such as the weighted *F-Score* used by our approach and the *Precision-Recall-Length* weighting in MMD, are somewhat basic. Developing more sophisticated metrics that consider additional characteristics like interpretability and robustness could improve explanation quality.

Improved Feature Selection Our approach uses impurity-based feature importance, which has known biases. Exploring more reliable and computationally efficient techniques like SHAP, or other model-agnostic methods, could enhance feature selection. This would also allow the use of more complex models beyond random forests.

Optimized Rule Generation Our decision tree-based approach shows promise even without parameter tuning. Further optimization, such as varying tree depths, using ensemble methods, or incorporating bootstrap aggregating (bagging), could improve rule quality.

XII. DATA AND TOOL AVAILABILITY

Our evaluated MMDFAST artifact is publicly available. The current versions of MMDFAST can be downloaded from:

https://github.com/martineberlein/mmdfast

XIII. ACKNOWLEDGMENTS

We thank Dennis Buchwinkler for his valuable contributions to this research on explaining model mispredictions. We also thank the anonymous reviewers for their constructive feedback and valuable suggestions. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under project Emperor (261444241).

REFERENCES

[1] S. Lee, Y. Kim, H. Kahng, S. Lee, S. Chung, T. Cheong, K. Shin, J. Park, and S. B. Kim, "Intelligent traffic control for autonomous vehicle systems based on machine learning," *Expert Syst. Appl.*, vol. 144, p. 113074, 2020. [Online]. Available: https://doi.org/10.1016/j.eswa.2019.113074

[2] A. Choudhury and D. Gupta, "A survey on medical diagnosis of diabetes using machine learning techniques," in *Recent Developments in Machine Learning and Data Analytics*, J. Kalita, V. E. Balas, S. Borah, and R. Pradhan, Eds. Singapore: Springer Singapore, 2019, pp. 67–78.

[3] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, and W. Cao, "A survey on theories and applications for self-driving cars based on deep learning methods," *Applied Sciences*, vol. 10, no. 8, 2020.

[4] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "Data cleaning: Overview and emerging challenges," in *Proceedings of the 2016 international conference on management of data*, 2016, pp. 2201–2206.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[6] M. Sayed-Mouchaweh and E. Lughofer, *Learning in non-stationary environments: methods and applications*. Springer Science & Business Media, 2012.

[7] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 4765–4774. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds. ACM, 2016, pp. 1135–1144. [Online]. Available: https://doi.org/10.1145/2939672.2939778

[9] J. Cito, I. Dillig, S. Kim, V. Murali, and S. Chandra, "Explaining mispredictions of machine learning models using rule induction," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 716–727. [Online]. Available: https://doi.org/10.1145/3468264.3468614

[10] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[11] M. Owhadi-Kareshk, S. Nadi, and J. Rubin, "Predicting merge conflicts in collaborative software development," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2019, Porto de Galinhas, Recife, Brazil, September 19-20, 2019*, 2019, pp. 1–11. [Online]. Available: https://doi.org/10.1109/ESEM.2019.8870173

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[13] J. Gesi, X. Shen, Y. Geng, Q. Chen, and I. Ahmed, "Leveraging feature bias for scalable misprediction explanation of machine learning models," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 1559–1570. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00135

[14] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[15] R. A. Disha and S. Waheed, "Performance analysis of machine learning models for intrusion detection system using gini impurity-based weighted random forest (giwrf) feature selection technique," *Cybersecurity*, vol. 5, no. 1, p. 1, 2022.

[16] J. Gesi, X. Shen, Y. Geng, Q. Chen, and I. Ahmed, "Leveraging feature bias for scalable misprediction explanation of machine learning models," 2023. [Online]. Available: https://github.com/Jirigesi/BGMD_MAPS

[17] H. B. Braiek and F. Khomh, "On testing machine learning programs," *J. Syst. Softw.*, vol. 164, p. 110542, 2020. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110542

[18] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5193–5254, 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09881-0

[19] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Trans. Software Eng.*, vol. 48, no. 2, pp. 1–36, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2019.2962027

[20] S. Udeshi and S. Chattopadhyay, "Grammar based directed testing of machine learning systems," *IEEE Trans. Software Eng.*, vol. 47, no. 11, pp. 2487–2503, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2953066

[21] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Deephunter: Hunting deep neural network defects via coverage-guided fuzzing," no. 12, pp. 146–157, 2019.

[22] H. You, Z. Wang, J. Chen, S. Liu, and S. Li, "Regression fuzzing for deep learning systems," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 82–94. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00019

[23] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Deepconcolic: testing and debugging deep neural networks," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 111–114. [Online]. Available: https://doi.org/10.1109/ICSE-Companion.2019.00051

[24] S. Kang, R. Feldt, and S. Yoo, "SINVAD: search-based image space navigation for DNN image classifier test input generation," in *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*. ACM, 2020, pp. 521–528. [Online]. Available: https://doi.org/10.1145/3387940.3391456

[25] ——, "Deceiving humans and machines alike: Search-based test input generation for dnns using variational autoencoders," vol. 33, no. 4, 2024, pp. 103:1–103:24. [Online]. Available: https://doi.org/10.1145/3635706

[26] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Computer Science*, 2014.

[27] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2016, pp. 372–387.

[28] C. Xie, Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, and A. L. Yuille, "Improving transferability of adversarial examples with input diversity," *CVPR*, pp. 2730–2739, 2019.

[29] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," *Commun. ACM*, vol. 62, no. 11, pp. 1–18, 2019.

[30] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," *arXiv*, pp. 2021–2031, 2017.

[31] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," pp. 31–36, 2018.

[32] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," pp. 50–56, 2018.

[33] N. Carlini and D. A. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," in *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*. IEEE Computer Society, 2018, pp. 1–7. [Online]. Available: https://doi.org/10.1109/SPW.2018.00009

[34] M. Cissé, Y. Adi, N. Neverova, and J. Keshet, "Houdini: Fooling deep structured prediction models," *CoRR*, vol. abs/1707.05373, 2017. [Online]. Available: http://arxiv.org/abs/1707.05373

[35] R. Taori, A. Kamsetty, B. Chu, and N. Vemuri, "Targeted adversarial examples for black box audio systems," pp. 15–20, 2019. [Online]. Available: https://doi.org/10.1109/SPW.2019.00016

[36] T. Du, S. Ji, J. Li, Q. Gu, T. Wang, and R. Beyah, "Sirenattack: Generating adversarial audio for end-to-end acoustic systems," in *ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, October 5-9, 2020*, H. Sun, S. Shieh, G. Gu, and G. Ateniese, Eds. ACM, 2020, pp. 357–369. [Online]. Available: https://doi.org/10.1145/3320269.3384733

[37] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, F. Tip and E. Bodden, Eds. ACM, 2018, pp. 118–128. [Online]. Available: https://doi.org/10.1145/3213846.3213858

[38] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, 2011. [Online]. Available: https://doi.org/10.1016/j.jss.2010.11.920

[39] X. Xie, Z. Zhang, T. Y. Chen, Y. Liu, P. Poon, and B. Xu, "METTLE: A metamorphic testing approach to assessing and validating unsupervised machine learning systems," *IEEE Trans. Reliab.*, vol. 69, no. 4, pp. 1293–1322, 2020. [Online]. Available: https://doi.org/10.1109/TR.2020.2972266

[40] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1016–1026. [Online]. Available: https://doi.org/10.1145/3180155.3180160

[41] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Testing autonomous cars for feature interaction failures using many-objective search," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, M. Huchard, C. Kästner, and G. Fraser, Eds. ACM, 2018, pp. 143–154. [Online]. Available: https://doi.org/10.1145/3238147.3238192

[42] C. Molnar, *Interpretable Machine Learning*, 2nd ed., 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[43] C. Molnar, G. Casalicchio, and B. Bischl, "Interpretable machine learning - A brief history, state-of-the-art and challenges," in *ECML PKDD 2020 Workshops - Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2020): SoGood 2020, PDFL 2020, MLCS 2020, NFMCP 2020, DINA 2020, EDML 2020, XKDD 2020 and INRA 2020, Ghent, Belgium, September 14-18, 2020, Proceedings*, ser. Communications in Computer and Information Science, vol. 1323. Springer, 2020, pp. 417–431. [Online]. Available: https://doi.org/10.1007/978-3-030-65965-3_28

[44] L. S. Shapley, "A value for n-person games," in *Contributions to the Theory of Games II*, H. W. Kuhn and A. W. Tucker, Eds. Princeton: Princeton University Press, 1953, pp. 307–317.

[45] J. A. Adebayo *et al.*, "Fairml: Toolbox for diagnosing bias in predictive modeling," Ph.D. dissertation, Massachusetts Institute of Technology, 2016.

[46] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, "Fooling lime and shap: Adversarial attacks on post hoc explanation methods," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 2020, pp. 180–186.

[47] G. Visani, E. Bagli, and F. Chesani, "Optilime: Optimized lime explanations for diagnostic computer algorithms," *arXiv preprint arXiv:2006.05714*, 2020.

[48] S. M. Shankaranarayana and D. Runje, "Alime: Autoencoder based approach for local interpretability," in *Intelligent Data Engineering and Automated Learning–IDEAL 2019: 20th International Conference, Manchester, UK, November 14–16, 2019, Proceedings, Part I 20*. Springer, 2019, pp. 454–463.

[49] T. Botari, F. Hvilshøj, R. Izbicki, and A. C. de Carvalho, "Melime: meaningful local explanation for machine learning models," *arXiv preprint arXiv:2009.05818*, 2020.

[50] L. Hu, J. Chen, V. N. Nair, and A. Sudjianto, "Locally interpretable models and effects based on supervised partitioning (lime-sup)," *arXiv preprint arXiv:1806.00663*, 2018.

[51] I. van der Linden, H. Haned, and E. Kanoulas, "Global aggregations of local explanations for black box models," *arXiv preprint arXiv:1907.03039*, 2019.

[52] S. Chen, S. Bateni, S. Grandhi, X. Li, C. Liu, and W. Yang, "Denas: automated rule generation by knowledge extraction from neural networks," in *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2020, pp. 813–825.

[53] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld, "Errudite: Scalable, reproducible, and testable error analysis," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 747–763.

[54] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: automated neural network model debugging via state differential analysis and input selection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 175–186.

[55] M. Eberlein, M. Smytzek, D. Steinhöfel, L. Grunske, and A. Zeller, "Semantic debugging," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, S. Chandra, K. Blincoe, and P. Tonella, Eds. ACM, 2023, pp. 438–449. [Online]. Available: https://doi.org/10.1145/3611643.3616296

[56] M. Eberlein, M. Raselimo, and L. Grunske, "Which inputs trigger my patch?" in *IEEE/ACM International Workshop on Automated Program Repair, APR@ICSE 2025, Ottawa, ON, Canada, April 29, 2025*. IEEE, 2025, pp. 3–10. [Online]. Available: https://doi.org/10.1109/APR66717.2025.00006