A Platform-Agnostic Framework for Automatically Identifying Performance Issue Reports with Heuristic Linguistic Patterns

Yutong Zhao*, Lu Xiao†, Sunny Wong‡
*yutongzhao@ucmo.edu, †lxiao6@stevens.edu, ‡sunny@computer.org
*University of Central Missouri, Warrensburg, MO, USA
†Stevens Institute of Technology, Hoboken, NJ, USA
‡Envestnet Inc., Berwyn, PA, USA

Abstract—Software performance is critical for system efficiency, with performance issues potentially resulting in budget overruns, project delays, and market losses. Such problems are reported to developers through issue tracking systems, which are often under-tagged, as the manual tagging process is voluntary and time-consuming. Existing automated performance issue tagging techniques, such as keyword matching and machine/deep learning models, struggle due to imbalanced datasets and a high degree of variance. This paper presents a novel hybrid classification approach, combining Heuristic Linguistic Patterns (*HLP*s) with machine/deep learning models to enable practitioners to automatically identify performance-related issues. The proposed approach works across three progressive levels: *HLP* tagging, sentence tagging, and issue tagging, with a focus on linguistic analysis of issue descriptions. The authors evaluate the approach on three different datasets collected from different projects and issue-tracking platforms to prove that the proposed framework is accurate, project- and platform-agnostic, and robust to imbalanced datasets. Furthermore, this study also examined how the two unique techniques of the framework, including the fuzzy *HLP* matching and the *Issue HLP Matrix*, contribute to the accuracy. Finally, the study explored the effectiveness and impact of two off-the-shelf feature selection techniques, *Boruta* and *RFE*, with the proposed framework. The results showed that the proposed framework has great potential for practitioners to accurately (with up to 100% precision, 66% recall, and 79% *F1*-score) identify performance issues, with robustness to imbalanced data and good transferability to new projects and issue tracking platforms.

Index Terms—software performance; software repository mining; automatic text classification; linguistic pattern

1 Introduction

Software performance is a critical quality attribute measured by the timeliness, responsiveness, and resource consumption of a system at run-time [1], [2], [3], [4], [5]. Performance issues can lead to severe consequences, including budget overrun, project delay, and market loss [1], [4].

Like other types of software bugs, performance problems are reported to developers via issue-tracking systems. Modern issue tracking systems support tagging a specific issue report as being performance-related (e.g., via a *"label"* in Apache's Jira issue-tracking system). This label allows practitioners to quickly retrieve and prioritize related problems. Literature in bug triage also emphasizes the crucial role of effectively tagging and assigning issue reports to appropriate experts [6], [7], particularly in large projects. However, performance issues are largely under-tagged because the manual tagging process is voluntary and laborious. In practice, for all four issue-tracking systems in this study (i.e., Apache's Jira, Bugzilla, Redmine, and Mantis Bug Tracker), the performance issues tagging rates are below 1% — when empirical studies [8] find that performance issues should be around 4% to 16%. This discrepancy suggests a gap in accurately identifying and tagging performance issues.

Automatic tagging of performance issue reports is ideal. However existing techniques are limited to achieving this goal. Simple techniques, such as keyword matching, tend to be inaccurate; while machine/deep learning methods struggle due to the imbalance of issue types for training. For example, our manual investigation, of around 2000 randomly selected issues from the aforementioned four issue tracking platforms, finds only 7% performance issues. With such a significantly unbalanced dataset, machine/deep learning models tend to miss the few positive cases — leading to high precision and low recall.

The motivation behind automatic tagging performance issue reports unfolds in two aspects. First, it could benefit researchers who focus on real-world performance issues. There have been increasing efforts to support reproducible research with shared benchmark datasets. For instance, the security community has established a CVE database of real-world vulnerabilities, which profoundly supports security research. However, such a database is currently missing for software performance research, which is particularly challenging to reproduce due to the complications from the combination of internal factors and environmental factors of systems. An automatic tagging approach facilitates the establishment of such a benchmark database. Second, it could also benefit performance engineering in practice, especially for projects emphasizing performance. According to an expert performance engineer with more than 20 years

of related experience [9], performance is often treated in a "firefighting mode", but big companies have been establishing dedicated teams to "shift-left" in performance engineering — i.e. treating performance upfront and in different phases of the development process. However, performance experts incur much higher operational costs than general software engineers. Therefore, an automatic tagging approach can pre-screen massive issue reports, narrowing down the search scope for performance experts. Also for open source communities, such as Apache, an automatic tagging approach could save manual effort if the projects were to identify the untagged performance issues, given the gap between the actual tag rate (below 1%) and the anticipated rate (4% to 16%) of performance issues.

This paper presents a novel, hybrid classification framework, to automatically tag performance-related issue reports, based on linguistic analysis of issue description. This approach stems from the observation that performance-related issues often contain similar linguistic characteristics at the sentence level. By manually learning from 980 real-life performance issue reports tagged by developers in Apache's JIRA issue tracking system, we derived a set of 80 *HLP*s that recur in these issue reports and capture common linguistic features of how performance issues are described. Admittedly, this was a labor-intensive process of approximately 762 human hours, cross-validated among four taggers to ensure reliability [10]. However, this manual effort is an up-front, one-time investment. Once derived, the HLPs serve as the "secret sauce" for automatically identifying more, new performance issues from different projects and across different platforms. As we will show in the evaluation, the effort for deriving the HLPs cannot be substituted by existing large language models (LLMs), as our framework significantly outperforms cutting-edge LLMs, including ChatGPT-3.5 and ChatGPT-4.0.

Empowered by the *HLP*s, the proposed framework is designed with a three-level classification structure to achieve high accuracy. Namely, the first level classification is for **Fuzzy *HLP* Matching**, which is newly added with this journal extension, to automatically and fuzzily match more *HLP* variations. In other words, this layer builds upon the 80 manually derived *HLP*s as the training dataset to identify new *HLP* variations that are not observed and tagged manually using machine learning methods. Next, the second level classification is for **Sentence Tagging**, where we classify if a given sentence is related to performance or not. In this level, the classification relies on an 80-dimensional vector, called *Sentence HLP Vector*, which is constructed from the output of the previous level and captures which *HLP*s are matched in the sentence. The third level classification is for **Issue Tagging**, which determines if a given issue report is related to performance or not. At this level, we concatenate an $N \times 80$ matrix from the *Sentence HLP Vector* from the previous level. N equals the number of sentences in the report. This matrix, namely *Issue HLP Matrix*, captures all the *HLP* features and the order in which they appear in the report. It serves as the learning feature for the final issue classification.

We evaluate our approach with six research questions. The first three RQs evaluate the effectiveness of our approach focusing on the accuracy (RQ1), robustness to imbalanced datasets (RQ2), and transferrability to new issue-tracking platforms (RQ3) of our framework, and compare it with the state-of-the-art baseline methods in the three aspects. Following this, RQ4 and RQ5 delve into our framework's unique techniques. Specifically, RQ4 evaluates the impact of fuzzy *HLP* tagger layer; while RQ5 evaluates the impact of *Issue Matrix* on the accuracy of the proposed framework. Lastly, RQ6 highlights the essentiality of all 80 *HLP*s and examines the impact of feature selection on our approach's accuracy and execution performance.

Our experiments with a total of 2,170 issue reports, containing a total of 13,197 sentences, collected over four different popular issue tracking platforms, including Apache JIRA, Bugzilla, Redmine, and MantisBT, have shown great potential of our framework in helping practitioners automatically identify performance issue reports. The results show that:

- Our framework achieves high issue classification accuracy with a precision up to 100%, recall up to 66%, and *F1*-score up to 79%, which outperforms cutting edge baseline approaches by up to 2.2 times in precision, 3.5 times in recall, and 2.8 times in *F1*-score.
- Our approach is robust to imbalanced data since the nature of data imbalancing has no significant impact on the accuracy (i.e. precision, recall, and *F1*-score) of our framework. In comparison, the *recall* of the baseline methods is compromised by up to 16% due to the imbalanced dataset with statistical significance.
- Our approach demonstrates its robustness when transferring to datasets gathered from different issue-tracking platforms, covering various application domains, without requiring additional pre-training. In comparison, the baseline methods are significantly compromised, especially in their precision (by up to 44% decrease) with statistical significance.
- In our approach, four models — NB, DT, CNN, and RNN — consistently showed substantial improvement in recall while ensuring a stable overall *F1*-score across various datasets when employing fuzzy *HLP* matching. Among these, the CNN model stands out as the top performer, making it our recommended choice for practitioners opting to use fuzzy *HLP* matching.
- Our approach benefits from the utility of the *HLP* Matrix. Specifically, it increases the upper bound *F1*-score by 9%, 5%, and 4% across the three datasets. Due to variations observed in our experiments, we recommend that practitioners take an exploratory approach with this feature. This could involve testing different dataset and model combinations to fine-tune and optimize accuracy effectively.
- In our approach, most *HLP*s are essential and retained during feature selection. Reducing the few *HLP*s only leads to marginal execution time improvements. Therefore, we recommend practitioners keep all 80 *HLP*s to maintain the comprehensive effectiveness of our method.

Finally, this work is a substantial extension to our prior work [10] in the following five aspects:

1) **Fuzzy *HLP* Matching:** This technique represents a significant advancement from the original two-layer classification structure to a more flexible three-layer framework. Unlike the strict regular expression-based matching of *HLP*s in prior work, fuzzy HLP matching uses machine learning models to identify potential matches of *HLP*s. This approach allows for greater flexibility in capturing a wider range of *HLP* variations, thus offering a valuable trade-off in four models — namely NB, DT, CNN, and RNN — as they consistently achieve the objective of substantially enhancing recall and ensuring a stable overall *F1*-score.
2) **Issue *HLP* Matrix:** Replacing the original Issue HLP Vector, the Issue *HLP* Matrix introduces the capability to recognize the sentence order in issue reports. This feature is crucial in capturing the contextual flow of information, which was not possible with the earlier vector-based approach. The impact of the *HLP* Matrix is sensitive to different models on different datasets, but the impact is generally positive. Despite the variation with datasets and models, the *HLP* Matrix feature increases the upper bound of the overall accuracy across the three datasets.
3) **Feature Selection Techniques:** In our experiment, most *HLP*s are essential and should be retained during feature selection, as reducing them leads to marginal execution time improvements. Therefore, we recommend practitioners keep all 80 *HLP*s to maintain the comprehensive effectiveness of our approach.
4) **Extended Evaluation for Platform and Domain Agnosticism:** We have expanded our evaluation to cover a wider range of issue-tracking platforms, including *Bugzilla*, *Redmine*, and *MantisBT*, in addition to Apache's JIRA. This extension, coupled with a **30%** increase in the evaluation dataset size, demonstrates the adaptability and practical applicability of our framework across various platforms and projects. In addition, we conducted an evaluation of our approach's accuracy across various software domains, categorizing issues from the datasets into six broad domains. Our findings reveal consistent accuracy across these domains, with average precision, recall, and *F1*-score remaining uniform. This consistency underscores our approach's versatility, affirming its effectiveness and suitability across a diverse range of software development environments, regardless of the specific domain involved.
5) **Comprehensive Evaluation:** Our thorough evaluation encompasses aspects like robustness to imbalanced data, transferability across projects and platforms, and the effectiveness of feature selection techniques. These evaluations were not part of our previous work and significantly contribute to demonstrating the practicality and effectiveness of the new techniques (fuzzy *HLP* matching, Issue *HLP* Matrix, and feature selection) in real-world applications.

The rest of the paper is organized as follows. Section 2 introduces background information. Section 3 details the proposed framework and the approach we take. Section 4 elaborates the evaluation design of this study. Section 5 presents the experiment results. Section 6 discusses the qualitative analysis of the RQs to complement the quantitative analysis, and the future directions. Section 7 discusses limitations and threats to the validity. Section 8 talks about related work. Section 9 concludes the paper. Section 10 provides data accessibility.

2 BACKGROUND

This paper tackles a text classification problem. We introduce the background information for this problem in this section.

2.1 Linguistic Patterns

Linguistic patterns are grammatical rules that enable their users to express themselves properly in a shared language [11]. Heuristic linguistic patterns have been used in previous research to classify text [12], [13]. For example, the pattern "As a [role], I want to [action], so that [benefit]" is a heuristic linguistic pattern used to describe user stories [14], which can be utilized to automatically match texts that describe user stories in a large corpus. In some cases, the classification is not definitive due to the fuzzy nature of the problem, and in such situations, researchers combine heuristic linguistic patterns with fuzzy logic [15], [16], [17]. Shi *et al.* [18], for example, combined linguistic patterns with fuzzy logic to classify issue report texts into various information types, including Intent, Benefit, Drawback, Example, Explanation, and Trivia. In their work, an input text could correspond to multiple information types without a definite answer. They assign a confidence value to each linguistic pattern to represent the degree of association between that pattern and an information type. The linguistic fuzzy model offers interpretability of the classification process [19], [20].

2.2 Machine/Deep Learning

Machine learning and deep learning models are commonly used for text classification [21]. The effectiveness of a classifier is dependent on the relevant features extracted from the data for training. A common approach is to transform input texts into a numerical representation in the form of vectors and matrices [22]. For example, the *Count Vector* is a matrix notation of a corpus where every row represents a document, every column represents a term, and every cell represents the frequency count of a particular term in a particular document [23]. However, calculating the frequency of terms can be problematic since all terms are considered equally important when assessing relevancy on a query. Hence, the *TF-IDF (Term Frequency - Inverse Document Frequency)* scales down the weights of terms with high collection frequency [24].

There are different levels of input tokens that can be used to generate *TF-IDF* scores, including *Word-Level TF-IDF* [25], *N-gram Level TF-IDF* [26], and *Character Level TF-IDF* [27]. *Word Embedding* is another form of representing words and documents using a dense vector representation [28]. It can be

generated using pre-trained embeddings such as *Glove* [29], *FastText* [30], and *Word2Vec* [31].

In addition to the feature extraction approach, the choice of machine learning models used can also impact the accuracy of classification. *Naive Bayes* [32], [33], [34], *Logistic Regression* [35], *Support Vector Machine* [36], [37], *Decision Tree* [38], *Random Forest* [39], and *Extreme Gradient Boosting* [40] are common machine learning models used for text classification.

In recent years, deep learning models, inspired by how the human brain works, have gained popularity in text classification. Two popular models are the *Convolutional Neural Networks (CNN)* and *Recurrent Neural Networks (RNN)*. The *Bidirectional Encoder Representations from Transformers (BERT)* model, which benefits from transfer learning, has been shown to outperform previous methods for a wide variety of natural language processing tasks [41], [42], [43]. In transfer learning, the model is first trained on a large text set to solve some general-purpose by training the model like language modeling and auto-encoding. This step, referred to as pre-training, prepares the deep learning model to rapidly learn new downstream tasks. In this work, we use token embedding as features for the *BERT* model.

2.3 Hybrid Approaches

Hybrid approaches combine a base classifier (i.e., the machine/deep learning model) with a rule-based system, to improve the classification [44]. These hybrid systems can be easily fine-tuned by adding specific rules for conflicting tags that have not been correctly modeled by the base classifier.

The proposed performance issue detection framework resides in this category: we train classic machine/ deep learning models with the *HLPs* derived learning features. To prove the advantage of our approach, we compare our approach with a comprehensive set of baseline methods, by combining different NLP features and machine/deep learning models discussed in the previous section. In our approach, the *HLP*s extracted from known performance issues provide more accurate learning features than the classic NLP features.

3 Study Approach

Our approach is inspired by the observation that performance-related issues often include sentences that share similar linguistic characteristics. We capture the common linguistic characteristics with *HLPs*, which we empirically derive from analyzing existing issue reports. Leveraging these linguistic patterns and machine learning techniques, we can classify new issue reports as performance-related or not. Therefore, our approach consists of two main phases: a preparation/learning phase to build the *HLP Set* and an issue classification phase. We first elaborate on linguistic patterns before detailing these two phases below.

3.1 Heuristic Linguistic Patterns

Heuristic linguistic patterns allow us to approximate whether some text relates to the topic of performance. An issue report that describes the system as *"running slow"* or *"inefficient"*, for example, would indicate that it likely concerns the system performance. These patterns offer a project-agnostic method for us to identify issue-report texts that describe the symptoms, root causes, solutions, and run-time measurements of performance issues. Our patterns operate at the sentence-level by analyzing the words and grammatical structure of a sentence. Following the approach of Shi et al. [18], we define four types of linguistic patterns: *lexical*, *profiling*, *structural*, and *semantic*.

Lexical linguistic patterns extend the basic keyword match concept by additionally considering synonyms, negated antonyms, and lemmatizations of that keyword/phrase. For example, an *efficiency* lexical pattern would include the such terms as *efficient*, *efficiency*, *inefficient*, and *inefficiency*. Another example, which we call the *infinite_loop* pattern, looks for *infinite*, *forever*, or *endless*; followed by *loop* or *iteration*.

When a performance issue is identified, developers often use a profiler to record performance characteristics of the system. We define *profiling linguistic patterns* to capture this information, as embedded in an issue report. Usually, the matching issue text contains a time or memory usage unit (e.g., milliseconds, megabytes), or the extent of performance change—measured in percentage terms, comparing the run-time parameters of before and after a code revision.

Structural linguistic patterns operate on a higher grammatical level of the sentence structure. These patterns can identify phrase structures that imply a performance issue. For example, we observe that "*when . . . run/execute/perform . . . for [NUMBER] seconds/minutes*" is a common way in different projects to describe performance issues that happen under a special input.

Semantic linguistic patterns extend lexical patterns by also incorporating sentiment analysis [45], to capture linguistic expressions that imply performance issues. For example, our *negative_necessary* pattern searches for a common way of describing the root cause or solution to performance issues that happen under unnecessary conditions. It searches for the word *necessary*, *required*, or *essential* in a sentence that has a negative sentiment.

In the following subsection, we detail how we derive our linguistic patterns for classifying performance-related issues.

3.2 Preparation/Learning Phase

The goal is to extract a comprehensive *HLP* set from known performance issue reports, which can be used to automatically tag performance issues in a new dataset. Figure 1 illustrates the key process of manually reviewing and deriving *HLP*s. The raw *input* is the developer-tagged performance issue reports on Apache's Jira issue-tracking system. The *output* is a comprehensive *HLP* set (highlighted in the yellow background in Figure 1). This phase contains five iterative steps, which are described in detail below. We chose Apache's projects for composing the datasets and building *HLP* Set due to their rich and accessible data on performance issues. Apache's vast, active open-source community offers well-documented, diverse projects with numerous developer-tagged performance issues. This variety provides a broad range of real-life software development challenges. Additionally, Apache's JIRA issue tracking system yields detailed performance issue records, including metadata crucial for our deep analysis of real-world performance issue management practices.

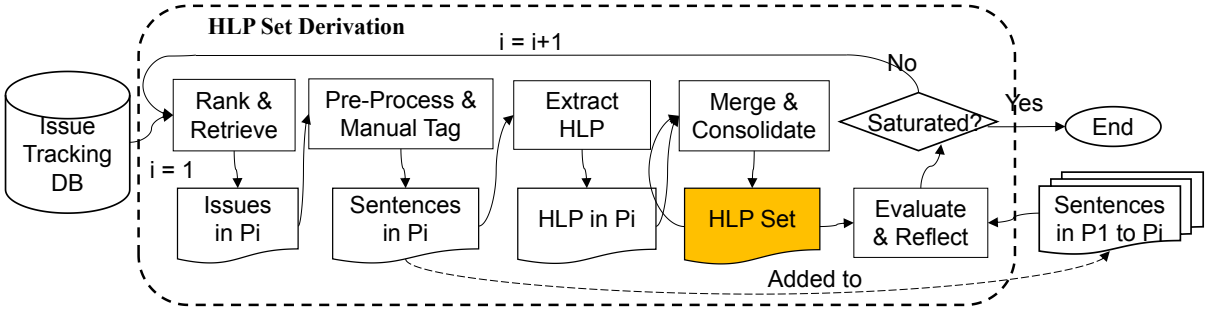


Fig. 1: Iterative *HLP Set* Derivation

3.2.1 Rank & Retrieve

We rank all the open source projects on the Apache Software Foundation in descending order based on the number of developer-tagged performance issues. On Apache Jira system, developers defined a special label, named "performance" to tag issues. In each iteration, i (starting from 1), we retrieve all the developer-tagged performance issue reports from the i-th project, namely P_i. In each iteration, issues from P_i serve as the source for us to learn how performance issues are described in practice. We start with projects with the largest number of tagged performance issues to accelerate the building process.

3.2.2 Pre-Process & Manual Tag

First, we pre-process the issue reports from project P_i. We use the *Stanford Core-NLP*[1] to break each issue report into sentences. For each sentence, we applied lemmatization, Part-Of-Speech (POS) tagging, Named-Entity-Recognition (NER) tagging, dependency parsing, and sentimental analysis using the annotators of *Stanford Core-NLP*. We apply standard data cleaning to remove stop words, such as *a*, *an*, and *the*.

Next, we manually tag each sentence in each issue as either performance-related or not. The reason is that a performance issue usually also contains many sentences that do not carry performance-related information. For example, developers may describe the general background information of an issue or include social notes. Thus, a sentence must contain a description relevant to performance problems, such as the symptoms, the causes, the optimization solutions, and performance profiling data, to be considered performance-related. The goal is to identify sentences that contain reusable information that can help identify similar issues in a different context.

A team of five, including a senior researcher, two senior Ph.D. candidates, a junior Ph.D. student, and a master's student with software development experience, collaborated on manual tagging. To minimize bias, the team was divided: one senior Ph.D. candidate and the junior Ph.D. student formed one group, while another senior Ph.D. candidate and the master's student formed the second group. The senior researcher mediated conflicts.

For Dataset 1 and Dataset 2, we re-used the tagging from our prior study [10], wherein the tagging processed has been detailed. For Dataset 3, the first group handled tagging in Bugzilla and Redmine, and the second group handled Redmine and MantisBT. Each group cross-validated their results, where the outcomes of Cohen Kappa are 89%, 85%, and 91%, separately.

3.2.3 Heuristic Linguistic Pattern Extraction

Based on the tagged performance-related sentences, we manually extract and summarize *HLPs* that can recur in different contexts for describing performance problems. This is a combination of automated and manual processes.

For each sentence that is manually tagged as performance-related in P_i, we identify the linguistic properties using the *Stanford Core-NLP*. First, we use the *Part-Of-Speech Tagger (POS Tagger)* to assign parts of speech tags to each word, such as noun, verb, adjective, etc. This helps us to extract lexical, structural and semantic patterns.

For example, *"load_nn"* is a lexical heuristic linguistic pattern, indicating that a sentence must contain keyword "load" or "loads" in the form of a noun, used in describing computation load(s). Similarly, *"nn_by_nn"* (structural heuristic linguistic pattern) captures issues like "pixel by pixel", "byte by byte", which is often used to describe a tedious computation process.

Meanwhile, we use *Named Entity Recogonizer Tagger (NER Tagger)* to capture specific terms for describing time or memory consumption of an issue. This helps to extract profiling *HLP*s such as *"Percentage"* (NER Tagger contains "PERCENT") and *"Duration"* (NER Tagger contains "DURATION") that describe the profiling measurements.

Furthermore, *Stanford Core-NLP* provides the dependency analysis of a sentence, and categorizes it in sentiment such as positive, neutral, and negative. This helps to extract semantic *HLP*s. Figure 2 shows the dependency analysis of a sentence in issue report *KAFKA-5512*. It also matches *"negative_necessary"* (a semantic *HLP*), since it has keyword *"unnecessary"* and its sentiment is categorized as negative.

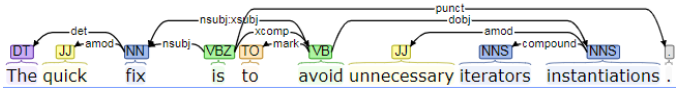


Fig. 2: An Example of Dependencies Analysis

By the end of this step, the output is a set of heuristic linguistic patterns from the project P_i.

1. https://stanfordnlp.github.io/CoreNLP/

3.2.4 *Merge & Consolidate*

We merge the *HLP* set from project P_i with the set built from previous iterations. The *HLP*s from P_i may overlap/duplicate with the existing *HLP* set. Thus, we merge overlapping, duplicating, or similar patterns together. For example, *"infinite_loop"* and *"loop_forever"* can be merged to one *HLP*, i.e. *"loop_infinite/forever*, which means that an infinite loop occurs. While merging, we also consolidate the merged heuristic linguistic pattern through divergent thinking. That is, we add possible variations of identified rules to be inclusive and predictive. For example, the example rule, *"loop_infinite/forever"*, above, can be extended to a more predictable rule, *"loop/iteration_infinite/forever"*.

3.2.5 *Evaluate & Reflect*

The last step of each iteration is to evaluate and reflect the updated *HLP* Set. First, we check whether the *HLP* Set has grown compared to previous iterations. If there are no (or only a few) new patterns added in the current iteration, it indicates that the *HLP* Set is (or close to being) saturated. Next, we evaluate the precision/recall of the *HLP* Set using the data from the processed projects. We use a naive matching approach: as long as a sentence matches one of the *HLP*s from the set, we consider it as performance-related. We calculate the precision/recall of this naive tagging by comparing it with our manual tagging. If the precision is low, it means that the *HLP*s can also frequently appear in non-performance-related sentences, implying that the *HLP* Set is irrelevant. If the recall is low, it means that the *HLP* Set is not comprehensive to capture all the different ways that performance issues are presented. As the *HLP* Set grows with iterations, the recall should increase gradually and stabilize when no more new rules can be found. We call this status as *HLP* set saturation, which means the *HLP* building is complete and no more iterations are needed.

3.2.6 *Heuristic Linguistic Pattern Saturation*

As reported in our previous study [10], it took 13 iterations (i.e. the process of iterative *HLP* set derivation illustrated in Figure 1) for us to reach *HLP* set saturation. We observed that the set grew rapidly in the first five iterations, but its growth slowed down after the sixth iteration. The set became stable at the 11th iteration. In total, we extracted 80 *HLP*s in the types of lexical (44%), structural (38%), semantic (10%), and profiling (8%). We evaluated the *HLP* set on the *HLP* building dataset and found that it could match performance-related sentences with a precision of 94% and a recall of 87%. These results indicate that the *HLP* set is saturated and can comprehensively and accurately capture the features of how performance issues are described based on the dataset for the *HLP* construction. We will build our issue tagging approach based on this *HLP* set.

Admittedly, even with *HLP* saturation based on the manual learning process, there is no guarantee that we have comprehensively captured all possible linguistic patterns of how developers may describe performance issues, due to the diverse nature of performance issues. As such, we are motivated to upgrade our approach from a two-level classification [10] to a three-level classification framework in this study. We added a first-level classification of *HLP* to automatically recognize potential variations of linguistic patterns based on machine learning models. We will explain this upgrade in the next subsection in detail.

3.3 Issue Report Classification Phase

Based on the constructed *HLP* set, we build our framework to automatically classify if an issue report is related to performance or not. Thus, the input to our framework is the text from an issue report; while the output is either "Yes" or "No", indicating whether this issue is related to performance problems. As shown in Figure 3, our framework works at three progressive levels for: 1) *HLP* tagging, 2) sentence tagging, and 3) issue tagging. Next, we explain the details of each level:

Fuzzy *HLP* Matching: In our previous version [10], our approach matches each of the extracted *HLP*s using strict regular-expression-based pattern matching to determine if a given sentence matches a certain *HLP* or not. However, as discussed earlier, although we reached saturation and cannot find additional new patterns using our learning dataset, we cannot guarantee that all possible variations of patterns, especially those that are unique to issue reports from a different platform, are also captured. As such, strict *HLP* matching is likely to ensure precision but may sacrifice recall. The newly proposed Fuzzy *HLP* Matching layer is motivated to increase recall without compromising the overall accuracy, as variations of *HLP* beyond the manually derived could also be identified through machine learning models.

We acknowledge that it is often more intuitive to aim for a higher precision, as it reduces manual effort to exclude false positives. However, the challenge of attaining high recall in scenarios characterized by data imbalance is well-known. Given the imbalanced nature of performance issues among all issue reports, it is intrinsically more challenging to achieve the recall, making the fuzzy matching technique a valuable asset. In addition, numerous literature have also underscored the importance of increasing recall in the context of bug report detection [4], [46], [47], [48]. More specifically, [46] underlines that performance bugs often remain undetected, making increased recall vital for early identification and resolution. [47] advocates the importance of high recall in statistical debugging for identifying potential faulty control paths. [48] advocates for improved recall in the early stages of performance debugging. Moreover, [4] reports that practitioners generally favor higher recall to prevent costly oversights in detecting performance bugs. The fuzzy *HLP* matching offers the option for practitioners to achieve a higher recall if that is what they prefer; while the strict *HLP* matching should be employed by those who prefer a higher precision. As we will elaborate in Section 4.1, RQ4 focuses on evaluating the impact of this fuzzy *HLP* matching on the precision, recall, and *F1*-score of our approach, compared to the strict matching mode.

In this level, we include 80 binary classifiers, each of which stands for "fuzzily" matching one of the 80 *HLP*s. For each classifier for pattern, HLP_i, the input is a sentence from an issue report. The output is "Yes" or "No" indicating whether this sentence matches HLP_i based on the machine learning model. We use the classic NLP features of the input sentence. More specifically, we pre-process the input

Study Overflow

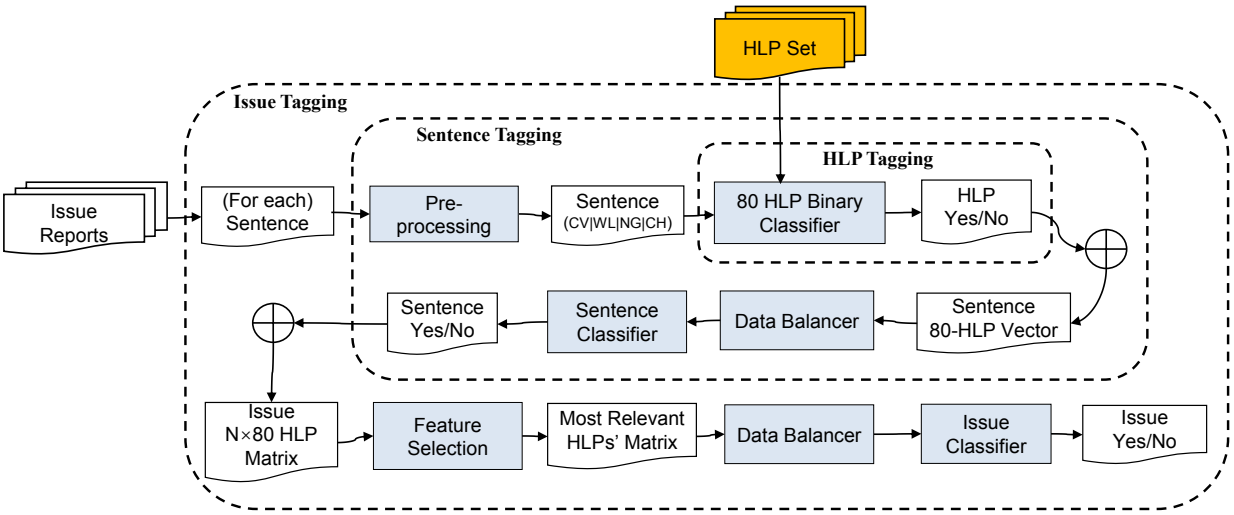


Fig. 3: Performance Issue Report Classification

For each RQ, the **average** precision, recall, and F1-score was calculated based on **100 times** execution of the related experiments.

sentence using *Stanford Core-NLP* to obtain the classic NLP features, including *Count Vectors (CV)*, *Word-Level TF-IDF (WL)*, *N-gram TF-IDF (NG)*, and *Character-Level TF-IDF (CH)*. By providing manually matched *HLP* as the training data, this level of classification can identify variations of *HLP*s that may have not been captured in the construction process. We envision that this level of classification serves to increase the recall of tagging performance issues.

Sentence Tagging: Next, we use the second-level classifier to determine if an input sentence is related to performance or not. The direct input to this level is a *Sentence HLP Vector*, resulting from the previous level of classification described above—the 80 binary classifiers for fuzzily matching the *HLP*s. The *Sentence HLP Vector* is a binary vector with n dimensions, where n is the total number of *HLP*s. More specifically, n is 80 in our study since we constructed 80 *HLP*s from our manual derivation phase. The i-th value in the vector is either 0 or 1, indicating whether the *HLP* tagging result from the previous level of the i-th *HLP* is "Yes" or "No". We use the *Sentence HLP Vector* as the learning feature to train machine/deep learning models. As such, given any sentence, we can use the trained model to determine if a sentence is related to performance problems.

Note that this level remains the same as our previous version [10], except that the input is from *HLP* classifiers in the first level instead of strict regular expression matching.

Issue Tagging: Finally, we perform the third level of classification, by taking the feature extracted from an entire issue report as the input, and output "Yes" or No," indicating whether this issue report is related to performance problems. In the previous version [10], we directly summed up the *Sentence HLP Vectors* of all sentences in an issue to form an *Issue HLP Vector*. However, the potential drawback of this treatment is that it will not capture the order of sentences, as such the context information behind the order of the sentences of the report will be missing. To overcome this limitation, in this study, we concatenate the *Sentence HLP Vectors* to construct an $s \times n$ matrix, where s equals the number of performance-related sentences and n equals the number of *HLP*s. This matrix is called the *Issue HLP Matrix*, which is used as the learning feature of machine/deep learning models to automatically identify performance-related issue reports.

As we will elaborate in Section 4.1, RQ5 focuses on evaluating the impact of introducing the *Issue HLP Matrix* on the overall performance of our approach, compared to using the *Issue HLP Vector* in our previous version [10].

Approach Variations: Our framework, illustrated in Figure 3, can be combined with different machine/deep learning models to provide variations of our approach. In this study, we experiment with six classic machine learning models and two advanced deep learning models, yielding eight distinctive variations of our framework, as listed in Table 1.

Note that our framework contains three levels of classifications, which requires three classifiers to work progressively together. That is, if we allow any combination of models to work at the three levels, we will yield $8^3 = 512$ variations of our approach. We keep the choice of model in the three classifiers to be consistent to eliminate the complexity. The motivation for this choice is to provide straightforward options for practitioners to operate with our approach.

At the *HLP* Tagging level, we incorporate four established NLP features, including: *Count Vectors (CV)*, *Word-Level TF-IDF (WL)*, *N-gram TF-IDF (NG)*, and *Character-Level TF-IDF (CH)*. These are used in conjunction with the six machine-learning models. Specifically, each of the six machine learning models is evaluated using each of the four NLP features. The feature yielding the highest *F1*-score value is selected as the input, which allows for optimal model accuracy. *Word Embedding* is employed as the input feature for the two deep learning models. At the Sentence Tagging level, the *Sentence HLP Vector* is utilized as input. Similarly, for the Issue Tagging level, the *IssueHLP Matrix* is employed as the input for each of the eight variations.

TABLE 1: Sentence/Issue Tagging Approach Variations

Type	Abbrev.	Model Name	Feature
HLP+ML	HLP+NB	Naive Bayes	Sentence *HLP* Vector OR Issue *HLP* Matrix
	HLP+LR	Logistic Regression	
	HLP+SVM	Support Vector Machine	
	HLP+DT	Decision Tree	
	HLP+RF	Random Forest	
	HLP+XGB	Extreme Gradient Boosting	
HLP+DL	HLP+CNN	Convolutional Neural Network	
	HLP+RNN	Long Short Term Memory RNN	

4 Evaluation Design

4.1 Research Questions

We evaluate our approach in six research questions. The first three RQs evaluate the effectiveness of our approach focusing on the accuracy (RQ1), robustness to imbalanced datasets (RQ2), and transferrability (RQ3). RQ4 and RQ5 investigate how the new techniques of fuzzy *HLP* matching (RQ4) and *Issue Matrix* for capturing sentence order (RQ5) contribute to our new framework, compared to the strict *HLP* matching and *Issue Vector*, respectively, in our original framework [10]. Lastly, RQ6 explores the essentiality of all 80 HLPs and examines the impact of feature selection on improving our approach's accuracy and execution performance. We elaborate the rationale behind each RQ in more detail below:

RQ1: What is the overall accuracy of our approach? How does it compare to the baseline methods? This RQ evaluates the tagging accuracy of our approach in identifying performance issue reports, in terms of its precision, recall, and *F1*-score, and how it compares to the baseline methods.

RQ2: How robust is our approach to imbalanced data? How does it compare to the baseline methods? As mentioned earlier, there is only 4% to 16% of performance issues in the issue tracking systems [4], [8]. It has been reported previously that data balancing could improve the accuracy of machine learning models [49], [50], [51]. This RQ investigates how robust is our approach to imbalanced data, especially how this compares to the baseline approach. That is, if we provide balanced data to train our approach, will its accuracy increase significantly? If so, it indicates that our approach is not robust to imbalanced data. In particular, we are interested to compare our approach with the baseline methods in terms of its robustness to imbalanced data.

RQ3: How well does our approach transfer to a new dataset from different issue-tracking platforms? How does it compare to the baseline methods? This RQ aims to evaluate whether our approach is project- and platform-agnostic. Given that the *HLP*s are built based on 980 JIRA issue reports from 13 Apache projects. We would like to evaluate whether our approach transfers well to issue reports of new projects, in particular, those from other platforms, such as Bugzilla, Redmine, and Mantis without requiring pre-training using new datasets.

RQ4: How does the fuzzy *HLP* matching impact our approach? We derived a total of 80 *HLP*s from 980 issue reports to reach a saturation status. However, there could be more patterns and variations of *HLP*s that are not captured comprehensively in this process. Thus, if we introduce a first layer of classification to "fuzzily" identify potential variations of *HLP*s, instead of using expression patterns to "strict-match" *HLP*s within sentences (as what we did in our original study [10]), how much will the accuracy of our approach change? The motivation behind the fuzzy *HLP* matching is to enhance the capability of our approach to identify potential matches of *HLP*s using machine learning models, thereby allowing for the detection of variation that might not strictly adhere to the 80 manually extracted of *HLP*s in our original study [10]. In other words, the overall accuracy of our approach should not be significantly compromised to allow the trade-off between precision and recall for practitioners when adopting the new fuzzy matching technique in our framework.

RQ5: How does the *Issue Matrix* feature, which captures the sentence order within an issue report, impact the accuracy of our approach? In our original framework [10], we use *Issue Vector* as the learning feature for issue level classification. However, the vector does not capture the context information among the sentences in an issue, i.e. their order. In this RQ, we evaluate whether and to what extent considering the sentence order through *Issue Matrix* as the learning feature in the issue-level classification will impact the accuracy of our approach.

RQ6: How do feature selection techniques impact the accuracy of and the required *HLP*s in our approach? In this RQ, we adopt two off-the-shelf feature selection techniques, namely *Boruta* and *Recursive Feature Elimination (RFE)*. The objective is twofold. First, we also aim to reveal whether and to what extent the 80 *HLP*s are truly necessary to our approach. Second, we aim to investigate whether and to what extent feature selection helps us to further improve the accuracy and execution performance of our approach.

4.2 Experiment Setup

This section talks about the evaluation datasets used in this study, our experimental design for answering the RQs, and the baseline methods that we compare our approach with.

4.2.1 Evaluation Dataset

We collected three different datasets as listed in Table 2, in preparation for evaluating our approach using datasets from different projects and from different issue-tracking platforms.

TABLE 2: Evaluation Datasets

ID	Name	Source Platform(s)	# Issues (P%)	# Sentences (P%)
1	Apache Homo.	*Apache Jira*	980 (8%)	4,790 (11%)
2	Apache Hetero.	*Apache Jira*	980 (6%)	5,371 (6%)
3	Other Platforms	*Bugzilla, Redmine, MantisBT*	210 (5%)	3,036 (4%)
Total			2,170 (6%)	13,197 (7%)

- **Dataset 1: Apache Homologous Data**. Collected to evaluate the efficacy of the *HLP* Set in identifying additional non-tagged performance issues, this dataset comprises 980 non-tagged issues and 4790 sentences. These issues were randomly selected in equal number to those used for building the *HLP* Set and are from the same projects used in the *HLP* Set construction. As a result, this dataset offers a nuanced view into the effectiveness of the *HLP* Set within a consistent project environment, revealing that only 8% of the issues and 11% of the sentences are performance-related.
- **Dataset 2** is for evaluating whether the *HLP*s are general for tagging issues from heterologous

projects, independent from the *HLP* building. To ensure a comprehensive evaluation, these issues were randomly selected from seven projects that are distinct from those used in Dataset 1. This approach was adopted to test the adaptability and applicability of the *HLP*s across different project environments. The dataset consists of 980 issues and 5371 sentences from these seven projects, not involved in the *HLP* building. Here, 6% of both issues and sentences are performance-related, providing a valuable perspective on the generalizability of our approach across diverse project contexts.

- **Dataset 3: Other Platforms Data**. It contains 210 issue reports with 3,036 sentences from Bugzilla, Redmine, and Mantis Bug Tracker. For Redmine and MantisBT, the issues were randomly selected to match the number of developer-tagged performance issues in these platforms. Of particular note, issue reports on Bugzilla only contain a single-sentence summary. Thus, the 210 issue reports (i.e 67 from Redmine and 143 from Mantis) do not count those from Bugzilla. Bugzilla data were included to augment the sentence-level tagging scope and were randomly sampled from its extensive issue database. But Bugzilla contributes 1,980 sentences that are only used for sentence-level tagging. This dataset is for evaluating whether our approach is platform-agnostic.

We followed the process in Section 3.2 to manually tag each sentence in the three datasets. For each dataset, we also manually tag each issue report, following a similar practice. Based on our manual tagging results, in Dataset 1, 8% of issues and 11% of sentences are performance-related; in Dataset 2, 6% of issues and 6% of sentences are performance-related; in Dateset 3, 5% of the 210 issues are performance-related, and 4% of the 3,036 sentences are performance-related. This underscores the challenge of automatically tagging these highly imbalanced datasets. Although only RQ3 specifically focuses on the transferability of our approach to different projects and to different platforms, we conduct the experiments of all other RQs on all three datasets to be comprehensive as well.

4.2.2 Comparison Baseline Methods

RQ1, RQ2, and RQ3 focus on evaluating the overall performance (RQ1), robustness to imbalanced dataset (RQ2), and transferability (RQ3) of our approach, respectively. In these RQs, we show the advantages of our approach in these three aspects by comparing it with a set of baseline methods. The baseline methods rely on the classic NLP features, such as *Count Vectors* and *Word Level TF-IDF*.

Table 3 shows 31 distinct variations of the baseline methods that are classified into three groups: 1) machine learning model-based methods (in the top part of the table); 2) deep learning model-based methods (in the middle part of the table); and 3) large language models (LLMs, in the bottom part of the table). Within the group of machine learning methods, a combination of six basic models and four distinct NLP features offers a total of 24 (i.e., 6×4) variations of the baseline methods. The deep learning model group offers five variations, specifically, the *BERT* model operating with the Token Embedding feature, and four other models leveraging the word embedding feature. The *LLM* group offers two variations, GPT-3.5 and GPT-4.0, operating with advanced transformer-based language models and utilizing dynamic word embeddings. GPT-3.5 utilizes a transformer-based language model architecture, and GPT-4 employs a more advanced version of this architecture, reflecting improvements in scale, efficiency, and language understanding capabilities. To rigorously assess GPT-3.5 and GPT-4.0 in identifying software performance issues, we adopted a straightforward experimental process. For each issue report in the three evaluation datasets, we engaged both ChatGPT versions with the prompt: "*Is the following description of an issue ticket related to software performance? If so, please tell me why.*" It reflects how end-users might interact with *LLMs* to classify and understand issue reports, without specialized prompt-engineering or fine-tuning. This process respects the user-friendly nature of *LLMs* and avoids potential bias introduced by intricate prompt engineering or parameter adjustments. To facilitate the experiment on the three datasets, we created a Python script to automate the interaction process with ChatGPTs.

TABLE 3: Comparison Baseline Methods

Baseline 1: ML Models + NLP Features		
Abbreviation	**ML Model**	**NLP Feature**
NB+CV|WL|NG|CH	Naive Bayes	Count Vectors (CV) Word Level TF-IDF (WL) N-gram Level TF-IDF (NG) Character Level TF-IDF (CH)
LR+CV|WL|NG|CH	Logistic Regression	
SVM+CV|WL|NG|CH	Support Vector Machine	
DT+CV|WL|NG|CH	Decision Tree	
RF+CV|WL|NG|CH	Random Forest	
XGB+CV|WL|NG|CH	Extreme Gradient Boosting	
Baseline 2: DL Models + NLP Features		
Abbreviation	**DL Model**	**NLP Feature**
BERT	BERT Classifier	Token Embedding
CNN	Convolutional Neural Network	Word Embedding
RNN-LSTM	Long Short Term Memory RNN	Word Embedding
RNN-GRU	Gated Recurrent Units RNN	Word Embedding
Bi-RNN	Bidirectional RNN	Word Embedding
Baseline 3: Large Language Models (LLMs)		
Abbreviation	**Transformer Architecture**	**NLP Feature**
ChatGPT-3.5	GPT-3.5	Word Embedding
ChatGPT-4	GPT-4	Word Embedding

4.2.3 Experiment Setting

All the RQs rely on the three classic information retrieval metrics, including precision, recall, and F1-score [52]:

- **Precision**, which measures the proportion of all positive classifications that are true positives.
- **Recall**, which measures the proportion of all the actual positive cases that are classified as positive.
- ***F1*-Score**, which is the harmonic mean of precision and recall. It measures the balance between precision and recall, which provides a single score that takes both precision and recall into account

For each RQ, whose setup will be elaborated in the next subsection, we ran each variation of our *HLP*-based approach and each variation of the baseline method 100 times repeatedly. Of particular note, RQ4, RQ5, and RQ6 focus on evaluating the impacts of specific features with our approach and thus comparison with the baseline methods do not apply to them. The precision, recall, and *F1*-score of each experiment were calculated as the average of the 100 repetitions. We use the three datasets introduced in Section 4.2.1 in our experiments. For each dataset, we

randomly divide the dataset into 70% for training and the remaining 30% for testing.

All the evaluation experiments are conducted on a machine equipped with a 12th Gen Intel Core i7-12800H CPU, 32 GB of RAM, an SSD, and an NVIDIA RTX A1000 Laptop GPU, providing insight into the practical application of our approach.

4.2.3.1 **RQ1: Overall Accuracy**: We compare the distribution of the accuracy of the two groups of approaches: 1) **Our Approach (*HLP*):** encompassing the eight variations of our *HLP*-based approach detailed in Section 3.3; and 2) **Machine and Deep Learning Models**, which include 29 distinct variations of methods based on machine learning and deep learning, leveraging traditional NLP learning features such as *TF-IDF*; and 3) **Large Language Models**, comprising 2 variations, GPT-3.5 and GPT-4, that utilize dynamic word embeddings.

Firstly, we examine and compare the distribution of three metrics (i.e. precision, recall, and *F1*-score) of the our approach and the baseline methods. Next, to quantify the statistical significance of the different distributions, we employ *Cliff's Delta* value for measuring the effect size of the comparison. *Cliff's Delta* effect size is a non-parametric measure, which evaluates the probability that a randomly selected value from one group will be larger than a randomly selected value from another group, minus the reverse probability [53]. The mathematical formula for Cliff's Delta is listed below:

$$\textit{Cliff's Delta} = \frac{N_{>} - N_{<}}{n_1 \cdot n_2} \tag{1}$$

where:

- $N_{>}$ is the number of pairs where a value from the first group is greater than a value from the second group;
- $N_{<}$ is the number of pairs where a value from the second group is greater than a value from the first group;
- n_1 and n_2 are the sample sizes of the first and second groups, respectively.

Cliff's Delta value ranges from -1 to 1, indicating the magnitude of difference between two groups, with 0 signifying no difference. Effect size thresholds for *Cliff's Delta* are classified as follows: a value less than 0.147 indicates a negligible effect size; between 0.147 and 0.33, a small effect size; between 0.33 and 0.474, a medium effect size; and greater than 0.474, a substantially large effect size [54]. In our study, we adopt 0.474 as the threshold for a substantial large effect size, following the guidelines from existing research, where an absolute *Cliff's Delta* (|*Cliff's* Δ|) value greater than 0.474 denotes significant differences between two groups.

4.2.3.2 **RQ2: Robustness to Imbalanced Data**: To investigate the robustness of our proposed *HLP* approach in handling imbalanced data, and compare such against the baseline methods, we undertake a dual set of experiments: 1) **Balanced Training Dataset (BT)**, which is controlled and composed of 50% performance-related sentences or issues and the remaining half are not performance-related. We tried composing such *BT* through random sampling from our evaluation dataset, as well as using the Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic samples of the minority class. The experiments show that both synthetic methods lead to the same conclusion in terms of our approach's robustness. And 2) **Imbalanced Training Dataset (IBT)**, which has an equal quantity of sentences and issues sampled from the untouched, original dataset. For the testing dataset, we maintain the percentage of performance-related data commensurate with that in the original dataset, since the imbalanced nature tends not to change in practice.

Similar to RQ1, we plot the accuracy distribution of our approach with *BT* and *IBT* as the training data, as well as that of the baseline approach. As our objective is to evaluate the robustness of our *HLP* approach and the baseline methods in handling imbalanced data, the comparison focuses on whether and to what extent the accuracy changes when transiting from *IBT* to *BT* with our *HLP* approach vs. with the baseline approach.

For statistically rigorous evaluation, we employ three metrics, 1) the average change — denoted as ch, 2) absolute *Cliff's Delta* effect size — denoted as |*Cliff's* Δ|, and 3) *P-Value* — denoted as p, to evaluate the impact of data imbalance on the precision, recall, and *F1*-score on our approach. The ch measures the difference between the average accuracy of using *IBT* vs. that of using *BT* as the training dataset on an approach. A positive value indicates increased accuracy using balanced data—meaning the approach is potentially not robust to data imbalance, which will be statistically confirmed by the two additional metrics, d and p.

The *Cliff's Delta* effect size (used in RQ1 as well) quantifies whether transiting from *IBT* to *BT* impacts an approach substantially. In the context of this RQ, transiting an approach from *IBT* to *BT* with absolute *Cliff's Delta* values (|*Cliff's* Δ|) $<= 0.474$ indicates robustness to data imbalance with no substantial change to accuracy.

The p from the *Mann-Whitney U Test* tests whether transiting from *BT* to *IBT* has a statistically significant impact. It is a non-parametric test to compare differences between two independent groups, particularly when the dependent variable is either ordinal or continuous and not normally distributed. Despite the availability of other similar tests, it is most appropriate based on the nature of our data. A p threshold of less than 0.05 indicates statistical significance [55]. In the context of this RQ, transiting an approach from *IBT* to *BT* with $p >= 0.05$ indicates robustness to data imbalance, as it means that from *IBT* to *BT* do not yield to statistically significantly different accuracy with an approach.

Note that based on the construction, the *Cliff's Delta* threshold poses a more strict condition than that of the *Mann-Whitney U Test P-Value* (p). In other words if |*Cliff's* Δ| > 0.474, p is always < 0.05 — meaning if the change is substantial, it must be statistically significant. However, a statistically significant change is not necessarily substantial.

4.2.3.3 **RQ3: Transferability**: To investigate the transferability of our approach to new datasets and new issue-tracking platforms, we conduct and compare two sets of experiments: 1) **D1/D1**: Both the training dataset and the testing dataset are from **Dataset 1** (introduced in Section 4.2.1). This experiment represents the accuracy of our

approach before the transfer. And 2) **D1/D3**: The training dataset still comes from **Dataset 1**; while the testing dataset comes from **Dataset 3** (introduced in Section 4.2.1 as well), which contains performance issue reports from different issue tracking platforms. This experiment represents the accuracy of our approach after transferring it to a completely different dataset and platforms without performing any pre-training using new datasets.

The evaluation of RQ3 follows the same structure as that of RQ2 — i.e. RQ2 evaluates the impact of transiting from *IBT* to *BT*; while, analogously, RQ3 evaluates the impact of transferring from *D1/D1* to *D1/D3*. Therefore, likewise, we plot the accuracy distribution of our approach with the transfer, as well as that of the baseline approach. Next, we use the three metrics, ch, *Cliff's* Δ, and p to reason whether the impact of transfer is significant and substantial. The objective of RQ3 is to demonstrate that our approach would not be significantly impacted by the transfer (i.e. $p >= 0.05$); while the baseline would be significantly (i.e. $p < 0.05$) or even largely ($|$*Cliff's* $\Delta| > 0.474$) impacted by the transfer due to the lack of essential domain knowledge — i.e., the *HLP*s.

4.2.3.4 **RQ4: Impact of fuzzy *HLP*-matching**: This RQ investigates how fuzzily matching *HLP*s (introduced in Section 3.3) — which replaces the original strict *HLP* matching — impacts the accuracy of our approach. Firstly, similar to the above RQs, we first plot the accuracy distribution of our approach when using fuzzing matching vs. using its strict matching counterpart. The objective is to gain an overall and intuitive understanding of the impact of fuzzy matching — particularly, as discussed earlier, the objective of fuzzy matching is to increase recall without compromising the overall *F1*-score.

More importantly, we conjecture that different models may be impacted differently by the fuzzy matching technique. We aim to gain a clear understanding of which models benefit most from this fuzzy matching technique for the objective of boosting recall and holding up *F1*-score. As such, practitioners can pick the most suitable models when adopting the fuzzy matching technique. Thus, we further provide statistically rigorous and fine-grained analysis of the impact of the fuzzy matching technique on each of the machine/deep learning models integrated with our framework (as listed in Table 1).

For each model, we use the same metrics as previous RQs, i.e. the average change — ch, absolute *Cliff's Delta* — $|$*Cliff's* $\Delta|$, and *P-Value* — p, to evaluate the impact of fuzzy matching on the precision, recall, and *F1*-score on our approach. Again, the c determines what is the impact (i.e. negative or positive); the $|$*Cliff's* $\Delta|$ measures if the impact is substantial (i.e. $|$*Cliff's* $\Delta| > 0.474$); p evaluates if the impact is statistically significant (i.e. $p < 0.05$). Our ultimate objective is to recommend models that meet two conditions: *C1:* substantially increase recall and *C2:* no significant compromise to *F1*-score. We formally define *C1* and *C2* in the following:

- C1: For the recall, $(ch > 0) \wedge (|$*Cliff's* $\Delta| > 0.474)$.
- C2: For the *F1*-score, $\neg(ch < 0 \wedge p < 0.05)$.

Of a particular note, we test each model against the three evaluation datasets (see Table 2). As such, we aim to identify models that consistently meet both *C1* and *C2* on all three datasets. Such models are reliably compatible with fuzzy matching to offer higher recall and thus should be recommended to practitioners.

4.2.3.5 **RQ5: Impact of *Issue HLP Matrix***: This RQ focuses on evaluating how the incorporation of the sentence order in a matrix (Section 3.3) influences the accuracy of our approach. Similar to previous RQs, we first plot the accuracy distribution of our approach using the *Issue HLP Matrix* feature, as well as that of using the *Issue HLP Vector* [10]. This provides an overall comparison between the two features.

Furthermore, similar to RQ4, we aim to gain a fine-grained understanding of how the *HLP* Matrix feature is compatible with different learning models on different datasets. A key hypothesis here is that the impact of adopting *HLP* Matrix may vary depending on the characteristics of the dataset, regarding which we will elaborate on a qualitative discussion in Section 6. Here, for each of the three accuracy measures, namely precision, recall, and *F1*-score, we distinguish three levels of impact using two metrics from RQ4, namely 1) the average change (ch), and 2) the absolute *Cliff's Delta* ($|$*Cliff's* $\Delta|$) value:

- $I_{void} = (|$*Cliff's* $\Delta| <= 0.474)$, which means there is no substantial impact on the checked accuracy measure.
- $I_{+} = (ch > 0) \& (|$*Cliff's* $\Delta| > 0.474)$, which means there is a substantial increase in the checked accuracy measure.
- $I_{-} = (ch < 0) \& (|$*Cliff's* $\Delta| > 0.474)$, which means there is a substantial decrease in the checked accuracy measure.

Note that we did not check the *P-Value* (p) here because of two considerations: 1) if $|$*Cliff's* $\Delta| > 0.474$, p is always < 0.05; and 2) we aim to highlight models that are *substantially* impacted by the *HLP* Matrix for practitioners, as if the impact is trivial, it does not justify adopting it. Similar to RQ4, we evaluate the impact of *HLP* Matrix on the three different evaluation datasets listed in Table 2 and distinguish the impact into the above three levels.

4.2.3.6 **RQ6: Impact of Feature Selection**: In this RQ, we investigate whether it is possible to employ feature selection techniques to trim unnecessary *HLP*s and thus increase the execution performance of our framework. To this end, we utilized two established feature selection methods: *Boruta* and *Recursive Feature Elimination (RFE)*. *Boruta* uses a random forest model to evaluate the importance of each feature against randomized shadow features. In contrast, *RFE* iteratively removes the least significant features from the dataset until reaching the desired feature count.

We assess the statistical difference between before and after adopting a feature selection technique using the same metrics as prior RQs, namely *Average Change* (ch), the absolute *Cliff's Delta* ($|$*Cliff's* $\Delta|$) value, and *P-Value* (p). Additionally, we also report S-*HLP*, which is the number of *HLP*s retained by the feature selection methods. This metric reflects the reduction in feature size. Furthermore, we also report the actual change in the execution time (denoted as t_δ) running our approach before and after the feature selection. It measures the actual improved performance efficiency after feature selection.

5 Evaluation Results

This section answers the six research questions.

5.1 RQ1: Approach Accuracy

Figure 4 shows the results of our experiments on the three different evaluation datasets in the three sub-figures. For example, Figure 4a shows the distribution of precision, recall, and *F1*-score of our approach based on Dataset 1 (Apache Homo), and how it compares to that of the machine and deep learning model-based baseline methods, in the left-lane, mid-lane, and right-lane, respectively. As we can see here, the precision of our approach ranges from 73% to 100% when using different machine/deep learning models; while the precision of the baseline methods ranges from 17% to 70% with different learning models. Similarly, the recall and *F1*-score of our approach both significantly outperform the baseline methods. We can make consistent observations based on all three datasets that our approach significantly outperforms the baseline methods with an average improvement of 54% in precision, 35% in the recall, and 41% in *F1*-score, across the three datasets. Furthermore, the calculated *Cliff's Delta* values for *F1*-score between our approach and all baseline methods, across these datasets, stand at 0.972, 0.875, and 1.0 respectively, which indicate a significant effect size (> 0.474), underscoring the substantial advantage of our approach over the baseline methods.

Particularly, Table 4 highlights the accuracy of the state-of-the-art *Large Language Models (LLMs)*, including ChatGPT-3.5 and ChatGPT-4. The precision and recall metrics for both *LLMs* fall notably short of our approach, revealing their limitations in capturing technical details pertinent to software performance.

TABLE 4: Issue Tagging Accuracy of ChatGPT-3.5 and ChatGPT-4

Dataset	ChatGPT-3.5			ChatGPT-4		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Dataset-1	25.13%	63.64%	36.03%	26.07%	79.22%	39.23%
Dataset-2	26.39%	34.55%	29.92%	33.57%	87.27%	48.49%
Dataset-3	12.50%	50.00%	20.00%	21.33%	60.00%	31.47%

RQ1 Takeaway Message: Our framework achieves a precision up to 100%, recall up to 66%, and *F1*-score up to 79% in automatically identifying performance issue reports. Our observations across all three datasets consistently demonstrate that our approach substantially outperforms baseline methods, with an average improvement of 54% in precision, 35% in recall, and 41% in *F1*-score.

5.2 RQ2: Impact of Data Imbalance

Figure 5 depicts the distribution of the precision, recall, and *F1*-score for our approach with *BT* and with *IBT* (the two blue bars in each lane) as the training data. We also show the same information for the baseline methods (the two yellow bars in each lane). We show the results across three datasets: Apache Homo (Figure 5a), Apache Hetero (Figure 5b), and Other Platforms (Figure 5c).

Figure 5 visually presents an intuitive impression of whether and to what extent transiting from *IBT* to *BT* impacts our *HLP* approach and the baseline approach, respectively. We can observe from all three sub-figures that the recall of the baseline approach is obviously improved when trained using *BT* compared to trained with *IBT*, indicating the lack of robustness to data imbalance with the baseline approach. On the contrary, our *HLP* approach overall shows quite stable accuracy in all metrics with the transition, indicating its overall robustness to data imbalance.

TABLE 5: Impact of Data Balancing (RQ2)

Metric	*HLP*: IBT→BT			BLM: IBT→BT		
Dataset 1: Apache Homo.						
	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value
Precision	+2.4%	0.297	0.3428	-5.8%	0.198	0.5076
Recall	+6.1%	0.250	0.4299	+16%	0.753	0.0079
***F1*-Score**	+5.8%	0.266	0.4001	+10.8%	0.481	0.0926
Dataset 2: Apache Hetero.						
	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value
Precision	+4.2%	0.25	0.4250	-12.7%	0.481	0.0929
Recall	+6.6%	0.281	0.3685	+11.2%	0.593	0.0377
***F1*-Score**	+7.8%	0.281	0.3823	+2.7%	0.198	0.5071
Dataset 3: Other Platforms.						
	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value
Precision	+10.9%	0.406	0.1172	-4.1%	0.243	0.4268
Recall	+1.9%	0.235	0.1800	+9.4%	0.716	0.0117
***F1*-Score**	+5.4%	0.344	0.2659	+7%	0.419	0.1432

a) "Avg. Ch. (↑↓)" represents the change between using Imbalance Data for training (IBT) to using Balance Data for training (BT) on our approach (column 2) and the baseline methods (column 5), respectively.
b) "*Cliff's* Δ" effect size quantifies the standardized difference in performance between models trained on IBT and BT as observed in our approach (column 3) and baseline methods (column 6). This metric assesses the magnitude of difference, disregarding the direction of the effect. In the context of RQ2, A value of $|Cliff's\ \Delta| > 0.474$ signifies a large effect size, as represented in red font, indicating a substantial difference.
c) "P-Value" indicates the statistical significance of the change between using IBT and BT for training on our approach (column 4) and baseline methods (column 7). A P-Value < 0.05 indicates a statistically significant difference. In the context of RQ2, a P-Value $>= 0.05$ indicates robustness to data imbalance, and is in black font. The red font highlights the lack of robustness to data imbalance.

Additionally, Table 5 shows the *Average Change*, *Cliff's Delta* effect size, and *Mann-Whitney U Test P-Value* when comparing the accuracy with using *IBT* vs. using *BT* as the training data. The values for the *HLP* approach are in column 3 and column 5; the values for the baseline approach are in column 4 and column 7. Based on the rationale introduced in Section 4, we can draw the following observations:

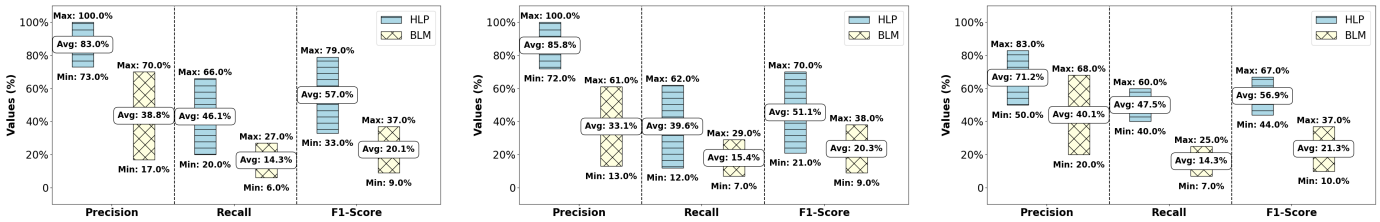
Our approach demonstrates robustness to data imbalance. More specifically, all the absolute *Cliff's Delta* values are $<= 0.474$, indicating no substantially large difference between using *IBT* and using *BT* as the training data on our *HLP* approach. Also, all the *P-Values* are > 0.05, indicating no statistically significant difference between using *IBT* and *BT* for training *HLP* approach.

In comparison, the baseline method lacks robustness to data imbalance, with the recall being significantly compromised. This aligns with the visual observations made with Figure 5. More specifically, the *Cliff's Delta* values associated with the recall are all above 0.474, indicating a large difference of baseline methods between the *IBT* and *BT* experiments. Specifically, the recall has an average increase between 9.4% to 16% when trained using *BT* than *IBT*.

RQ2 Takeaway Message: Our approach is robust in dealing with data imbalance. In comparison, the baseline approach is vulnerable to data imbalance — as the recall witness an average decrease between 9.4% to 16% without data balancing.

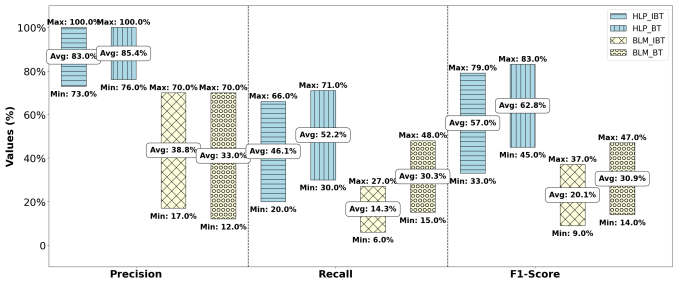
5.3 RQ3: Transferability

Figure 6 depicts the distribution of the accuracy (i.e. precision, recall, and *F1*-score) of our approach when transferring from

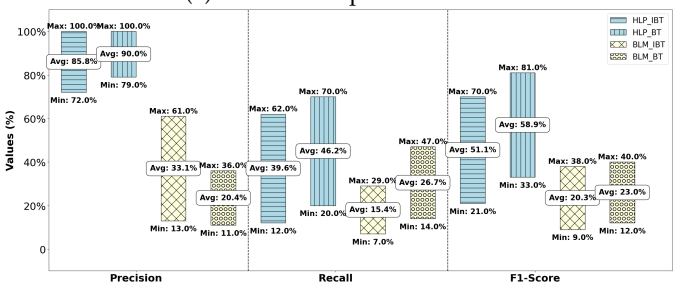


(a) Dataset 1: Apache Homo. (b) Dataset 2: Apache Hetero. (c) Dataset 3: Other Platforms.

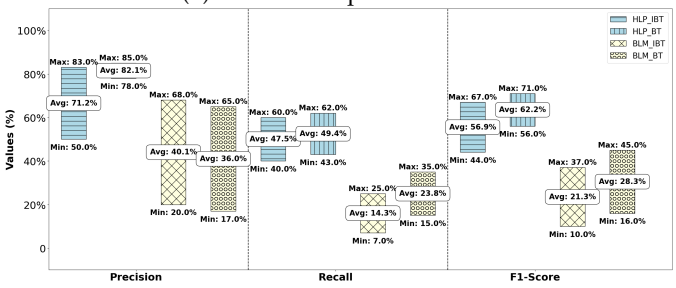
Fig. 4: Overall Accuracy Distribution of Our Approach and Machine/Deep Learning Model-based Baseline Methods (RQ1)



(a) Dataset 1: Apache Homo.

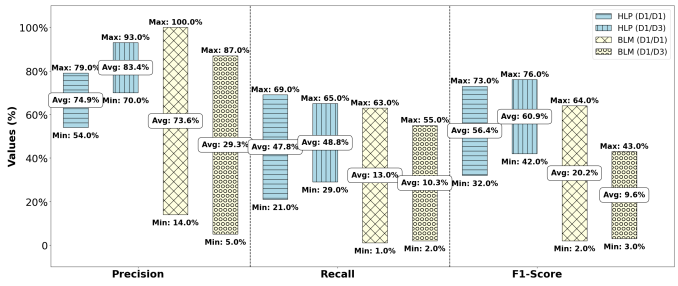


(b) Dataset 2: Apache Hetero.

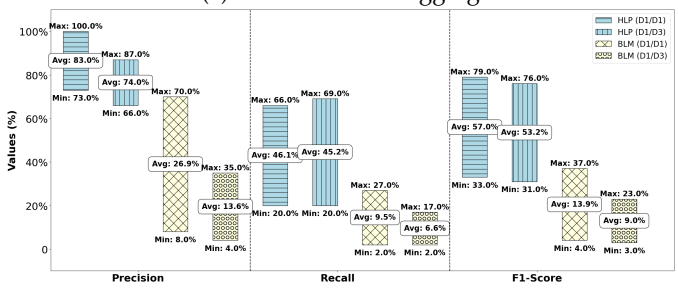


(c) Dataset 3: Other Platforms.

Fig. 5: Accuracy Distribution of Our Approach and Baseline Methods under Imbalanced vs. Balanced Training Data (RQ2)



(a) Sentence-level Tagging



(b) Issue-level Tagging

Fig. 6: Accuracy Distribution of Our Approach and Baseline Methods with vs. without Dataset Transfer (RQ3)

TABLE 6: Impact of Dataset Transfer (RQ3)

Metric	*HLP*: D1/D1→D1/D3			BLM: D1/D1→D1/D3		
	Sentence tagging level					
	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value
Precision	+8.5%	0.563	0.0649	-44.2%	0.763	0.0001
Recall	+1%	0.047	0.9163	-2.8%	0.253	0.0984
***F1*-Score**	+4.5%	0.188	0.5632	-10.6%	0.556	0.0003
Metric	Issue tagging level					
	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value	Avg. Ch. (↑↓)	|*Cliff's* Δ|	P-Value
Precision	-9%	0.431	0.0824	-13.2%	0.618	0.0018
Recall	-0.9%	0.047	0.9162	-2.9%	0.298	0.0508
***F1*-Score**	-3.8%	0.172	0.5990	-4.9%	0.481	0.0059

Note*: a) "Avg. Change (↑↓)" represents the change between the two experiments, i.e. D1/D1 and D1/D3, which indicates the impact of transferring from the Apache Homo. Dataset (D1) to Other Platform Dataset (D3), on our approach (column 2) and the baseline methods (column 5).
b) "*Cliff's* Δ" effect size measures the standardized difference between the experiments with D1/D1 and D1/D3 for our approach (column 3) and for the baseline methods (column 6). A value of $|Cliff's\ \Delta| > 0.474$ signifies a large effect size.
c) "P-Value" indicates the statistical significance of the change between the experiments with D1/D1 and D1/D3 for our approach (column 4) and for the baseline methods (column 7). A P-Value < 0.05 indicates a statistically significant difference.

D1/D1 to *D1/D3* (i.e. the two blue bars in each lane), as well as the distribution of the same information for the baseline approach (i.e. the two yellow bars in each lane). The two sub-figures presents the information for the sentence-level tagging (Figure 6a) and the issue-level tagging (Figure 6b) respectively. We can make the following visual and intuitive observations: The *HLP* approach seems to be quite stable with the transfer at both the sentence- and issue-tagging levels — with the recall and *F1*-score being particularly stable. In comparison, the baseline approach is negatively impacted by the transfer at both tagging levels.

Table 6 provides statistical analysis of whether and to what extent the transfer impacts the *HLP* approach (column 2 to column 4) and the baseline approach (column 5 to column 7) at both the sentence- (the upper half of the table) and issue-tagging (the lower half of the table) levels. We can draw conclusions that assert the observations from Figure 6. More specifically, all the *P-Values* of the *HLP* approach are > 0.05, asserting no statistical significant difference between the accuracy of *D1/D1* and *D1/D3* experiments with the *HLP* approach. Contrarily, the *Cliff's Delta* values and the *P-Values* of the baseline methods indicate that its precision and

F1-score are substantially impacted with data transferring, as the *Cliff's Delta* values are > 0.474 and the *P-Values* < 0.05. Additionally, the "Avg. Ch." (column 5) indicates that impact of the transfer on the baseline approach is negative. For example, the precision witness a decrease of 44.2% at the sentence and a decrease of 13.2% at the issue-level.

RQ3 Takeaway Message: Our approach is project and platform-agnostic, as it is robust without pre-training on new testing datasets. Contrarily, baseline methods suffer a substantial compromise in precision (up to 44.2%) when applied to new datasets without pre-training.

5.4 RQ4: Impact of Fuzzy *HLP* Matching

TABLE 7: Impact of Fuzzy *HLP* Matching for Each Variation in Our HLP-based Approach (RQ4)

Model	Precision			Recall			F1-Score				
	ch (↑↓)	|*Cliff's* Δ|	*p*	*ch* (↑↓)	|*Cliff's* Δ|	Impact	Strict	Fuzzy	*ch* (↑↓)	*p*	Impact
Dataset 1: Apache Homo.											
HLP+NB	-13%	1.0	1.56e-39	+13%	1.0	C1	33%	46%	+13%	3.57e-38	C2
HLP+LR	+3%	0.246	1.33e-03	-9%	0.53		61%	55%	-6%	9.11e-10	
HLP+SVM	-26%	0.524	1.49e-32	-1%	0.059		41%	37%	-4%	2.15e-07	
HLP+DT	0%	0.00	8.61e-02	+10%	0.483	C1	46%	54%	+8%	4.70e-15	C2
HLP+RF	-16%	0.556	4.37e-32	+3%	0.273		62%	58%	-4%	1.10e-05	
HLP+XGB	-26%	0.485	9.00e-32	-2%	0.19		76%	63%	-13%	1.37e-25	
HLP+CNN	+5%	0.385	6.91e-07	+9%	0.527	C1	68%	75%	+7%	7.98e-14	C2
HLP+RNN	+7%	0.482	8.82e-11	+11%	0.522	C1	66%	75%	+9%	1.12e-17	C2
Dataset 2: Apache Hetero.											
HLP+NB	-28%	0.851	3.26e-34	+9%	0.546	C1	21%	33%	+12%	1.07e-21	C2
HLP+LR	-11%	0.586	7.33e-23	+4%	0.237		38%	41%	+3%	3.20e-03	C2
HLP+SVM	-10%	0.533	1.72e-17	+2%	0.201		41%	43%	+2%	2.45e-02	C2
HLP+DT	-8%	0.485	4.62e-13	+6%	0.462		70%	72%	+2%	4.69e-02	C2
HLP+RF	-16%	0.622	1.93e-28	+7%	0.397		60%	61%	+1%	7.19e-02	C2
HLP+XGB	-11%	0.515	7.89e-23	+7%	0.397		55%	56%	+1%	1.07e-01	C2
HLP+CNN	-11%	0.583	9.12e-23	+29%	0.845	C1	56%	69%	+13%	2.05e-26	C2
HLP+RNN	-11%	0.493	7.7e-25	+12%	0.581	C1	68%	69%	+1%	8.86e-01	C2
Dataset 3: Other Platforms.											
HLP+NB	-22%	0.71	4.87e-33	+13%	0.539	C1	53%	55%	+2%	1.08e-01	C2
HLP+LR	-20%	0.696	5.02e-33	+3%	0.201		63%	58%	-5%	6.33e-07	
HLP+SVM	-20%	0.648	5.02e-33	+3%	0.261		63%	58%	-5%	4.61e-07	
HLP+DT	+2%	0.148	1.87e-01	+12%	0.516	C1	44%	52%	+8%	9.45e-13	C2
HLP+RF	+2%	0.160	1.57e-01	+11%	0.516	C1	47%	55%	+8%	9.45e-13	C2
HLP+XGB	-7%	0.462	3.01e-12	+3%	0.184		59%	58%	-1%	9.44e-02	
HLP+CNN	-3%	0.290	3.51e-11	+5%	0.476	C1	67%	68%	+1%	7.7e-01	C2
HLP+RNN	-7%	0.495	2.61e-11	+13%	0.557	C1	59%	63%	+4%	1.37e-06	C2

Note*: a) "Avg. Change (↑↓)" represents the average accuracy change of our approach between using strict *HLP* matching and fuzzy *HLP* matching.
b) "*Cliff's Delta*" effect size measures the difference between using the strict *HLP* matching vs. using the fuzzy *HLP* matching. A value of $|Cliff's\ \Delta| > 0.474$ signifies a large effect size.
c) "P-Value" indicates the statistical significance of the comparison between using the strict *HLP* matching vs. using the fuzzy *HLP* matching. A P-Value < 0.05 indicates a statistically significant difference.

Figure 7 shows the accuracy distribution of our approach using the fuzzy matching technique (i.e. the blue bars) vs. using its strict matching counterpart (i.e. the yellow bars) in our framework. Our experiment spans all three datasets, as shown in Figure 7a, Figure 7b, and Figure 7c, respectively. We can make an intuitive observation, which aligns with our motivation: *The fuzzy matching technique increases the recall of our approach without sacrificing the overall F1-score compared to the strict matching*. More specifically, with the fuzzy matching, the recall of our *HLP* approach witnesses an average of 4.6%, 9.5%, and 7.9% increase; while the *F1*-score increases slightly by 1.3%, 4.4%, and 1.5%, across the three datasets.

As described in Section 4, we conduct a fine-grained analysis of fuzzy matching's impact on each model, as shown in Table 7. For each model, we carefully match the two conditions across three datasets: *C1: recall has substantial increase; and C2: F1-score no significant compromise*. Consequently, four models stand out, namely the *NB*, *DT*, *CNN*, and *RNN*, which consistently achieve the objective of boosting recall without compromising *F1*-score. For example, the *NB* model witnessed a 13% increase in recall (*C1*), as well as a 13% increase in *F1*-score (*C2*), which are both substantial ($|Cliff's\ \Delta| > 0.474$). In contrast, other models are not as reliable when used with fuzzing matching. For instance, the *F1*-score of the *SVM* has a significant decrease on *Dateset 1: Apache Homo.* (i.e. $c = -4\%$ and $p < 0.05$, thus failed *C2*), but a significant increase on *Dataset 2: Apache Hetero.* (i.e. $c = 2\%$ and $p < 0.05$ and thus passed *C2*). Therefore, we do not recommend *SVM* to adopt fuzzy matching due to the lack of reliability across different datasets. Lastly, by comparing the overall accuracy of all the models (as shown in column 8 and column 9), we particularly recommend *CNN* to use fuzzy matching, as it is also a constant winner among all the models.

RQ4 Takeaway Message: We recommend four models to use the fuzzy *HLP* matching technique, namely *NB*, *DT*, *CNN*, and *RNN*, as they consistently achieve the objective of substantially enhancing recall and ensuring a stable overall *F1*-score. In particular, we recommend the *CNN* model as the top performer.

5.5 RQ5: Impact of Issue Matrix for Sentence Order

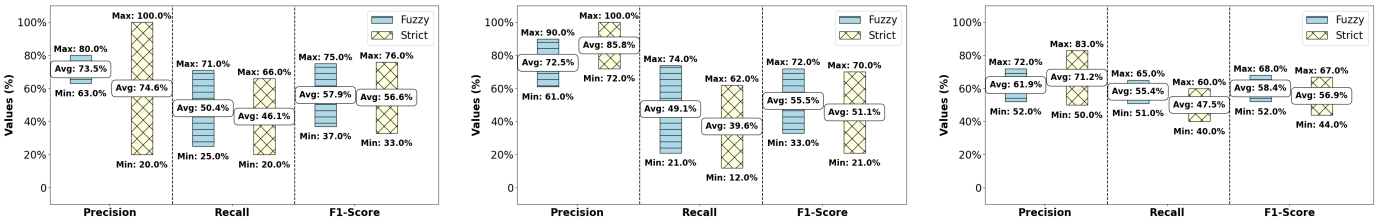
Figure 8 illustrates the accuracy distribution of our approach when utilizing the *Issue HLP Matrix* (i.e., the blue bars) compared to the *Issue HLP Vector* (i.e., the yellow bars). This comparison spans across three datasets, specifically showcased in Figure 8a, Figure 8b, and Figure 8c. An intuitive insight emerges from this analysis: This methodological shift yields notable enhancements in our framework's — an uplift in the average precision by up to 8.4%, recall by up to 3.8%, and *F1*-score by up to 4.8%.

TABLE 8: Impact of Issue Matrix for Each Variation in Our HLP-based Approach (RQ5)

Model	Precision			Recall			F1-Score				
	ch (↑↓)	|*Cliff's* Δ|	Impact	*ch* (↑↓)	|*Cliff's* Δ|	Impact	Vector	Matrix	*ch* (↑↓)	|*Cliff's* Δ|	Impact
Dataset 1: Apache Homo.											
HLP+NB	+1%	0.083	I_{void}	-1%	0.065	I_{void}	34%	33%	-1%	0.053	I_{void}
HLP+LR	-3%	0.272	I_{void}	+8%	0.495	I_+	56%	61%	+5%	0.462	I_{void}
HLP+SVM	0%	0.035	I_{void}	+12%	0.623	I_+	25%	41%	+16%	0.589	I_+
HLP+DT	-5%	0.476	I_-	-6%	0.493	I_-	52%	46%	-6%	0.5	I_-
HLP+RF	+4%	0.321	I_{void}	+8%	0.493	I_+	55%	62%	+7%	0.5	I_+
HLP+XGB	+10%	0.538	I_+	+9%	0.48	I_+	67%	76%	+9%	0.545	I_+
HLP+CNN	+1%	0.083	I_{void}	+1%	0.053	I_{void}	67%	68%	+1%	0.101	I_{void}
HLP+RNN	-1%	0.083	I_{void}	-1%	0.095	I_{void}	67%	66%	-1%	0.083	I_{void}
Dataset 2: Apache Hetero.											
HLP+NB	0%	0.0	I_{void}	1%	0.176	I_{void}	19%	21%	2%	0.239	I_{void}
HLP+LR	3%	0.233	I_{void}	-2%	0.202	I_{void}	40%	38%	-2%	0.228	I_{void}
HLP+SVM	0%	0.00	I_{void}	12%	0.697	I_+	25%	41%	16%	0.868	I_+
HLP+DT	6%	0.285	I_{void}	5%	0.333	I_{void}	65%	70%	5%	0.307	I_{void}
HLP+RF	-1%	0.272	I_{void}	-2%	0.272	I_{void}	62%	60%	-2%	0.287	I_{void}
HLP+XGB	3%	0.342	I_{void}	-2%	0.272	I_{void}	56%	55%	-1%	0.263	I_{void}
HLP+CNN	4%	0.298	I_{void}	-8%	0.609	I_-	61%	56%	-5%	0.307	I_{void}
HLP+RNN	-3%	0.351	I_{void}	4%	0.399	I_{void}	61%	68%	7%	0.443	I_{void}
Dataset 3: Other Platforms.											
HLP+NB	5%	0.338	I_{void}	10%	0.603	I_+	43%	53%	10%	0.659	I_+
HLP+LR	16%	0.896	I_+	10%	0.631	I_+	50%	63%	13%	0.764	I_+
HLP+SVM	26%	1.0	I_+	10%	0.506	I_+	47%	63%	16%	0.783	I_+
HLP+DT	0%	0.00	I_{void}	0%	0.00	I_{void}	44%	44%	0%	0.00	I_{void}
HLP+RF	-10%	0.715	I_-	-20%	0.780	I_-	63%	47%	-16%	0.868	I_-
HLP+XGB	11%	0.781	I_+	-10%	0.666	I_-	60%	59%	-1%	0.122	I_{void}
HLP+CNN	15%	0.724	I_+	0%	0.00	I_{void}	60%	67%	7%	0.467	I_{void}
HLP+RNN	4%	0.273	I_{void}	10%	0.484	I_+	50%	59%	9%	0.729	I_+

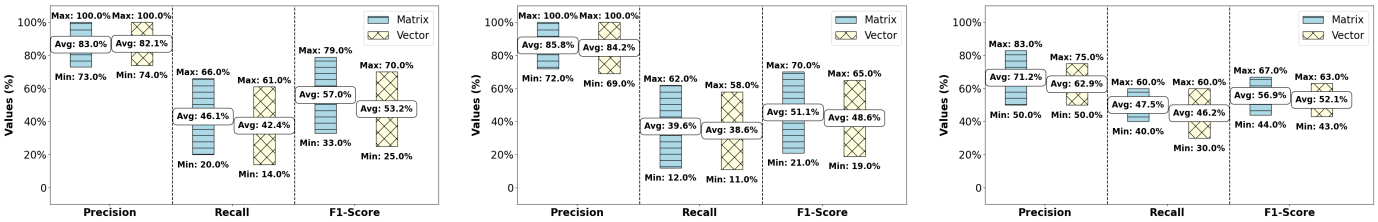
Note*: a) "Avg. Change (↑↓)" represents the average accuracy change of our approach using issue vector (without considering the sentence order in an issue) vs. using issue matrix (which captures the sentence order in an issue) as the learning feature.
b) "*Cliff's Delta*" effect size measures the difference between using the issue vector and issue matrix. A value of $|Cliff's\ \Delta| > 0.474$ signifies a large effect size.
c) "P-Value" indicates the statistical significance of the comparison between using the the issue vector and issue matrix. A P-Value < 0.05 indicates a statistically significant difference.ison between using the strict *HLP* matching vs. using the fuzzy *HLP* matching. A P-Value < 0.05 indicates a statistically significant difference.

Table 8 shows the details of the three levels of impact (i.e. I_{void}, I_+, and I_- defined in Section 4) on each model, across the three datasets. We can observe that, **the impact of the *HLP* Matrix feature is sensitive to different models on different datasets, but the impact is generally positive.** Specifically, on Dataset 2, it does not have a substantial impact except for *SVM*, which has a 16% increase in *F1*-score; on Dataset 1, *SVM*, *RF*, and *XGB* have substantial improvement with up to 16% increase in *F1*-score; and on Dataset 3, *NB*, *LR*, *SVM*, and *RNN* have substantial improvement in *F1*-score by up to 16%. In rare cases, the *HLP* Matrix may have a negative impact, such as the *DT* on Dataset 1 and *RF* on Dataset 3.



(a) Dataset 1: Apache Homo. (b) Dataset 2: Apache Hetero. (c) Dataset 3: Other Platforms.

Fig. 7: Accuracy Distribution of Our Approach with Strict vs. Fuzzy *HLP* Matching (RQ4)



(a) Dataset 1: Apache Homo. (b) Dataset 2: Apache Hetero. (c) Dataset 3: Other Platforms.

Fig. 8: Accuracy Distribution of Our Approach between Using Issue Vector vs. Issue Matrix (RQ5)

Despite the variations with datasets and models, the *HLP* Matrix increases the upper bound of the overall accuracy (*F1*-score) across the three datasets (shown in column 8 and column 9) which justifies its adoption. For instance, the upper bound on Dataset 1 increases from 67% with *HLP* Vector to 76% with *HLP* Matrix (both using *XGB* model); on Dataset 2, the upper bound *F1*-score increases from 65% to 70% both using the *DT* model; on Dataset 3, the upper bound *F1*-score increases from 63% (using the *RF* model) to 67% (using the *CNN* model).

RQ5 Takeaway Message: The *HLP* Matrix generally enhances the performance of our approach. Particularly, it increases the upper bound *F1*-score by 9%, 5%, and 4% on the three datasets, which justifies its adoption. However, due to the variations observed in our experiment, we advise practitioners to adopt a more exploratory approach when using this feature, e.g. testing their datasets and model combinations to fine-tune and optimize the accuracy. We will provide qualitative discussion regarding this in Section 6.

5.6 RQ6: Impact of Feature Selection

First, Table 9 shows that the adoption of feature selection technique, regardless of *Boruta* or *RFE*, does not impact the accuracy of the *HLP* approach, as all the *Cliff's Delta* values are $<= 0.474$ and all the *P-Values* are $>= 0.05$. While maintaining the accuracy during the feature selection, 67 to 76 *HLP*s are retained by *Boruta*; while 59 to 60 *HLP*s are retained by *RFE*. Thus, we conclude that most of the 80 *HLP*s are essential and practitioners cannot significantly trim the number of *HLP*s. Finally, the slight reduction in *HLP*s leads to a minor execution time reduction, between 2.1 to 16.8 seconds.

TABLE 9: Impact of Feature Selection Algorithms (RQ6)

Metric	Impact of *Boruta*					Impact of *RFE*				
Dataset 1: Apache Homo.										
	ch(↑,↓)	|*Cliff's* Δ|	*p*	S-*HLP*s	t_δ (s)	*ch*(↑,↓)	|*Cliff's* Δ|	*p*	S-*HLP*s	t_δ (s)
Precision	+5%	0.04	0.6285			+3%	0.03	0.7117		
Recall	0	0.01	0.9655	67	-7.5	+2%	0.022	0.8594	59	-10.0
***F1*-Score**	+1%	0.05	0.9648			+2%	0.02	0.8037		
Dataset 2: Apache Hetero.										
	ch(↑,↓)	|*Cliff's* Δ|	*p*	S-*HLP*s	t_δ (s)	*ch*(↑,↓)	|*Cliff's* Δ|	*p*	S-*HLP*s	t_δ (s)
Precision	+3%	0.089	0.6854			+3%	0.090	0.7007		
Recall	+2%	0.095	0.6644	68	-2.2	+1%	0.048	0.9520	59	-2.1
***F1*-Score**	+2%	0.1	0.6168			+2%	0.148	0.7906		
Dataset 3: Other Platforms.										
	ch(↑,↓)	|*Cliff's* Δ|	*p*	S-*HLP*s	t_δ (s)	*ch*(↑,↓)	|*Cliff's* Δ|	*p*	S-*HLP*s	t_δ (s)
Precision	+7%	0.109	0.2087			+5%	0.097	0.4110		
Recall	+2%	0.035	0.7906	76	-5.2	+2%	0.025	0.8350	60	-16.8
***F1*-Score**	+4%	0.094	0.3296			+3%	0.038	0.7649		

Note*: a) "Avg. Ch. (↑,↓)" represents the average change of our approach after using Boruta (column 2) or RFE (column 7).
b) "*Cliff's Delta*" effect size measures the difference between using the issue vector and issue matrix. A value of $|Cliff's\ \Delta| > 0.474$ signifies a large effect size.
c) "P-Value" indicates the statistical significance of with and without using a feature selection algorithm. A P-Value < 0.05 indicates a statistically significant difference.
d) "S-*HLP*s" shows the number of selected *HLP*s by the applied feature selection algorithm.
e) "Exec Time Change (%)" represents the average change of the execution time of our approach after using Boruta (column 6) and RFE (column 11) in percentile.

RQ6 Takeaway Message: The majority of the *HLP*s are deemed essential and retained during feature selection. The reduction in execution time is consequently marginal. Therefore, we recommend keeping all the 80 *HLP*s.

6 Discussions and Future Directions

In this section, we first provide qualitative discussion of the RQs to complement the quantitative analysis. Next, we discuss several future directions.

6.1 Qualitative Analysis

6.1.1 *RQ1: Approach Accuracy*

For the qualitative analysis of RQ1, we manually reviewed all the issue reports identified by our approach but missed by the best-performing baseline machine/deep learning model — which includes a total of 52 issue reports across the three datasets. Reviewing the majority (90%) of these issue reports points to a consistent strength of our approach in its capability to capture the highly domain-specific information

related to software performance, compared to the classic deep/machine learning models. Meanwhile, examination of the remaining 10% issue reports does not provide an intuitive understanding of our approach's strength. However, we would like to point out that AI techniques are notorious for not being explainable in some cases [56], [57].

Following we provide two representative examples to show case the advantage of our approach. First, the issue report "IGNITE-5859", which describes an out-of-memory (OOM) error occurring for "*integer values greater than* 2^{30}," aligning with concerns about arithmetic overflow. This phenomenon occurs when a calculation surpasses the maximum storage capacity of a data type — here, an integer in Java — resulting in incorrect outcomes and potential program crashes. This issue could be detected by our *HLP* "STR: memory" due to the mention of the OOM error. However, fully capturing the issue's intricacies requires an understanding of Java's integer limits and the effects of exceeding these limits during arithmetic operations, a depth of understanding often lacking in traditional methods due to their general training. Another example, the issue report "IMPALA-733" describes a situation where the underlying problem (low disk space) **indirectly** affects performance, and it suggests a solution of cleaning up unsuccessful operations for saving time. This case can be identified by our *HLP*, "STR: save_time" and "LEX: disk_spill". Yet, traditional methods, which rely on direct and explicit context, may overlook the nuanced performance implications embedded within the report.

On top of this, we also manually reviewed all the issue reports identified by our approach but missed by the LLMs (i.e. ChatGPTs) – a total of 25 reports. The reason for this separate qualitative analysis of LLMs is that it does not require pre-training like the ML/DL models. The analysis provides consistent insights regarding the advantage of our approach over the LLMs. Our approach serves as a "domain expert" while the LLMs are based on general knowledge. For instance, the report "SOLR-6453" highlights a performance degradation caused by inefficient loop execution. Our *HLP* "STR: loop" captured this report. However, LLMs failed to link "loop" to efficiency and thus missed this report.

Moreover, it is essential to consider the trade-offs between our *HLP*-based approach and LLM-based solutions when applied to a new domain, such as security. LLMs, while more challenging to debug and associated with higher training and inference costs, offer broader applicability with minimal manual intervention required for extracting *HLP*s. This generality makes LLMs more versatile in new application domains beyond the scope of this study — identifying software performance issues. In contrast, the *HLP*-based approach, offering higher accuracy, requires more initial manual effort and domain knowledge to establish the domain-specific *HLP* set. Acknowledging these trade-offs is critical for practitioners when considering the practical application of the *HLP* approach versus LLMs. It highlights a key decision point: the choice between LLMs' more general applicability and less manual intervention vs. the higher accuracy offered by the *HLP*-based approach but higher upfront manual effort to establish the initial *HLP* set.

6.1.2 *RQ2: Impact of Data Imbalance*

For RQ2, we reviewed 51 issue reports identified by our *HLP*-based approach but missed by the baseline methods — specifically from the synthesized imbalance dataset (i.e., IBT) in our experiment. Again, the majority (78%) of these issue reports reinforce our understanding of the *HLP*'s strength under an imbalanced data environment in capturing highly domain-specific information in software performance. However, the other 22% issues cannot be reasoned clearly for the strength of our approach.

We provide two representative examples from the IBT experiment. For example, consider the issue report "AMQ-299" describes a "*memory leak*" in the *JMSStatsImpl* class and a subsequent "*out of memory error*". Traditional methods merely identify the recurring usage of the term "*memory*" as performance related, thus it fails to recognize its critical role in the context of performance discussion. Our *HLP*s, in comparison, adeptly detect vital information related to "*memory*", such as "*memory leak*" or "*memory consumption*" which are highly performance-related. In addition, our *HLP*s have a distinct capability to recognize technical specifics crucial in performance discussions, a task at which traditional NLP features often fall short. For instance, in the issue report "CASSANDRA-7220", the discussion revolves around specific performance issues that relate to time synchronization and CPU load increase. Traditional methods miss the nuanced significance of these technical terms, as they may not appear highly frequently. In contrast, our profiling and structural *HLP*s accurately identify such technical details. For instance, the *HLP*, "time_units", captures the ms (millisecond) as in the synchronization time, while "CPU" *HLP* picks up terms that may indicate CPU load increment, ensuring a thorough and accurate assessment of the report.

6.1.3 *RQ3: Transferability*

Likewise, we reviewed 36 issue reports that are exclusively identified by our approach but missed by the baseline methods in the scenario of skipping pre-training, i.e., using Apache's JIRA issues (Dataset 1) as the training dataset and using Redmine and MantisBT (Dataset 3) as the testing dataset. We particularly focus on delineating the differences observed in the issues sourced from the Apache's data versus those from Redmine and MantisBT data.

For example, "OAK-8067" from Apache, characterized by a large block of extensive code snippets for implementing the performance optimization solution by using lazy disk I/O loading. In contrast, MantisBT exemplary issue reports, "#30127" and "#27960", focus more on broader impacts on users and workflows, often in a less technical language and lacking the intricate technical details. Our *HLP*s adeptly capture these diverse environments by effectively parsing the technical intricacies of Apache JIRA reports and simultaneously grasping the broader, less technical implications in MantisBT reports. This adaptability contrasts with traditional NLP methods that focus mainly on term frequency, which can lead to misinterpretation and inaccuracies in diverse datasets. For instance, traditional methods might overemphasize common programming keywords like "**Class**", "**Exception**", or "**ERROR**" in technical reports, but misapply this emphasis in less technical datasets without the necessary pre-training adjustments.

All the issue reports further reinforce the observations from the above examples. We acknowledge that not every case presents a clear-cut scenario, highlighting the necessity for methodologies that enhance transparency in the machine and deep learning [56], [57].

6.1.4 *RQ4: Impact of Fuzzy HLP Matching*

For RQ4, we reviewed issue reports that were identified with the fuzzy *HLP* matching, yet overlooked by strict *HLP* matching. Given that the accuracy improvement from integrating fuzzy *HLP* matching is modest, we examine only 11 issue reports. The examination confirms our initial motivation to capture more variations of *HLP*s through fuzzy matching. For instance, the issue reports "IMPALA-7474" from Dataset 1, "AMQ-3031" and "HBASE-14489" from Dataset 2 highlighted performance degradation due to heavy CPU usage. As an exemplary case, "AMQ-3031" contains the phrase "*The CPU load increased to 170% over a long period*", which exemplifies the successful implementation of fuzzy matching. Our designated semantic *HLP*, named "CPU", encompasses terms like "*cpu*" and descriptors such as "*a lot of*", "*higher*", "*lower*", "*limit*", "*limitation*", "*usage*", "*utilization*", "*waiting*", "*low level*". These are formulated to encompass a broad spectrum of expressions related to CPU performance issues. Although the exact keywords were not presented in "AMQ-3031", the context clearly suggests a scenario of increased CPU usage, identifiable through fuzzy matching. Hence, we suggest that practitioners consider adopting fuzzy matching to achieve a higher level of recall.

6.1.5 *RQ5: Impact of Issue HLP Matrix for Sentence Order*

Similar to the analysis of RQ4, we examined 10 reports that were uniquely identified by using the *Issue HLP Matrix* as input feature. This confirms our motivation to apply the matrix feature to capture contextual information in longer issue reports. For example, "HBASE-6747" presents a compelling case "*Enable server-side limit default on the size of results returned (to prevent OOMs).....*", followed by an in-depth discussion of the problem and a proposed solution. Only 3 out of the total 14 sentences in this report align with our defined *HLP*s. But notably, these three sentences are strategically positioned at crucial points in the narrative: the first, second, and last sentences. This distribution is not random but rather indicative of the report's primary focus on addressing Out-of-Memory (OOM) issues. Traditional Issue *HLP* Vector methods, which rely primarily on *HLP* weighting, might overlook the significance of locations of the matched *HLP*s due to their sparse and scattered presence. In contrast, the *HLP* Matrix, which incorporates sentence order into the analysis, effectively identifies this report as pertinent to performance concerns, demonstrating the value of considering sentence sequencing in issue report classification. This example is representative of the majority of the reviewed cases (80%), while the minority (20%) cannot be reasoned as intuitively.

To enhance the application of the matrix feature, we recommend practitioners focus on several key characteristics within issue reports. These include complex issue narratives, where problems and their solutions are articulated in a multi-layered, narrative form. Additionally, reports where key information is strategically positioned at specific points rather than clustered, are ideal candidates for adopting the matrix feature. Reports that contain subtle performance-related information, which may not be overtly stated but are implied in the overall context, also benefit from this feature. Lastly, lengthy reports with detailed descriptions, where the sequence of sentences plays a crucial role in understanding the issue, are particularly suited for the matrix feature approach. This methodological focus ensures a comprehensive grasp of the report's content and context, enhancing the accuracy of classification.

6.1.6 *RQ6: Impact of Feature Selection*

For RQ6, we reviewed 7 issue reports impacted by the feature selection techniques. We documented the reduced *HLP*s in the experiment and then analyzed the presence of these *HLP*s within the reviewed issue reports. Our analysis reveals that the feature selection techniques tend to diminish *HLP*s that seldom align with issue reports. For example, *HLP*s such as like 'roundtrip', 'per_NN_per_NN', 'cardinality_improvement', and 'short_lived_sessions', etc., were consistently omitted across all three datasets by both *Boruta* and *RFE*. This is primarily due to their sparse occurrence in the analyzed reports, typically showing only 1 or 2 matches. Conversely, *HLP*s exhibit more frequent matches were preserved.

As a result, the overall count of *HLP* matches did not reduce significantly, leading to only modest improvements in the execution performance of our approach. This insight underscores the importance of maintaining a comprehensive set of *HLP*s, as each one, regardless of its match frequency, plays a critical role in enhancing the precision and reliability of issue report classification.

6.2 Future Directions

6.2.1 *Domain-Specific HLPs*

We carried out an evaluation of our approach's accuracy across diverse software subject domains, following a structured experimental procedure: For the Apache JIRA issue-tracking system utilized in Dataset 1 and Dataset 2, we based the domain categorization on the specific project categories listed on the Apache website (https://projects.apache.org/projects.html?category). Each project evaluated in our study is associated with a "Category" classification on this website, allowing us to inherit the domain classification for our software projects in these datasets. For Dataset 3, the domain categorization was sourced from the "Category" tags in the MantisBT and Redmine databases. Adopting these original category classifications as references ensures our analysis aligns with the pre-established domain classifications of the chosen software projects.

Consequently, we classified all issues across the three datasets into six broad domains: 1) Database-related, 2) Web Frameworks, 3) Libraries and Tools, 4) Query and Analytic Engines, 5) Network Server/Clients, and 6) Others. This classification facilitates a comprehensive understanding of our approach's performance across varied software domains.

The data presented in Table 10 indicates a uniformity in the accuracy of our approach across various subject domains.

TABLE 10: Impact of Subject Domain on Results

Domain	Projects	# Issues	# Perf-Issues	Avg Precision	Avg Recall	Avg F1-Score
Database-related	IGNITE, HIVE, CASSANDRA, TEZ, HBASE	677	50	79%	46%	0.582
Web Framework	SLING, TAP5, FLEX, SOLR	495	25	73%	51%	0.596
Libraries and Tools	SVN, MESOS, PDFBOX, KAFKA, LUCENE, CB, LOG4J2	369	13	68%	55%	0.605
Query and Analytic Engines	IMPALA, SPARK	317	34	72%	51%	0.6
Network Server/Client	AMQ	91	9	68%	52%	0.585
Others	OAK, MantisBT, etc.	221	9	64%	61%	0.624

The average precision, recall, and *F1*-score remain consistent across different domains, highlighting the domain-agnostic quality of our methodology. This finding reinforces the versatility of our approach, confirming its suitability and effectiveness in a wide range of software development environments, irrespective of the specific domain.

Acknowledging the diversity in software project domains, our future work includes tailoring our approach with domain-specific *HLP*s. Users will have the option to select the domain that best fits their project, ensuring that the analysis of issues is more accurately aligned with the specific challenges and characteristics of their domain. This customization will significantly enhance the relevance and effectiveness of the approach for various software development contexts. We will conduct further research to identify and integrate these domain-specific *HLP*s, collaborating with domain experts and analyzing domain-specific datasets to ensure the accuracy and applicability of our approach.

6.2.2 Deployment of the Framework

A key future objective is to make our proposed approach widely accessible by deploying it online and integrating it as a plugin in popular issue tracking systems such as Apache's JIRA, Bugzilla, Redmine, and MantisBT. This plugin will automatically tag issues as performance-related, leveraging the insights and methods developed in our study. The deployment aims to enhance the efficiency of issue tracking systems and aid practitioners in quickly identifying and addressing performance issues. The plugin will feature a user-friendly interface and configurable settings to adapt to different project environments and organizational policies.

Inspired by the findings from RQ4 and RQ5, we plan to introduce user-flexible modes in our approach. More specifically, the users can choose to toggle on and off the two new techniques evaluated in RQ4 and RQ5. If users favor a higher precision, they can turn off the fuzzy matching technique; to the opposite, they can turn it on if they prefer a higher recall. Similarly, depending on the complexity and availability of the contextual information of the issue reports, users can also choose to switch between the *HLP* Matrix vs. *HLP* Vector feature, based on their fine-tuning experiments. As discussed earlier, choosing the most appropriate domain subject should be made available to ensure the best accuracy based on the nature of the project. These options will provide flexibility and enhanced effectiveness in issue tagging, catering to different needs and scenarios in software development.

6.2.3 Practitioner Feedback and Usability Study

Our framework holds significant potential for a diverse range of users, including software developers, researchers, and students. We plan to engage with different user groups to collect their feedback and conduct usability testing of our framework when it is deployed.

We plan to engage with software developers to collect feedback on our framework's integration into their workflows, especially in identifying performance issues. This will involve structured interviews and surveys, focusing on their experience to improve the framework's practicality and usability. Additionally, developers will participate in usability tests where we'll measure the effectiveness of interaction with our framework, assessing task completion time and error rates in identifying performance issues. These insights will be crucial for refining our framework to suit real-world software development needs.

We aim to offer an open database of software performance issue reports, based upon our framework. Researchers who are interested in software performance research can draw and submit performance issue report data to construct common benchmark datasets. Furthermore, researchers whose interests are in issue report classification and triage can build upon our framework using *HLP*s that focus on other areas, such as security, to solve similar challenges.

In educational settings, we plan to integrate our framework in coursework like "*SSW-533 Software Measurement and Estimation*" offered in the Software Engineering Program at Stevens Institute of Technology. Students can employ our framework to identify performance-related issues from real-world projects. This application in coursework will give students practical experience in bug classification, testing, and resolution, as well as an understanding of performance's impact on software quality. This integration of our framework into the curriculum serves as an effective bridge between theoretical knowledge and real-world application, enhancing students' software engineering skills.

7 Limitations and Threats to Validity

7.1 Limitations

First, our framework only focused on issue descriptions as the source of information, without considering other sources such as source code changes, system commit logs, and issue discussions. While issue descriptions are often used as the primary source of information for issue tracking, using additional data sources could potentially improve the accuracy of our approach. Future work could investigate the effectiveness of integrating multiple data sources to improve the performance of our approach.

Second, the heuristic linguistic patterns (*HLP*s) capture how practitioners describe performance problems in general terms. If issues are described only in project-specific terms and understood only by project experts, our approach will be compromised.

In addition, while we recognize the manual extraction of *HLP*s can be labor-intensive and susceptible to human error, current advances in language modeling are yet to overcome the difficulties in accurately generating performance-information-related linguistic patterns automatically. Future work is aimed at harnessing more advanced large language models that can facilitate the automatic extraction and refinement of *HLP*s, reducing the need for intensive manual intervention.

Lastly, we have not separately evaluated the accuracy of our approach on datasets of different software domains. We believe that *HLP*s may result in varying accuracy for different software domains. A possible solution to the last two limitations is to tune the heuristic linguistic patterns with project/domain-specific concepts.

7.2 Threats to Validity

A potential threat to the validity of our study is that the accuracy of our approach may depend on the quality of the training data. While we made efforts to ensure the quality of the ground truth, there may still be errors or biases in the data tagging process that could affect the accuracy of our approach. We acknowledge that the manual tagging of performance-related sentences and issues could pose an internal threat to validity. Any manual effort is subjective to bias derived from individual expertise and understanding. The taggers are not intimately familiar with the reviewed projects, but this should not compromise their ability to recognize general performance problems. It is our intention not to rely on project experts, as our goal is to derive general and transferable linguistic patterns. In addition, we tried to mitigate the risk posed by the manual tagging of the ground truth dataset by allowing multiple taggers to cross-validate results, and by asking for tagging comments to increase transparency.

Another potential threat to the validity of our approach is that the 80 *HLP*s we used may not cover all possible patterns in real-life performance issues accurately. As a result, some issues may not be detected or may be misclassified, leading to lowering accuracy. However, we attempted to address this issue by deriving the *HLP*s based on our analysis of a large number of real-life performance issues through an interactive process.

8 Related Work

8.1 Issue Categorization

Software issue categorization is a crucial process that helps developers and testers prioritize their tasks. To tackle the large volume of issue reports, previous research has proposed various methods for automatic issue categorization. Antoniol *et al.* [58] demonstrated the effectiveness of automatic classifiers such as *Decision Trees*, *Naive Bayes*, and *Logistic Regression* in distinguishing bugs from other issues, while Pandey *et al.* [59] compared more classifiers and found that *Random Forest* is the most effective in text mining of bugs. Some studies aim to categorize issue reports into more diverse categories, such as Ohira *et al.* [60], [61] who classified issues into security, performance, and breakage bugs, and Limsettho [62] who developed an unsupervised framework to group bug reports based on textual similarity. However, previous studies did not consider performance as a specific issue type.

Other studies focused on the internal content of issue reports. Shi *et al.* [18] leveraged linguistic fuzzy rules to classify sentences in feature requests issue reports into different categories, while Aggarwal *et al.* [63] used machine learning models to detect duplicate issue reports. Furthermore, some studies focused on classifying the severity, priority, and complexity of issue reports, such as Lamkanfi *et al.* [64] who proposed a technique to identify bug severity, Tian *et al.* who used machine learning classifiers to predict bug priorities [65], and Zhang *et al.* [66] who proposed a Markov-based method to estimate bug-fixing time. In this study, we focus on performance-related issues and propose a novel approach based on linguistic analysis of issue descriptions to automatically detect such issues. Our approach outperforms baseline methods and demonstrates promising results in handling imbalanced training data, dataset transfer, and feature selection.

8.2 Performance Issue Analysis

A rich body of prior studies has focused on performance issue analysis from different perspectives [4], [6], [7], [8], [46], [67], [68], [69], [70], [71]. Recent studies that rely on real-life performance issues use keyword matching and manual verification to extract a dataset. The most common keywords include *"fast, slow, perform, latency, throughput, optimize, speed, heuristic, waste, efficient, unnecessary, redundant, too many times, lot of time, too much time"* [46], [72], [73]. However, keyword matching compromises the accuracy of the retrieved data. Moreover, there could be many different ways to describe performance issues beyond what is captured in known keywords. Thus, due to the lack of rigor in the keyword-matching approach, researchers have to invest a large amount of time in manual verification.

In addition, prior studies usually focused on specific types of performance bugs based on a limited number of issue reports. Nistor *et al.* [74] studied 150 performance bugs caused by inefficient loops. Yu *et al.* [75] studied 106 performance bugs caused by synchronization bottlenecks. Selakovic *et al.* [69] presented an empirical study of 98 performance bugs written in JavaScript language. Our study provides an adequate and diversified dataset of more than 1000 real-life performance issues, which can serve as a ground truth dataset for future studies on performance bugs.

To address the limitations of the keyword-matching approach, researchers have attempted to utilize natural language processing techniques for performance issue analysis. Liu *et al.* [76] used topic modeling to identify performance-related issues in a Hadoop issue tracking system. Zaman *et al.* [4] proposed a statistical text-mining approach that identifies performance-related issues based on specific patterns and keywords. To the best of our knowledge, our study is the first to propose a linguistic-based approach for performance issue analysis that leverages the rich features of issue descriptions in natural language.

9 Conclusion

In conclusion, our study proposed a hybrid, novel framework for automatically detecting performance-related

issues in software projects, based on linguistic analysis of issue descriptions. The key strength of the proposed framework lies in two aspects: 1) the 80 unique *HLP*s that were derived in a manual inspection of 980 developer-tagged performance issues; and 2) the unique three-layer classification architecture that works on the *HLP* tagging, sentence tagging, and issue tagging progressively.

We conducted a thorough and comprehensive evaluation of our framework on a total of 2,170 issue reports, which contain 13,197 sentences, across four popular issue tracking platforms. We have demonstrated that our approach 1) provides high accuracy, which significantly outperforms the baseline methods; 2) is robust to imbalanced datasets particularly compared to the baseline methods; 3) has good transferability to new projects and new issue tracking platforms without requiring the effort of pre-training. We also demonstrated that the new fuzzy *HLP* matching technique offers a favorable trade-off of precision for higher recall across four models: NB, DT, CNN, and RNN. Specifically, the CNN model is recommended for its superior performance in applying fuzzy *HLP* matching. Additionally, the unique *Issue HLP Matrix* generally enhances accuracy by capturing sentence order in issue reports, though its impact varies across models. Practitioners are encouraged to adopt an exploratory approach when utilizing this feature. Lastly, our findings reveal minimal impact of feature selection techniques on the accuracy and execution performance of our framework, suggesting the use of the complete set of 80 *HLP*s for optimal results.

Overall, our approach can benefit practitioners and researchers who are interested in identifying and diagnosing performance issues in software projects. We believe that our approach shows great potential to help software developers and researchers to better understand and manage performance issues in software projects.

10 Data Accessibility

The data supporting the findings of this study are available on our dedicated Google Site, accessible at "https://sites.google.com/view/hlp-data-package". It can also be accessed at Zenodo – "https://zenodo.org/records/10944186". This resource includes both raw and processed datasets used in the study, consisting of three parts:

- **Saturated *Heuristic Linguistic Pattern* Set:** This contains the 80 *HLP*s derived from the 980 developer-tagged issue tickets from the Apache's JIRA system.
- **Manual Tagging Results:** This contains the sentence- and issue-tagging results for the three datasets.
- **Experiment Results:** This includes the detailed data (e.g., for different DL/ML models in the baseline) from RQ1 to RQ6.
- **Qualitative Analysis:** This contains the issue tickets selected for the qualitative analysis to understand why our approach outperforms baselines in different RQs.
- **LLM Experiment Data:** This presents the detailed results from the experiment with ChatGPT-3.5 and ChatGPT-4.0; as well as the Python script we created to run the experiment with ChatGPTs.

For specific details about the contents of each part, please refer to the provided Google Spreadsheet. Kindly cite this paper appropriately when using the data for subsequent research.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation (NSF) under grants CCF-2044888 and DUE-2142531.

We would also like to express our sincere gratitude to the individuals, Zhihao Deng, Zihan Song, Dhanalakshmi Nangunuri, Gengwu Zhao, Chenhao Wei, and Habib ur Rehman, who helped us with manually tagging the performance-related issues and sentences in the datasets used in this study. We extend our thanks to Andre B. Bondi for his invaluable guidance on real-practice performance issue assignment and handling processes, offering insights that significantly enriched our understanding and approach. We would also like to thank the anonymous reviewers for their valuable feedback, which helped us to improve this work.

References

[1] Connie U Smith and Lloyd G Williams. *Performance solutions: a practical guide to creating responsive, scalable software*, volume 1. Addison-Wesley Reading, 2002.

[2] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[3] Guoqing (Harry) Xu and Atanas Rountev. Precise memory leak detection for java software using container profiling. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008* [3], pages 151–160.

[4] Shahed Zaman, Bram Adams, and Ahmed E Hassan. A qualitative study on performance bugs. In *2012 9th IEEE working conference on mining software repositories (MSR)*, pages 199–208. IEEE, 2012.

[5] Guoqing Xu, Dacong Yan, and Atanas Rountev. Static detection of loop-invariant data structures. In *European Conference on Object-Oriented Programming*, pages 738–763. Springer, 2012.

[6] Dong-Gun Lee and Yeong-Seok Seo. Improving bug report triage performance using artificial intelligence based document generation model. *Human-centric Computing and Information Sciences*, 10(1):26–47, 2020.

[7] Dong-Gun Lee and Yeong-Seok Seo. Systematic review of bug report processing techniques to improve software management performance. *J. Inf. Process. Syst.*, 15(4):967–985, 2019.

[8] Adrian Nistor, Tian Jiang, and Lin Tan. Discovering, reporting, and fixing performance bugs. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 237–246. IEEE Press, 2013.

[9] André B Bondi. *Foundations of software and system performance engineering: process, performance modeling, requirements, testing, scalability, and practice*. Pearson Education, Upper Saddle River, New Jersey, USA, 2015.

[10] Yutong Zhao, Lu Xiao, Pouria Babvey, Lei Sun, Sunny Wong, Angel A Martinez, and Xiao Wang. Automatically identifying performance issue reports with heuristic linguistic patterns. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 964–975, 2020.

[11] Alberto Rodrigues da Silva. Linguistic patterns and linguistic styles for requirements specification (i) an application case with the rigorous rsl/business-level language. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs*, pages 1–27, 2017.

[12] Hisao Ishibuchi, Tomoharu Nakashima, and Tadahiko Murata. Three-objective genetics-based machine learning for linguistic rule extraction. *Information Sciences*, 136(1-4):109–133, 2001.

[13] Prerna Chikersal, Soujanya Poria, Erik Cambria, Alexander Gelbukh, and Chng Eng Siong. Modelling public sentiment in twitter: using linguistic patterns to enhance supervised learning. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 49–65. Springer, 2015.

[14] Laurens Müter, Tejaswini Deoskar, Max Mathijssen, Sjaak Brinkkemper, and Fabiano Dalpiaz. Refinement of user stories into backlog items: Linguistic structure and action verbs. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 109–116. Springer, 2019.

[15] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[16] Lotfi A Zadeh. The concept of a linguistic variable and its application to approximate reasoning—ii. *Information sciences*, 8(4):301–357, 1975.

[17] Maria Jose Gacto, Rafael Alcalá, and Francisco Herrera. Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures. *Information Sciences*, 181(20):4340–4360, 2011.

[18] Lin Shi, Celia Chen, Qing Wang, Shoubin Li, and Barry Boehm. Understanding feature requests by leveraging fuzzy method and linguistic analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 440–450. IEEE Press, 2017.

[19] Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.

[20] EH Mamdani and S Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of human-computer studies*, 51(2):135–147, 1999.

[21] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[22] Justin Martineau, Tim Finin, Anupam Joshi, and Shamit Patel. Improving binary classification on text problems using differential word features. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2019–2024, 2009.

[23] C Jashubhai Rameshbhai and Joy Paulose. Opinion mining on newspaper headlines using svm and nlp. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(3):2152–2163, 2019.

[24] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 26(3):1–37, 2008.

[25] Donghwa Kim, Deokseong Seo, Suhyoun Cho, and Pilsung Kang. Multi-co-training for document classification using various document representations: Tf–idf, lda, and doc2vec. *Information Sciences*, 477:15–29, March 2019.

[26] Man Li, Cheng Ling, and Jingyang Gao. An efficient cnn-based classification on g-protein coupled receptors using tf-idf and n-gram. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 924–931. IEEE, 2017.

[27] Judit Acs, László Grad-Gyenge, and Thiago Bruno Rodrigues de Rezende Oliveira. A two-level classifier for discriminating similar languages. In *Proceedings of the Joint Workshop on Language Technology for Closely Related Languages, Varieties and Dialects*, pages 73–77, 2015.

[28] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014.

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[30] Ben Athiwaratkun, Andrew Gordon Wilson, and Anima Anandkumar. Probabilistic fasttext for multi-sense word embeddings. *arXiv preprint arXiv:1806.02901*, 2018.

[31] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[32] Jingnian Chen, Houkuan Huang, Shengfeng Tian, and Youli Qu. Feature selection for text classification with naïve bayes. *Expert Systems with Applications*, 36(3):5432–5435, 2009.

[33] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[34] Fuchun Peng and Dale Schuurmans. Combining naive bayes and n-gram language models for text classification. In *European Conference on Information Retrieval*, pages 335–350. Springer, 2003.

[35] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[36] Aixin Sun, Ee-Peng Lim, and Ying Liu. On strategies for imbalanced text classification using svm: A comparative study. *Decision Support Systems*, 48(1):191–201, 2009.

[37] Fabrice Colas and Pavel Brazdil. Comparison of svm and some older classification algorithms in text classification tasks. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 169–178. Springer, 2006.

[38] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[39] Mahesh Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[40] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4, 2015.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[43] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, 2019.

[44] Julio Villena-Román, Sonia Collada-Pérez, Sara Lana-Serrano, and José Carlos González-Cristóbal. Hybrid approach combining machine learning and a rule-based expert system for text categorization. In *Twenty-Fourth International FLAIRS Conference*, 2011.

[45] Richard Tong. An operational system for detecting and tracking opinions in on-line discussions. In *Working Notes of the SIGIR Workshop on Operational Text Classification*, pages 1–6, 2001.

[46] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and detecting real-world performance bugs. *ACM SIGPLAN Notices*, 47(6):77–88, 2012.

[47] Lingxiao Jiang and Zhendong Su. Context-aware statistical debugging: from bug predictors to faulty control flow paths. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 184–193, 2007.

[48] Shi Han, Yingnong Dang, Song Ge, Dongmei Zhang, and Tao Xie. Performance debugging in the large via mining millions of stack traces. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 145–155. IEEE, 2012.

[49] Fabiano Pecorelli, Dario Di Nucci, Coen De Roover, and Andrea De Lucia. A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. *Journal of Systems and Software*, 169:110693, 2020.

[50] Fabiano Pecorelli, Dario Di Nucci, Coen De Roover, and Andrea De Lucia. On the role of data balancing for machine learning-based code smell detection. In *Proceedings of the 3rd ACM SIGSOFT international workshop on machine learning techniques for software quality evaluation*, pages 19–24, 2019.

[51] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[52] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[53] Melinda R Hess and Jeffrey D Kromrey. Robust confidence intervals for effect sizes: A comparative study of cohen'sd and cliff's delta under non-normality and heterogeneous variances. In *annual meeting of the American Educational Research Association*, volume 1. Citeseer, 2004.

[54] Joel S Burma, Rebecca M Wassmuth, Courtney M Kennedy, Lauren N Miutz, Kailey T Newel, Joseph Carere, and Jonathan D Smirl. Does task complexity impact the neurovascular coupling response similarly between males and females? *Physiological Reports*, 9(17):1–18, 2021.

[55] Ronald Aylmer Fisher. Statistical methods for research workers. In *Breakthroughs in statistics: Methodology and distribution*, pages 66–70. Springer, 1970.

[56] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, June 2020.

[57] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021.

[58] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement? a text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pages 304–318, 2008.

[59] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, 13(4):279–297, 2017.

[60] Yutaro Kashiwa, Hayato Yoshiyuki, Yusuke Kukita, and Masao Ohira. A pilot study of diversity in high impact bugs. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 536–540. IEEE, 2014.

[61] Masao Ohira, Yutaro Kashiwa, Yosuke Yamatani, Hayato Yoshiyuki, Yoshiya Maeda, Nachai Limsettho, Keisuke Fujino, Hideaki Hata, Akinori Ihara, and Kenichi Matsumoto. A dataset of high impact bugs: Manually-classified issue reports. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 518–521. IEEE, 2015.

[62] Nachai Limsettho, Hideaki Hata, Akito Monden, and Kenichi Matsumoto. Automatic unsupervised bug report categorization. In *2014 6th International Workshop on Empirical Software Engineering in Practice*, pages 7–12. IEEE, 2014.

[63] Karan Aggarwal, Finbarr Timbers, Tanner Rutgers, Abram Hindle, Eleni Stroulia, and Russell Greiner. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process*, 29(3):1821–1842, October 2017.

[64] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 1–10. IEEE, 2010.

[65] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, 20(5):1354–1383, 2015.

[66] Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1042–1051. IEEE, 2013.

[67] Sebastian Baltes, Oliver Moseler, Fabian Beck, and Stephan Diehl. Navigate, understand, communicate: How developers locate performance bugs. In *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*, pages 1–10. IEEE, 2015.

[68] Yepang Liu, Chang Xu, and Shing-Chi Cheung. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1013–1024. ACM, 2014.

[69] Marija Selakovic and Michael Pradel. Performance issues and optimizations in javascript: an empirical study. In *Proceedings of the 38th International Conference on Software Engineering*, pages 61–72. ACM, 2016.

[70] Linhai Song and Shan Lu. Statistical debugging for real-world performance problems. In *ACM SIGPLAN Notices*, volume 49, pages 561–578. ACM, 2014.

[71] Yutong Zhao, Lu Xiao, Wang Xiao, Bihuan Chen, and Yang Liu. Localized or architectural: an empirical study of performance issues dichotomy. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 316–317. IEEE, 2019.

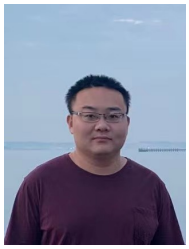
[72] Michael Pradel, Markus Huggler, and Thomas R Gross. Performance regression testing of concurrent classes. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 13–25. ACM, 2014.

[73] Zhifei Chen, Bihuan Chen, Lu Xiao, Xiao Wang, Lin Chen, Yang Liu, and Baowen Xu. Speedoo: prioritizing performance optimization opportunities. In *Proceedings of the 40th International Conference on Software Engineering*, pages 811–821. ACM, 2018.

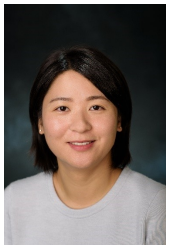
[74] Adrian Nistor, Po-Chun Chang, Cosmin Radoi, and Shan Lu. Caramel: Detecting and fixing performance problems that have non-intrusive fixes. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 902–912. IEEE, 2015.

[75] Tingting Yu, Junmei Ding, Qingxia Zheng, Nanyu Han, Jialin Yu, Yunjuan Yang, Junjun Li, Yuelin Mu, Qian Wu, and Zunxi Huang. Identification and characterization of a new alkaline sgnh hydrolase from a thermophilic bacterium bacillus sp. k91. *Journal of Microbiology and Biotechnology*, 26(4):730–738, 2016.

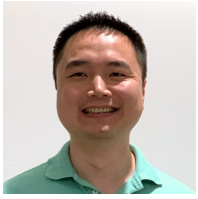
[76] Yang Liu, Yukun Zeng, and Xuefeng Piao. High-responsive scheduling with mapreduce performance prediction on hadoop yarn. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 238–247. IEEE, 2016.



Yutong Zhao is an Assistant Professor in the Department of Computer Science and Cybersecurity at University of Central Missouri. His research focuses on software architecture and performance analysis. He earned his Ph.D. in Software Engineering at Stevens Institute of Technology in 2023, advised by Dr. Lu Xiao.



Lu Xiao is an Assistant Professor in the School of Systems and Enterprises at Stevens Institute of Technology. Her research focuses on software architecture, software evolution and maintenance. In particular, she is interested in modeling and analyzing software architecture and its evolution for addressing quality problems, such as maintenance quality and performance. She earned her PhD in Computer Science at Drexel University in 2016, advised by Dr. Yuanfang Cai.



Sunny Wong is a Senior Software Architect at Envestnet, with over a decade of experience in the aerospace/defense, healthcare, and finance industries. He was named Young Engineer of the Year in 2019 by the IEEE Philadelphia Section. Sunny received his Ph.D. in computer science from Drexel University. His research interests include software architecture and design modeling, tools support for improving developer productivity, and applications of AI techniques to software development.