Noname manuscript No.
(will be inserted by the editor)

The impact of Concept drift and Data leakage on Log Level Prediction Models

Youssef Esseddiq Ouatiti · Mohammed Sayagh · Noureddine Kerzazi · Bram Adams · Ahmed E. Hassan

Received: date / Accepted: date

Abstract Developers insert logging statements to collect information about the execution of their systems. Along with a logging framework (e.g., Log4j), practitioners can decide which log statement to print or suppress by tagging each log line with a log level. Since picking the right log level for a new logging statement is not straightforward, machine learning models for log level prediction (LLP) were proposed by prior studies. While these models show good performances, they are still subject to the context in which they are applied, specifically to the way practitioners decide on log levels in different phases of the development history of their projects (e.g., debugging vs. testing). For example, Openstack developers interchangeably increased/decreased the verbosity of their logs across the history of the project in response to code changes (e.g., before vs after fixing a new bug). Thus, the manifestation of these changing log verbosity choices across time can lead to concept drift and data leakage issues, which we wish to quantify in this paper on LLP models. In this paper, we empirically quantify the impact of data leakage and concept drift on the performance and interpretability of LLP models in three large open-source

Youssef Esseddiq Ouatiti
Queen's university
E-mail: youssefesseddiq.ouatiti@queensu.ca

Mohammed Sayagh
ETS - Québec University
E-mail: mohammed.sayagh@etsmtl.ca

Noureddine Kerzazi
ENSIAS - Morocco
E-mail: nkerzazi@gmail.com

Bram Adams
Queen's university
E-mail: bram.adams@queensu.ca

Ahmed E. Hassan
Queen's university
E-mail: ahmed@cs.queensu.ca

systems. Additionally, we compare the performance and interpretability of several time-aware approaches to tackle time-related issues. We observe that both shallow and deep-learning-based models suffer from both time-related issues. We also observe that training a model on just a window of the historical data (i.e., contextual model) outperforms models that are trained on the whole historical data (i.e., all-knowing model) in the case of our shallow LLP model. Finally, we observe that contextual models exhibit a different (even contradictory) model interpretability, with a (very) weak correlation between the ranking of important features of the pairs of contextual models we compared. Our findings suggest that data leakage and concept drift should be taken into consideration for LLP models. We also invite practitioners to include the size of the historical window as an additional hyperparameter to tune a suitable contextual model instead of leveraging all-knowing models.

Keywords Machine learning · Concept drift · Data leakage · Software logging · Log level prediction

1 Introduction

The practice of inserting logging statements is an important component of the development activity as it provides insights about the execution of software systems [27, 46, 55, 30, 57], so practitioners (e.g., developers, release managers, operators) can prevent and easily fix errors. Each logging statement requires in addition to a message, a log level (e.g., trace, info, warn, error, fatal) that decides the verbosity of that logging statement. Practitioners can then adjust the log levels via a logging framework, to decide which log statements to trace and which ones to ignore. Such a log level adjustment suppresses unnecessary logging statements that might cause noise and overhead to the system. Inversely, the log level can be set to show more verbose logging statements which provides further information during debugging tasks.

Given that choosing the suitable log level for logging statements is not a straightforward task [37, 56] and that developers typically make initial poor log level choices [27, 37], prior studies [28, 29] suggested machine learning models that leverage different metrics to predict the right log level for a logging statement. Such metrics quantify properties about the logging statement (e.g., length of the logging statement), the containing block (e.g., type of block), the containing file (e.g., number of logging statements), the change in the file (e.g., code churn) and the history of the file (e.g., number of revisions). Such models achieve an average AUC performance ranging from 0.75 to 0.81 [28] and from 0.79 to 0.85 [29] across the evaluated projects.

However, neither of these studies [28, 29] takes into account that logging practices can change over time for a variety of reasons not captured by earlier metrics, which can impact the performance and interpretation of these models. In fact, logging strategies are unstable [21] and can change depending on the development phase and the performance of the system. Such changes in the logging practices reflect a global change in the logging strategy from one

period of time to another and are not limited to the update of the log level for certain logging statements. For instance, during debugging activities OpenStack developers typically opt to increase the verbosity of their logs, in order to resolve bugs [1]. Inversely, at another period of the history of the OpenStack project, developers might decide to reduce the verbosity as their logs are getting noisy without any abnormal system activity [2]. Additionally, performance monitoring activities can also motivate log verbosity tweaks as excessive logging from a previous time period can negatively impact the performance of systems [54].

The impact of changing logging practices can lead to two well-known time-issues for LLPs. On the one hand, concept drift refers to the phenomenon according to which the statistical properties of a variable (dependent or independent) unexpectedly change over time [11, 12]. This can occur as the logging practices on which LLP models were trained become obsolete with time (e.g., after the adoption of a new logging strategy). On the other hand, this drift of continuously changing log levels also leads to a higher impact of so-called data leakage on model performance. Data leakage refers to situations where the usage of random train/test splits biases the model by giving it access to future information [22]. This can occur if future logging practices (e.g., logging data when debugging an issue in March) were used to predict past log level decisions (e.g., when debugging an issue in January). While a known issue for other software analytics approaches, the restless nature of log levels can be expected to only exacerbate the impact of data leakage.

While prior studies measured the impact of concept drift and/or data leakage on bug prediction models [8, 4] and AIOps models [34], no prior studies focused on the impact of these time-related issues on LLP models. In fact, given the context-dependent nature of machine learning [2, 6], the impact of time-related issues (i.e., concept drift and data leakage) cannot be directly deducted from the impact on other software engineering machine learning models. For instance, the domain of logging is different than previously studied domains from one side, and the type of data and model are different between log level prediction and the existing studies from another side. These last differences are in terms of the type of data used (stream vs. batch data, in comparison with AIOps data) and the type of predicted outcomes (binary vs. ordinal predicted response, in comparison with defect prediction data). Lastly, while LLP models predict decisions controlled by developers (i.e., the developers choose the log level), prior studied models (e.g., defect prediction) are out of the developer's control and depend mainly on the system characteristics (e.g., bug proneness).

Due to these two differences (i.e., the domain and the type of data), different optimization and fine-tuning routes (e.g., different hyperparameters) might be followed to achieve the best-performing model for each domain, as suggested by Agrawal et al. [2]. These problem-specific optimizations –combined

[1] https://bugs.launchpad.net/nova/+bug/1715785
[2] https://github.com/openstack/swift/pull/15

with the domain and data differences– make software engineering models vary significantly [59]. For example, Bennin et al. [4] reported that –even within the context of defect prediction– some models are more robust to time-related issues than others.

Thus, our work aims to study the impact of the phenomena of data leakage and concept drift on log level prediction models, which are not explored yet, and involve substantially different types of data compared to data on which the two time issues were evaluated in the past (i.e., bug prediction and AIOps). On the one hand, LLPs might not be as impacted as models based on stream data (e.g., AIOps models), as logging strategy changes can be infrequent in time (e.g., projects with long release cycle). On the other hand, one could assume that these log level prediction models might be severely impacted by time-related issues in case of abrupt changes in the logging practice.

Specifically, we evaluate both a state-of-the-art shallow log level predictor (aka. Shallow-LLP) proposed by Li et al. [28] and a state-of-the-art deep log level predictor (aka. DL-LLP) proposed by Li et al. [29]. Note that we evaluate both models as they can have different advantages over each other. In particular, the DL-LLP can be used for its high prediction performances, while the Shallow-LLP as it can be also used for prediction it is more importantly used for the interpretation and explanation of the important factors related to the selection of log levels. Furthermore, we evaluate different time-aware modeling techniques to deal with the data leakage and concept drift observed in log level prediction. To do so, we focus on the following research questions:

RQ1. What is the impact of data leakage on the performance of log level prediction models?
We observe that randomly splitting the training and testing data (i.e., random splitting approach) overestimates the AUC performance of the Shallow-LLP by a median[3] of 7% (Hadoop), 3% (Spring) and 3% (OpenStack). Similarly, the DL-LLP trained using a random-approach overestimates the AUC performance by a median[3] of 9%, 2% and 3.5% for Hadoop, Spring and OpenStack respectively. Such an overestimation can lead up to a maximum of 17% (Shallow-LLP) and 27% (DL-LLP). We also observe that models based on random splitting statistically significantly outperform the models that are based on a time-aware splitting approach in a median[3] of 90% (Hadoop) , 75% (Spring) and 71% (OpenStack) of the testing time frames for the Shallow-LLP, and in a median of 57% (Hadoop), 67% (Spring) and 100% (OpenStack) of the testing time frames for the DL-LLP.

RQ2. What is the impact of concept drift on the performance of log level prediction models?
Shallow-LLP and DL-LLP see statistically significant performance drops occur as early as a median[3] of 1.5, 1 and 1 time frames (two months length) after the end of the training period for Hadoop, Spring and OpenStack respectively. The magnitude of this AUC performance decrease is estimated

[3] Median over the time frame sizes from 4 to 24 months.

at a median[4] of 3.6% (Hadoop), 5.3% (Spring), and 2.8% (OpenStack) for the Shallow-LLP and at a median[4] of 4.5% (Hadoop), 8.2% (Spring) and 3.2% (OpenStack) for the DL-LLP. Across all the testing time frames of a trained model, we observe that the median[4] AUC performance decreases are 8.2% (Hadoop), 5.9% (Spring) and 3.8% (OpenStack) for the Shallow-LLP and 7.7% (Hadoop), 8.7% (Spring) and 7.3% (OpenStack) for the DL-LLP.

Since we observe that both log level predictors (Shallow-LLP and DL-LLP) suffer from the data leakage and concept drift problems, we further investigate the following RQ:

RQ3. What is the best performing time-based modeling strategy for log level prediction models?
Shallow-LLP contextual models trained on a window of historical data statistically significantly outperform (in terms of AUC) all-knowing models (i.e., trained using the entire history) in a median (over different time frame sizes) of 94%, 82% and 86% of the testing time frames (i.e., testing datasets of a given size) for Hadoop, Spring and OpenStack respectively. Meanwhile, we do not observe a statistically significant difference between the performance of contextual DL-LLPs and that of the all-knowing DL-LLP, which is likely due to the trade-off between data quality and quantity that is more pronounced in the case of data-hungry models like the DL-LLP. Furthermore, we find that in the context of the Shallow-LLP no training time frame size is consistently better than the other sizes in all testing time frames. Therefore, the size of the training data for contextual models should be considered as a hyperparameter to tune for LLPs.
While the prior research questions evaluate the impact of time-issues on the performance of LLPs, the following research question investigates the impact of time on the interpretation of LLPs, as time-changing interpretation can be unreliable and confusing. We only investigate Shallow-LLPs, since they are more interpretable compared to the DL-LLPs.
RQ4. How does the interpretability of the log level prediction models change over time?
The interpretability of Shallow-LLPs changes over time. The correlation between the ranking of the most important features of different contextual models is very weak to weak for a median (across different time frame sizes) of 88%, 80% and 79% of the pairs of compared contextual models (e.g., 6 months contextual model trained on time frame T1 and 6 months contextual model trained on time frame T2) for Hadoop, Spring and OpenStack, respectively. Up to 40% (Hadoop), 30% (Spring) and 22% (OpenStack) of the most important features that are shared between different contextual models with the same training size can have a contradictory impact (i.e., an important feature has a positive impact on a given contextual model while the same feature has a negative impact on another contextual model).

[4] Median over the time frame sizes from 4 to 24 months.

We summarize our contribution as follows: (1) Quantifying the impact of data leakage and concept drift on state-of-the-art LLPs. (2) Studying the impact of time on models' interpretation. (3) Evaluating different time-aware modeling strategies designed to eliminate/mitigate time-related issues. Our results suggest that log level prediction models (both Shallow-LLP and DL-LLP) suffer from data leakage and concept drift. While such time-related issues can be mitigated using contextual models for Shallow-LLPs, the identification of the right window size for a contextual model should be considered as a hyperparameter to tweak when experimenting with log level prediction models. Furthermore, our work provides developers with insights on how to mitigate time-related issues when leveraging existing LLPs, as well as a systematic approach to evaluate new LLPs for potential time-related issues. Such insights can improve the tooling efforts for future LLPs, and caution against the blind usage of existing LLPs that might not stand the test of live-environment usage.

Paper structure: The paper is structured as follows. Section 2 presents the background information and discusses the closest work to our study. Section 3 covers the approach used in this study. Section 4 presents our results. Section 5 discuses threats to the validity of our results. Finally, Section 6 concludes our study.

2 Background and Related work

The goal of this section is to present background about log level prediction and discuss the closest work to our paper. In particular, we discuss four main research areas related to our paper: logging practices, the application of machine learning to logging practices, concept drift and data leakage in machine learning models, and time-related issues detection for software analytics models.

2.1 Background on Log level prediction

Developers insert log levels into their logging statements to describe the severity of the recorded events (see example in Figure 1). Along with the logging framework (e.g., Log4j) threshold, these log levels allow printing relevant logging messages and suppressing unnecessary ones. For example, if the threshold is set at *Warning*, all logging statements with a higher verbosity (i.e., *Info*, *Debug* and *Trace*) will not be recorded in the log file. However, picking the right log level for a given logging statement is not a simple task [37, 56], as developers do not know for sure how the code will be used in the future [37]. Thus, it is not unusual that developers change the log levels of their logging statements throughout the history of development [56].

[5] https://github.com/apache/hadoop/commit/002dd6968b89ded6a77858ccb50c9b2df074c226

```
        } catch (Exception e) {
-         return; // message logged in renewDT method
+         LOG.error("Exception renewing token" + token + ". Not rescheduled", e);
+         removeFailedDelegationToken(dttr);
        }
```

Fig. 1: Example of an added logging statement to the Hadoop project [5]

Model	Features class	Features description
Shallow-LLP	Logging statement	- Length of the logging statement. - Number of variables - Frequency of tokens in the logging statements.
	Containing block	- Number of LOC in the containing block. - Type of the containing block. - Exception type (if containing block is catch block).
	Containing file	- Logging statement density in the file. - Number of logging statements in the file. - Average logging statement length in the file. - Average number of variables in the logging statement of the file. - Number of logging statements in the file. - McCabe complexity. - Fan In.
	Change features	- Code churn - Logging statements churn. - Portion of changed logging statements among changed lines of code.
	Historic features	- Number of lines changed in the history of the containing file. - Number of revisions in history of the containing file. - Number of changed logging statements in the history of the containing file. - Portion of changed logging statements among changed lines if code in the history of the containing file. - Number of revisions that change logging statements.
DL-LLP	Syntactic context	- The sequence of AST tree nodes containing the logging statement (e.g., [Method Declaration , If statement, Logging statement, Method invocation]). The beginning of the sequence is marked by the start of the method and the end is marked by the end of the basic block (e.g., if block) that contains the logging statement.
	Logging message	- The sequence of natural language tokens in the logging message. * This sequence of tokens in inserted instead of the "Logging statement" tag in the example above (i.e., [Method declaration, If statement, startLogStmnt, this, is, a, logging, message, endLogStmnt, Method invocation])

Table 1: Features used to train the Shallow [28] and DL [29] LLPs

In order to assist developers with the log level decision, machine learning models were proposed by prior research [28, 3, 29]. In this paper, we study the state-of-the-art shallow log level predictor (aka. Shallow-LLP) suggested by Li et al. [28], in addition to the state-of-the-art (aka. DL-LLP) log level prediction model suggested by Li et al. [29]. Both log level predictors predict an ordinal variable (i.e., ordered categorical variable) ranging from 1 to N, where N is the number of log levels supported by the logging framework (e.g., N=6 for Log4j). Table 1 summarizes the features used by each of our studied models to predict the appropriate log level.

2.2 Prior work on logging practices

Many research efforts were conducted to understand and automate logging practices [9, 27, 40, 57]. For example, Fu et al. [9] conducted a mixed qualitative and quantitative study to recommend the source code locations to which a logging statement should be added. Yuan et al. [57] introduced LogEnhancer, a tool to support the diagnosis of software failures by automatically inferring variables that may affect whether a logging statement is relevant or not. Pecchia et al. [40] observed that –despite having different purposes– different product lines have similar logging styles. Li et al. [27] conducted a qualitative study to analyze the benefits and costs of logging from the developers' perspective and draw a comprehensive picture of the logging practices.

Our study is different from this line of research as we quantify the impact of concept drift and data leakage on log level prediction models, rather than investigating logging practices in general.

2.3 Prior work on the application of machine learning to logging practices

Another line of research focuses on leveraging statistical/machine learning models to facilitate the logging practices for developers [28, 29, 18, 25, 26, 32, 61]. Zhu et al. [61] gathered features from code snippets, such as exception catch blocks, and trained a model to guide developers with their logging decisions. Li et al. [26] leveraged random forest models to suggest log changes based on commit information. Liu et al. [32] trained a model that ranks the source code variables by their likelihood of being logged. Li et al. [26] leveraged topic models using code snippets to infer the events that are likely to be logged.

Our work is different from this line of research as instead of leveraging machine learning models to support logging activities, we aim to quantify the impact of data leakage and concept drift on log level prediction models. Such time-related issues can occur as log level assignment patterns change due to changing logging strategies (e.g., more verbose logs when dealing with bugs [46]).

2.4 Prior work on time-related issues in machine learning

A large amount of research efforts studied time-related issues exhibited by software analytics machine learning models [10, 14, 20, 31, 33, 34, 42]. For instance, Lu et al. [20] surveyed the state-of-the-art knowledge on concept drift including three main concerns: *concept drift detection*, *concept drift understanding*, and *concept drift adaptation*. The authors claim that drift detection research should not only focus on identifying drift occurrence time accurately but also on providing the information of drift severity and regions. Lu et al. [33] studied the impact of noisy datasets on concept drift and proposed a two-step

approach to handle this issue. Liu et al. [31] argued that concept drift might take place in only some sections of the historical data and consequently, older non-drifted data should not be dismissed. Therefore, the authors introduced a novel metric that helps in detecting regional concept drift through investigating the distribution of the nearest neighbors of the drifted data. Ramrez et al. [42] surveyed research work on data processing aiming to react effectively to drift.

Our work complements this line of research by investigating concept drift and data leakage for log level prediction models that are supposed to be differently impacted by (i.e., more impacted or less impacted) these time-related issues, as their domain, data and model optimization strategies (e.g., hyperparameter tuning) are different than models studied in the past (e.g., AIOps models).

2.5 Prior work on time-related issue detection for software analytics models

Other research efforts focused on identifying time-related issues (e.g., concept drift, data leakage) in models used to assist software engineers with their activities (e.g., defect prediction) [8, 34, 35]. For instance, Ekanayake et al. [8] found that defect prediction models suffer from concept drift through their evolution history. Such concept drift can be anticipated when the number of authors modifying a studied project suddenly changes (i.e., increases, decreases), or when the authors themselves change. Lyu et al. [34] studied the impact of data splitting decisions on the performance of AIOps models and suggested that these models leverage time-aware modeling approaches in order to mitigate the effect of data leakage. Such an effect is exhibited through the unrealistic performance that AIOps models trained using a random train/test split achieve compared to the performance that those same models achieve using a time-aware train/test split (i.e., the train data is historically before the testing data). Lyu et al. [35] studied the stability of the interpretability of AIOps models with respect to time and reported that models trained on specific time frames or on the entire history of a project have consistent important feature rankings.

3 Methodology

In this Section, we discuss how we train, test and interpret our models, using a methodology similar to a large number of prior studies that leverage machine learning techniques to assist software engineering practitioners [28, 29, 39, 34, 35, 48, 17, 23, 41, 53].

In this paper, we study the impact of concept drift and data leakage on log level prediction models, so that we can understand how time context changes can affect models used to assist developers with logging activities. Log level prediction models predict the appropriate log level for a newly introduced

Project	Programming language	Span of data (days)	Number of added logging statements	Average per day
Hadoop	Java	3790	16841	4.44
OpenStack	Python	2368	23955	10.11
Spring	Java	3207	6018	1.87

Table 2: Studied Projects historical information

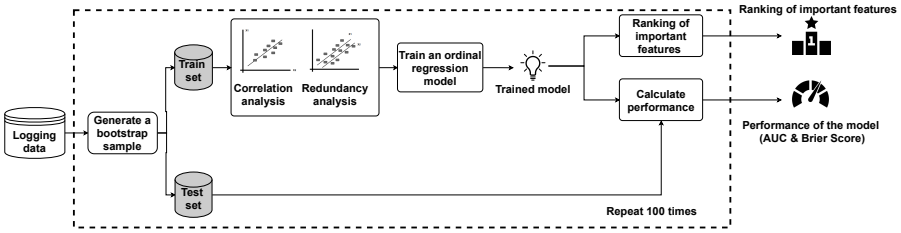


Fig. 2: An overview of the methodology for training our models. Both training and testing datasets are further detailed in the approaches of our research questions.

logging statement, hence they require an ordinal predictor (i.e., categorical and ordered dependent variable) such as the ordinal logistic regression model suggested by Li et al. [28] or the ordinal deep learning model suggested by Li et al. [29]. For the purposes of our study, we leverage three popular, large and well-maintained open-source projects: Hadoop is a distributed computing system, Spring is a modular project that offers a vast pool of functionalities to Java Developers, and OpenStack is a cloud computing platform. Each of the studied projects has been developed and maintained for at least six years. Similar to prior work [28, 29], we collect all the revisions from the Github repositories of each of the studied projects. Next, we use the "git diff" command to collect code changes between revisions. The added logging statements along with their labels (i.e., log levels) are then obtained using a regular expression used by prior works [28, 29]. Finally, we extract the respective features for the Shallow and DL-LPPs following each of the approaches from prior works [28, 29]. We summarize information about the datasets used in our study in Table 2.

To quantify concept drift and data leakage for log level prediction models, we follow the modeling approach discussed in the remainder of this section. Our machine learning pipeline (i.e., data preparation, modeling, testing and feature importance) is similar to approaches followed by prior studies [28, 34, 35, 39]. While our training and testing datasets differ based on each of our experiments (as discussed in the approach of each of our research questions), the following training and testing steps, also shown in Figure 2, are common to all of the case studies of our paper.

Note that we leverage the same features (see Table 1) used by Li et al. [28] (ordinal regression) and Li et al. [29] (deep learning model) to train their respective models.

3.1 Bootstrap Sampling

Similar to prior work [24, 52], we leverage the out-of-sample bootstrap validation technique to train and test our models. This approach consists of generating a sample with replacement (i.e., bootstrap sample) on which the model is trained. The model performance is then tested using the observations that were not selected in the bootstrap sample (i.e., out-of-bootstrap sample). This process of sampling, training and testing is repeated 100 times to guarantee robust findings. Note that for the case of time-aware models (i.e., models taking into consideration the chronological order of data), our training bootstrap samples are always coming historically before the testing datasets, as explained in the approach section of RQ1. Note that this sampling strategy is used for both of our evaluated models.

Note that the two following steps (3.2 and 3.3) only apply to the Shallow-LLP trained using ordinal regression.

3.2 Correlation Analysis

For each training set, we conduct a correlation analysis to avoid erroneous model interpretation [19, 49]. Indeed, in order to guarantee a consistent ranking when interpreting models, it is recommended to discard one out of each pair of correlated features [19]. Similar to prior work [24, 43], our correlation analysis leverages Spearman correlation with a threshold of 0.7. We opt for Spearman correlation due to its resilience to non-normally distributed data. Finally, we list our features by order of priority to guarantee a consistent correlation analysis across the different bootstrap iterations. The highly prioritized features are the ones related to exceptions (e.g., type of exception), the most important features reported by Li et al. [28] and the features related to logging activity (e.g., we keep log churn rather than code churn) as these features are characteristic to the logging activity. For each pair of highly correlated features (i.e., $\rho > 0.7$), we keep the feature with the highest priority.

3.3 Redundancy Analysis

As correlation analysis is not able to completely eliminate collinearity, we conduct a redundancy analysis similar to prior studies [24, 41, 50]. It consists of reducing the collinearity by iteratively identifying which independent feature is explainable by the other independent features. To do so, different preliminary models are built for each bootstrap sample, each of these models explains one independent feature with the other ones. We exclude an independent feature if its associated preliminary model has an $R^2 > 0.9$. We leverage the implementation of redundancy analysis provided by the *redun* function from the *rms* package [1].

3.4 Training and Testing

3.4.1 Shallow-LLP - Ordinal regression model:

Using the remaining features from the previous steps, we train our ordinal regression models –which are an extension of logistic regression for ordinal dependent variables– that predict the suitable log level (i.e., ordinal output variable) for a given logging statement. These models are tested on a different dataset, according to the specific RQ under analysis. To evaluate our model, we consider the AUC (Area under the ROC curve) and Brier Score as two standard performance metrics similar to prior studies [28, 41, 50]. While the AUC measures the discrimination ability of the model (the higher above 0.5, the better), Brier Score captures how often a model can predict the right class. An AUC of 50% is equivalent to a random guess. The Brier score ranges between 0 and 2 and the lower it is, the better the model is. A random guess model has a Brier Score of 0.8. While Brier Score is designed for multi-class evaluation, we leverage the multi-class AUC score generalization proposed by Hand and Till [13], as it has been vastly used by prior work [28, 59, 44] to evaluate multi-class classifiers (e.g., Human expert effort estimation), and implemented by popular machine learning libraries (e.g., Scikit-learning [6] and pROC [7]).

3.4.2 DL-LLP - Ordinal deep learning model:

Using the features explained in Table 1, we train a deep learning model similar to the approach followed by Li et al. [29]. This DL-LLP is a state-of-the-art log level predictor, that uses the syntactic features (e.g., containing blocks) and log message features to predict an ordinally encoded output (i.e., the log levels). The architecture of the DL-LLP consists of an embedding layer, an RNN and an output layer. While the embedding layer takes the input features (i.e., contextual and log message features) and represents them in the form of a sequential vector (i.e., the sequence of code statements within the block that contains the logging statement), the RNN layer (Bi-LSTM) is used to train the deep learning model. Finally, the output layer receives the output from the RNN layer and generates an ordinal output reflecting the predicted log level. We refer to Li et al. [29] for further details about the implementation. We leverage the AUC score to evaluate our DL-LLPs.

Note that the approaches used to perform our empirical evaluation in RQ1, RQ2 and RQ3 (as explained in each RQ's approach) are the same regardless of the model used. For instance, we quantify concept drift for the Shallow-LLP in the same way we quantify it for the DL-LLP.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc$_a uc_s core.html$

[7] https://www.rdocumentation.org/packages/pROC/versions/1.18.0

3.5 Feature Importance:

Similar to prior work [23, 53], we use Wald's χ^2 to evaluate the impact of each feature on predicting the log level. A large χ^2 for a given feature mirrors the large explanatory power of that feature. From each of our 100 bootstrap iterations, we obtain a ranking of features based on importance. We then apply a Scott-Knott [7] clustering technique on the 100 rankings, similar to previous work [23, 41, 53], in order to aggregate those rankings into a final ranking. The Scott-Knott approach uses a hierarchical clustering to group the means of importance scores into groups that are statistically different. We leverage these generated rankings to compare the interpretability of the different models we train.

Note that for the context of our study we used approaches (i.e., analysis of variance) aiming for the global explainability of our evaluated log level predictor [51], which are widely used in the field of software engineering [48, 17, 23, 41, 53, 39] and are similar to how the models we evaluate were interpreted by Lee et al. [23]. That is, we do not interpret how the model takes individual decisions (aka., model-agnostic techniques).

4 Results

RQ1. What is the impact of data leakage on the performance of log level prediction models?

Motivation: The goal of this research question is to quantify the impact of data leakage on the way log level prediction models were evaluated by prior studies [28, 29]. In fact, ML models suffering from data leakage typically offer unrealistic performance that might not be replicable in a life-environment setup. Having insights about data leakage for LLPs would help practitioners better evaluate their LLPs and safeguard against unrealistic performance expectations.

Approach: To quantify the impact of data leakage on log level prediction models, we compare the AUC and Brier Score performance of models built using two separate approaches that are described as follows:

- *The random-based model*: trained using a random train/test split that is susceptible to data leakage. The random split does not take into consideration the chronological order of data, as shown in Figure 3. For example, a model that uses the first two batches and the last batch of data for training might suffer from data leakage when predicting on the testing batch (3rd batch). This potential data leakage is due to the fact that the model might gain knowledge coming from the future (4th batch) on the testing data (3rd batch). This approach was leveraged by Li et al. [28] to train their log level prediction models.

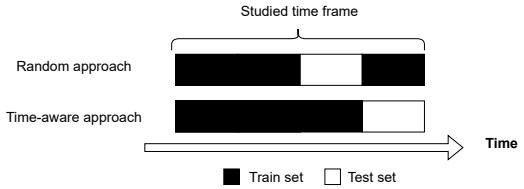


Fig. 3: Overview of the modeling approaches for a single studied time frame.

– *The time-aware approach model*: trained taking into consideration the chronological order of data, as shown in Figure 3. The time-aware approach represents how the model would be used in practice (i.e., without having access to training data from the future).

Note that the difference between the two approaches is related to the train/test splits, while the leveraged features and the learning algorithm (i.e., ordinal regression) remain the same.

To train our two models, we follow the approach shown in Figure 4. In particular, we consider the following steps to compare the random-based models and the time-aware models:

First, we split the whole history of data equally. Each part of the history is hereafter referred to as a "time frame". A time frame has a length of a number of months (e.g., four-months time frame) and contains all the existing logging statements in those months. In our paper, we evaluate different time frame sizes that range between four and 24 months to make our results generalizable to different time frame sizes. The lower bound that we used is a four-month time frame, since this is long enough to train and test our models without risking overfitting.

Then, we build both types of models:

(i) The random-based model is trained on a bootstrap sample from a time frame (e.g., the first 4 months of a project) and tested on the out-of-bootstrap-sample portion of the same time frame. We consider the steps discussed in Section 3.4 to train our model. To guarantee that each time frame has enough data to train our model, we set a threshold of 300 observations for every training time frame, in order to have 10 features per observation [15].
(ii) For the same time frame, we train a time-aware model on a bootstrap sample from the chronologically first 70% of the data and test that model on the following 30% of the same time frame data. Similarly to the random model, we consider time frames with at least 300 observations for training a model.

We repeat the previous two steps 100 times by leveraging different bootstrap samples to end up with a distribution of 100 performance (i.e., AUC and Brier Score) measurements for the random-based models and another distribution of 100 measurements for the time-aware models. We statistically compare

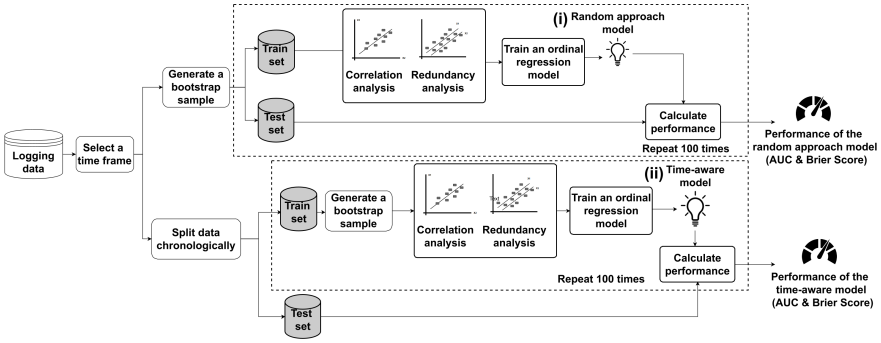


Fig. 4: An overview of the methodology for quantifying data leakage

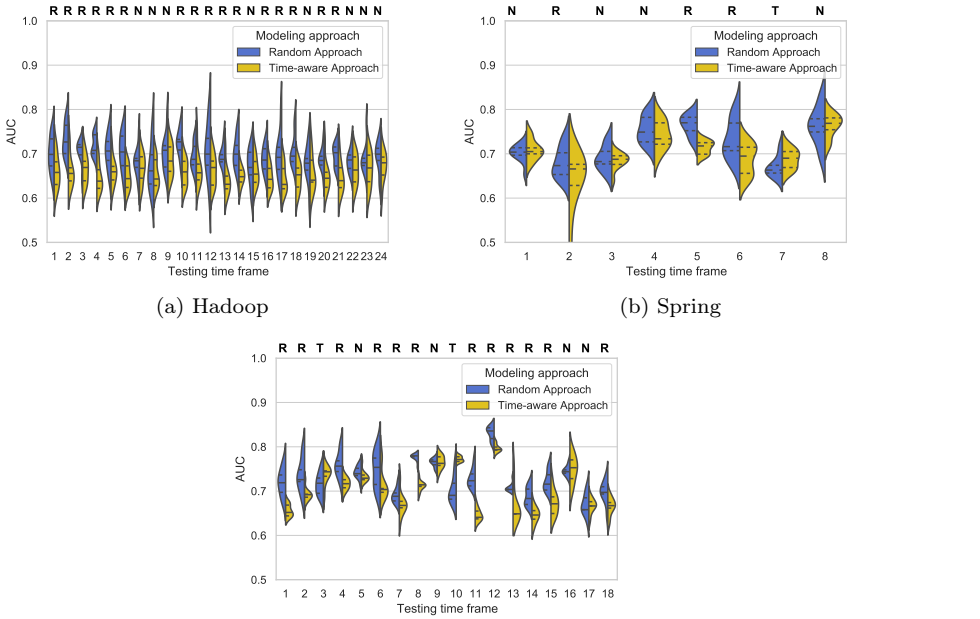
these two distributions to identify whether they are different using a Wilcoxon test ($\alpha = 0.01$). If so, we also measure the amount of differences as well as the magnitude of such a difference using Cohen's d. Note that the median performance values reported in our findings for individual models (e.g., LLP trained on 4 months worth of data) are calculated over the 100 bootstrap samples. Meanwhile, global findings about a given project (e.g., LLPs trained using Hadoop data) are aggregated based on the data frame size (e.g., median over the different time frame sizes used to train a contextual model).

We repeat the same experiments on the following time frames (e.g., the 2nd 4 months of a project) of the same size. Finally, we repeat everything for the other time frame sizes (e.g, a time frame of 5 months), up to 24 months.

We leverage the same approach to quantify data leakage for both of our evaluated log level predictors (i.e., Shallow and Deep learning).

Results: Random-based models overestimate the AUC performance on the test sets by a median[8] that ranges between 3% and 7% (Shallow-LLP) and between 2% and 9% (DL-LLP) compared to time-aware models across our evaluated time frame sizes and case studies. In fact, Hadoop's Shallow-LLP trained using the random approach overestimates the time-aware models by a median[8] of 2% (observed for four-month based models) to 17% (observed for seven-month based models). The overestimation ranges between 1% and 14% for Spring and between 1% and 12% for OpenStack. For example, our evaluated four-month random-based models inflate the AUC performance compared to the four-month time-aware models by a median[8] AUC of 2%, 2%, and 3% for Hadoop, Spring and OpenStack respectively, as shown in Figure 5. We observe similar overestimation for the DL-LLP as shown in Figure 6 for our evaluated four-month based models. Such an overestimation indicates that one has to re-evaluate the LLP models using the time-aware approach as there is a chance that using a simple random-data splitting approach can overestimate the performances of these

[8] Median over the 100 bootstrap samples.



(a) Hadoop

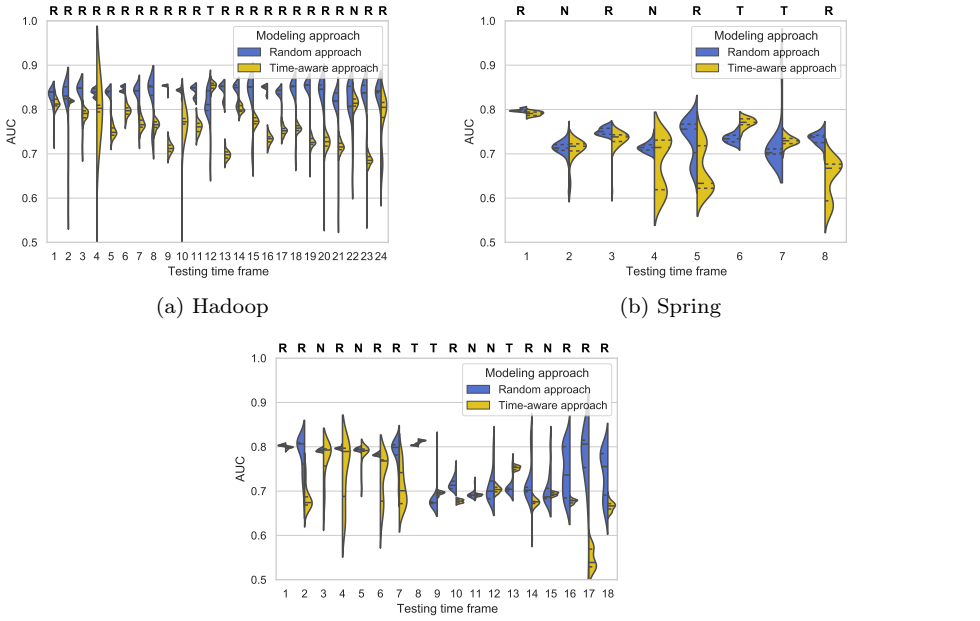
(b) Spring

(c) OpenStack

Fig. 5: AUC performance of the random and time-aware Shallow-LLPs on four-month testing time frames. **R** indicates time frames where the random approach is the best performing, **T** indicates time frames where the time-aware approach model is the best performing and **N** indicates time frames where there is no significant difference between the performance of the two models.

models, hence mislead practitioners that might consider the model as good when it has low performances (as close as a random guess as we observed in the case of Hadoop Shallow-LLP with six month time frames). Note that our observations stand for the Brier Score as well. The figures for the other time frame sizes are available in the online appendix [38].

Across the different time frame sizes, a median of 90%, 75%, and 71% (Shallow-LLP) and a median of 57%, 67%, and 100% (DL-LLP) of our evaluated random-based models are statistically significantly (Wilcoxon test; $\alpha = 0.01$) better-performing than our evaluated time-based models for Hadoop, Spring and OpenStack respectively. For example, we observe that 67%, 37% and 67% of the four-month random-based Shallow-LLPs statistically significantly (Wilcoxon test; $\alpha = 0.01$) outperform the four-month time-aware models for Hadoop, Spring and OpenStack respectively. Similarly, we observe that 91%, 50% and 55% of the four-month random-based DL-LLPs statistically significantly (Wilcoxon test; $\alpha = 0.01$) outperform the four-month time-aware models for Hadoop, Spring and Open-



(a) Hadoop

(b) Spring

(c) OpenStack

Fig. 6: AUC performance of the random and time-aware DL-LLPs on four-month testing time frames. **R** indicates time frames where the random approach is the best performing, **T** indicates time frames where the time-aware approach model is the best performing and **N** indicates time frames where there is no significant difference between the performance of the two models.

Stack respectively. All of these differences have a large effect size (Cohen's d, $d > 0.7$). On the other hand, only a median (over different time frame sizes) of 0%, 0% and 14% of the time-aware Shallow-LLPs statistically significantly outperform the random-based models. We observe similar results for the DL-LLPs, as only a median (over different time frame sizes) of 0%, 0% and 0% of the time-aware DL-LLPs statistically significantly outperform the random-based models.

These findings can be explained by the fact that the distributions of the independent features are different between the training and test sets that are leveraged for our time-aware models. We observe that a median of eight to 13 (depending on the evaluated time frame size) features are statistically significantly different (Wilcoxon test, $\alpha = 0.01$) between the training and test datasets that are used for our Hadoop time-aware models. This median number of statistically different features ranges between 8 and 14 for Spring and five and nine for OpenStack. For example, our four-month based model has a median[9] of nine, 11 and seven features (out of 25) that are statistically signifi-

[9] Median over the 100 bootstrap samples.

cantly different between the training and testing sets of the time-aware model trained on Hadoop, Spring and OpenStack respectively. Note that the OpenStack project, which has the most time frames in which the time-aware model outperforms the corresponding random-based model (median of 14%), is also associated with the lowest number of significantly different features between the training and testing sets used by the time-aware model.

Summary of RQ1

Log level prediction models trained using the random approach can overestimate the expected performance of a time-aware log level predictor by a median AUC up to 17% (Shallow-LLP) and 27% (DL-LLP) higher. **Our findings suggest leveraging time-aware approaches for shallow and DL LLPs for more realistic performance estimations**.

RQ2. What is the impact of concept drift on the performance of log level prediction models?

Motivation: The goal of this research question is to quantify concept drift on log level prediction models. While log choice strategies might vary from one time period to another (e.g., when debugging vs. after fix), it is not clear whether a concept drift caused performance decrease exists for LLPs, and if such a performance decrease exists, how soon after the end of the training dataset the concept drift can be manifested. Such insights would warn practitioners on the importance of updating their models so they can better maintain their LLPs (e.g., plan for updates).

Approach: To quantify the impact of concept drift on log level prediction models, we train a model (Deep and shallow LLPs) on a selected time frame and test that model on each of the following time frames, as described in the following steps and illustrated in Figure 7.

We split the whole existing data into equal time frames. For each time frame (TF), we perform the following:

- We train a model on a bootstrap sample from the first 70% of the observations of TF.
- We test our model on the remaining 30% of the observations to obtain a *baseline performance*.
- We test our model on each two-month time frame that chronologically follows TF and compare the obtained performance to the baseline performance. The two-month testing time frame guarantees enough observations (i.e., at least 50 observations) to have statistically significant findings. Note that we implement Bonferroni correction [16], as we are performing multiple comparison tests.

We repeat the same analysis with 99 other bootstrap samples from the chronologically first 70% of the observations of TF. We end up with 100 base-

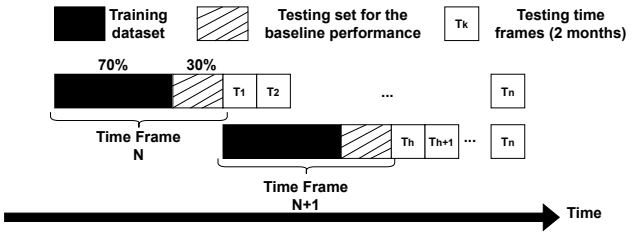


Fig. 7: Overview of model evaluation under concept drift for a specific time frame size.

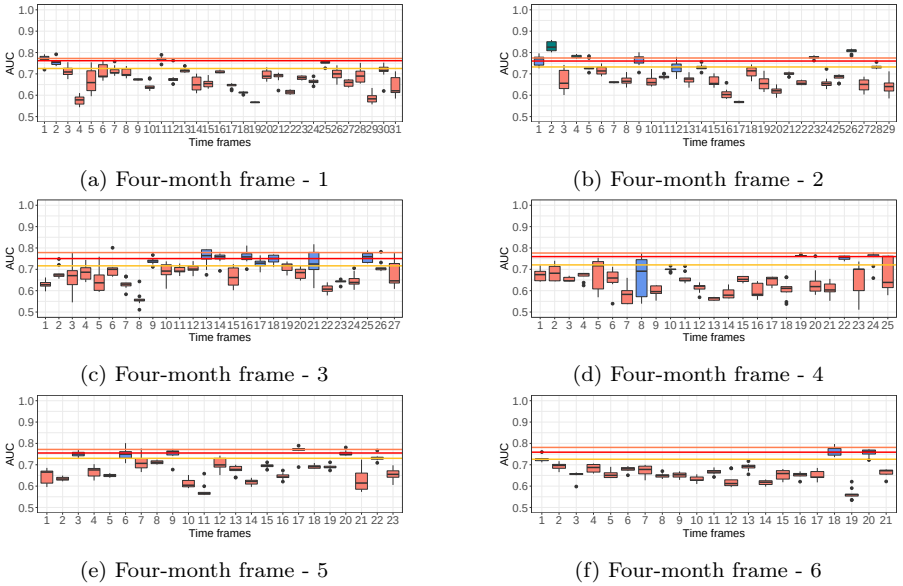
line performance measurements and 100 performance measurements for each of the two month testing time frames.

We then repeat all previous steps with a different training window size. Starting with a time frame of four months (first 70% for training and remaining 30% to measure the baseline evaluation) up to 24 months, with an increment of two months.

Results: Across the different time frame sizes, the performance of our evaluated LLPs (Shallow and deep) drops significantly a median of 1.5 (Hadoop) or one (Spring and OpenStack) testing time frame after the end of the training data period. For example, we observe that the Shallow-LLP trained using four-month time frames (i.e., four-month based Shallow-LLP) takes a median[10] of 1.5, 1 and 1 testing time frames to drop statistically significantly (Wilcoxon test; $\alpha = 0.01$) below the baseline performance in terms of AUC for Hadoop, Spring and OpenStack respectively. Meanwhile, the four-month based DL-LLP takes a median[12] of one, one and 1.5 time frames to drop below the baseline performance.

Furthermore, the performance of our Shallow-LLP is statistically significantly (Wilcoxon test; $\alpha = 0.01$) lower than the baseline performance in a median[12] of 56% (observed for 24-month based Shallow-LLPs) to 85% (observed for 10-month based Shallow-LLPs) of the testing time frames for Hadoop. The same median[12] percentage ranges from 60% to 96% and 30% to 81% for Spring and OpenStack respectively. Figures 8, 9 and 10 highlight the concept drift for our evaluated four-month based Shallow-LLPs. We observe similar findings for our DL-LLPs (shown in Figures 11, 12 and 13), as the performance on testing time frames is statistically below the baseline performance in a median[12] of 78% (observed for four month based DL-LLPs) to 100% (observed for eight-month based DL-LLPs) of the testing time frames of Hadoop. Such a median[12] of time frames in which the DL-LLPs perform statistically significantly below the baseline ranges between 79% and 98% and between 89% and 100% for the respective DL-LLPs of Spring and OpenStack. Finally, we observe that the performance of our models (Shallow and DL) statistically significantly exceeds the baseline performance in a median[11] time ranging from

[10] Median over the 100 bootstrap samples.
[11] Median over the 100 bootstrap samples.



(a) Four-month frame - 1

(b) Four-month frame - 2

(c) Four-month frame - 3

(d) Four-month frame - 4

(e) Four-month frame - 5

(f) Four-month frame - 6

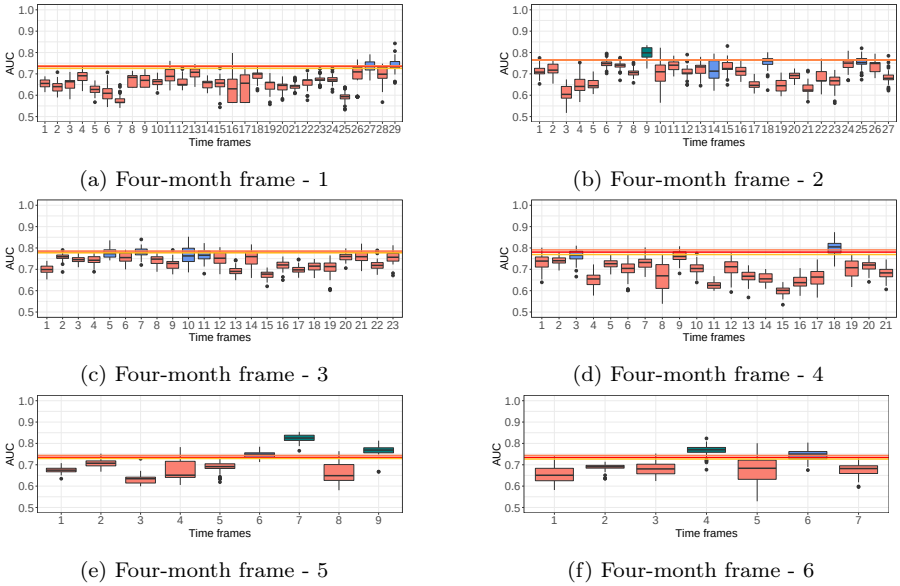
Fig. 8: AUC performance of Hadoop's Shallow-LLPs across time. the lines represent the 1st quantile, median and 3rd quantile for the baseline AUC performance. Red boxplots show time frames with a performance statistically below the baseline, blue boxplots show no statistical difference and green boxplots show time frames with a performance statistically better than the baseline.

0% to 8%, 0% to 13% and 0% to 40% (Shallow-LLP) and in 0% to 0%, 0% to 50% and 0% to 0% (DL-LLP) of the testing time frames of Hadoop, Spring and OpenStack respectively. This range depends on the size of the training time frame.

The diminishing performance of the LLPs can be explained by the fast increase in the number of statistically different independent features between the baseline model data and the future test sets. In fact, we observe that at least 50% of the features leveraged by the Shallow-LLP model, including those features related to the log message and the context of the logging statement (used to train the DL-LLP) showed statistically significant differences between the baseline data and the data of the testing time frames, after a median[12] time ranging between one and three (Hadoop), 1 and 1.5 (Spring) and 1 and 5.5 (OpenStack) time frames.

Additionally, we observe that 52%, 56% and 34% of the features of adjacent testing time frames of Hadoop, Spring and OpenStack are statistically significantly different (Wilcoxon, α=0.01). For example, the churn of logging statements for the OpenStack project in the time frame between August and October 2014 (a median of 36) is statistically significantly different (Wilcoxon

[12] Median over time frame sizes from 4 to 24.



(a) Four-month frame - 1

(b) Four-month frame - 2

(c) Four-month frame - 3

(d) Four-month frame - 4

(e) Four-month frame - 5

(f) Four-month frame - 6

Fig. 9: AUC performance of Spring's Shallow-LLPs across time. the lines represent the 1st quantile, median and 3rd quantile for the baseline AUC performance. Red boxplots show time frames with a performance statistically below the baseline, blue boxplots show no statistical difference and green boxplots show time frames with a performance statistically better than the baseline.

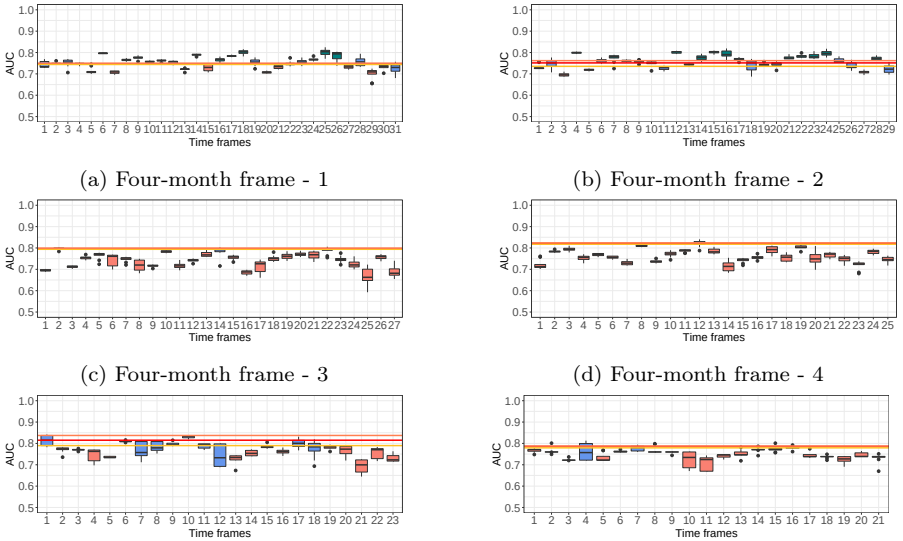
test; $\alpha = 0.01$) than the churn for the logging statements of the same project (i.e., OpenStack) in the following time frame (a median of 31). In fact, maintainers of OpenStack conducted a number of logging maintenance[13] activities (e.g., changing verbosity of logs) in the period between August and October 2014, which resulted in a higher churn for logging statements in that period (median 36) compared to the next time frame (median churn 31) as well as the previous time frame (median churn 21). Only 0%, 1% and 2% of the adjacent testing time frames had the same distribution for Hadoop, Spring and OpenStack, respectively.

Across the different training time frame sizes, the performance of 27% (Hadoop), 63% (Spring) and 72% (OpenStack) of our Shallow-LLPs exceeds the baseline performance after dropping below it. In fact, our Shallow-LLPs statistically significantly (Wilcoxon test; $\alpha = 0.01$) exceed the baseline performance on a median[14] of 0% (observed for the four-month based models) to 4.1% (observed for the 18-month based models) of the testing time frames for Hadoop. Similarly, this occurs on a median[15] of 0% to 13% and 0% to 44% of the testing time frames of Spring and OpenStack respec-

[13] https://github.com/openstack/openstack/commit/56cc320240c983742c467f7afd7cc6b11dde8625

[14] Median over the 100 bootstrap samples.

[15] Median over the 100 bootstrap samples.



(a) Four-month frame - 1

(b) Four-month frame - 2

(c) Four-month frame - 3

(d) Four-month frame - 4

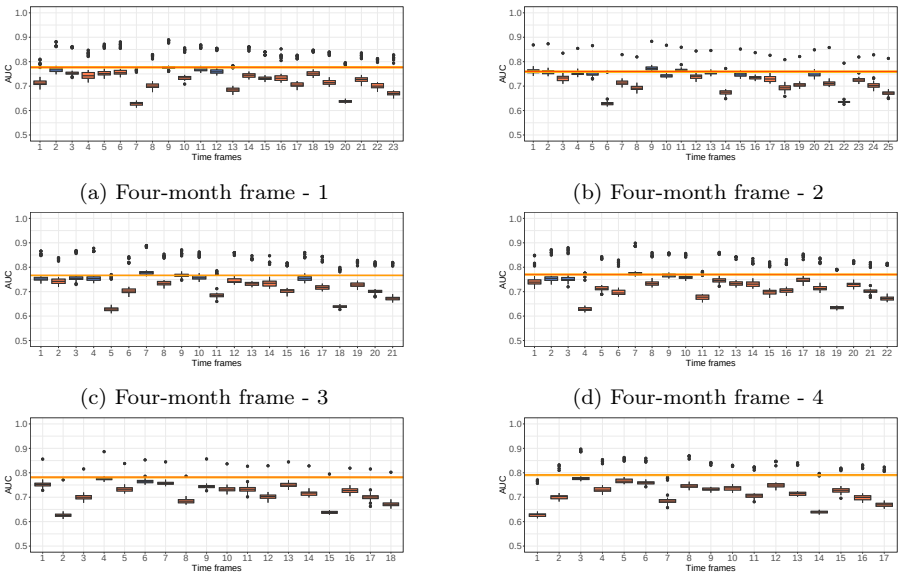
(e) Four-month frame - 5

(f) Four-month frame - 6

Fig. 10: AUC performance of OpenStack's Shallow-LLPs across time. the lines represent the 1st quantile, median and 3rd quantile for the baseline AUC performance. Red boxplots show time frames with a performance statistically below the baseline, blue boxplots show no statistical difference and green boxplots show time frames with a performance statistically better than the baseline.

tively. For example, the four-month based model for all our evaluated projects statistically significantly exceeds the baseline performance in a median[15] 0% of the testing time frames, as shown in Figures 8, 9 and 10.

Concept drift might be a more serious problem for DL-LLPs, as their performance typically never becomes as good as the baseline performance after dropping below it. In fact, we observe that the performance of a median of 0% (Hadoop), 50% (Spring) and 0% (OpenStack) of the DL-LLPs (depending on the training frame size) exceeds the baseline performance after dropping below it, which indicates that DL-LLPs are less likely to re-exceed the baseline performance compared to Shallow-LLPs.



(a) Four-month frame - 1

(b) Four-month frame - 2

(c) Four-month frame - 3

(d) Four-month frame - 4

(e) Four-month frame - 5

(f) Four-month frame - 6

Fig. 11: AUC performance of Hadoop's DL-LLPs across time. the lines represent the 1st quantile, median and 3rd quantile for the baseline AUC performance. Red boxplots show time frames with a performance statistically below the baseline, blue boxplots show no statistical difference and green boxplots show time frames with a performance statistically better than the baseline.

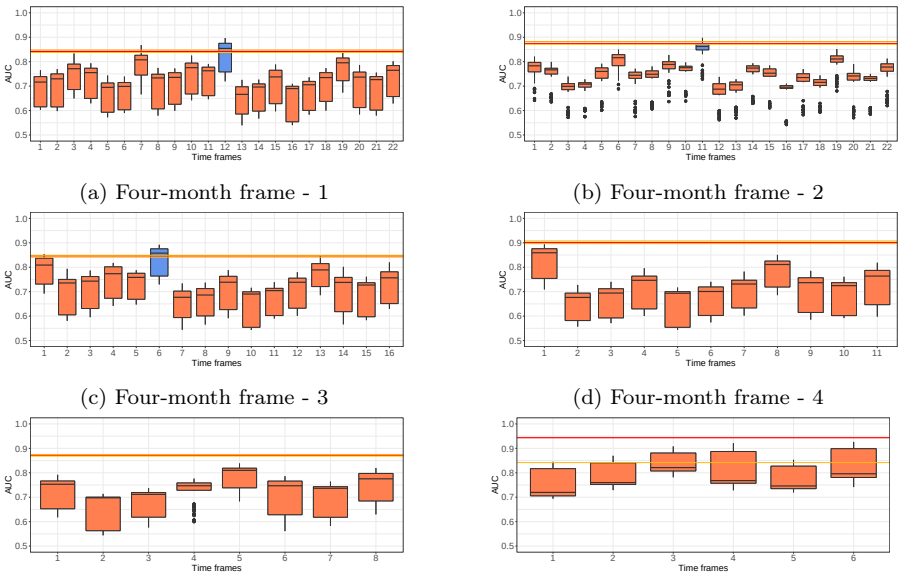
Summary of RQ2

Log level prediction models suffer from concept drift as their AUC performance drops on future testing time frames after a median[a] of just 1.5, 1 and 1 testing time frames for Hadoop, Spring and OpenStack respectively. The effect of concept drift is more severe on the DL-LLPs, for which the performance drops significantly below the baseline performance in 22% (Hadoop), 19% (Spring) and 59% (OpenStack) more testing time frames than the Shallow-LLP. **Our results suggest the need for continuous concept drift tracking and frequent updates to the LLPs, especially the DL-LLP, whose performance is less stable throughout time compared to the Shallow-LLP.**

[a] Median over the time frame sizes from 4 to 24 months.

RQ3. What is the best performing time-based modeling strategy for log level prediction models?

Motivation: Since previous research questions suggest using time-aware approaches, we explore different strategies to train time-aware models as an



(a) Four-month frame - 1

(b) Four-month frame - 2

(c) Four-month frame - 3

(d) Four-month frame - 4

(e) Four-month frame - 5

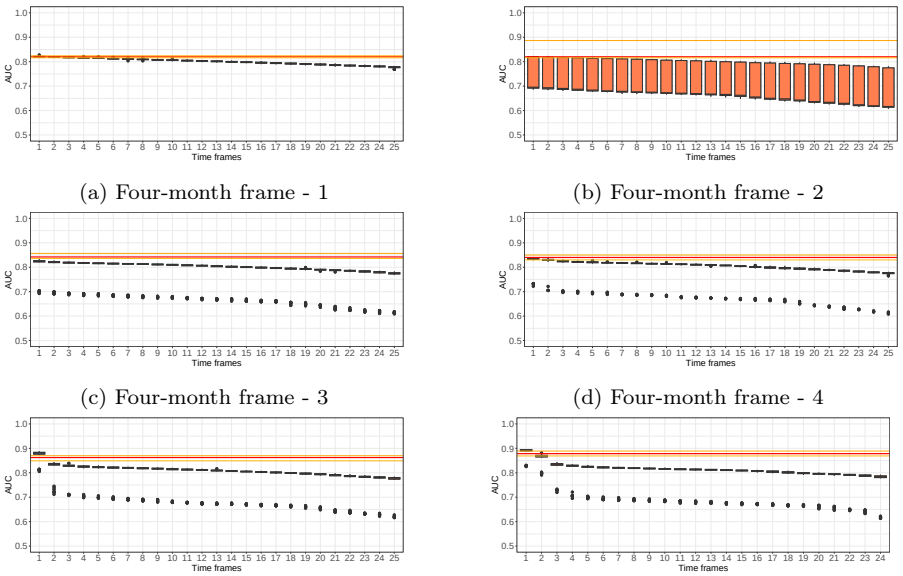
(f) Four-month frame - 6

Fig. 12: AUC performance of Spring's DL-LLPs across time. the lines represent the 1st quantile, median and 3rd quantile for the baseline AUC performance. Red boxplots show time frames with a performance statistically below the baseline, blue boxplots show no statistical difference and green boxplots show time frames with a performance statistically better than the baseline.

approach to mitigate the impact of concept drift. Specifically, we evaluate two time-aware modeling strategies in this research question: *contextual model* that leverages recent data and *all-knowing-model* that leverages the whole history of data. In particular, leveraging just the most recent data in a contextual model might not benefit from recurring events that exist throughout the whole history of data. On the other hand, leveraging the whole available historical data in an all-knowing-model might contain noisy data that are drifting from the current data. In this research question, we quantify such a trade-off by comparing the two approaches for both the Shallow-LLP and the DL-LLP to identify which of the two approaches better fits the log level prediction models..

Approach: To compare the two time-aware models, we split the whole history of the available data into two-month time frames, which are used as testing time frames. We compare on each of these testing time frames how our two evaluated types of models perform. We train and test our two types of models as discussed below:

– *All-knowing model*: We train a model that leverages all the existing data prior to each of our testing time frames. For example, we train a model



(a) Four-month frame - 1

(b) Four-month frame - 2

(c) Four-month frame - 3

(d) Four-month frame - 4

(e) Four-month frame - 5

(f) Four-month frame - 6

Fig. 13: AUC performance of OpenStack's DL-LLPs across time. the lines represent the 1st quantile, median and 3rd quantile for the baseline AUC performance. Red boxplots show time frames with a performance statistically below the baseline, blue boxplots show no statistical difference and green boxplots show time frames with a performance statistically better than the baseline.

on data from the first three years for a testing time frame that covers the 37th and 38th months of a project.

– *Contextual models*: We train a contextual model on data that belongs to N months prior to each of our testing time frames. For our experiment, we evaluated different time frame sizes (i.e., N) that range from four to 24 months, with two months increment, to train our contextual models. In other words, we train a four-month, six-month, eight-month, and up to 24-month based models for each of our testing time frames. For the early testing time frames, we train contextual models based on the amount of existing data. For example, we train a four-month to 12-month models for the testing time frame that covers the 13th and 14th months of a project. Note that we consider the 12-month model, in our example, as the all-knowing model since it is trained on the whole available data.

For both types of models, we evaluate 100 models, each on a different bootstrap sample such that we obtain for each model 100 performance measurements to use for a statistically robust comparison. Note that the comparison of two models is performed on the same test time frame. In other words, we do not compare two models on different testing time frames, as shown in Fig-

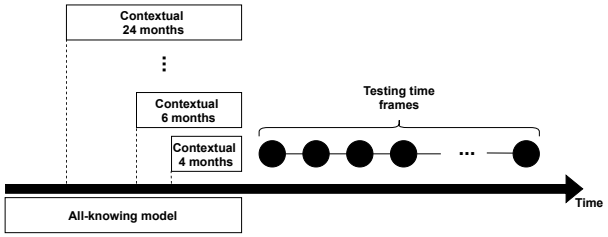


Fig. 14: Comparison of contextual and all-knowing models. We compare the models shown in the figure just on their first following testing time frame. We train new contextual and all-knowing models for each testing time frame.

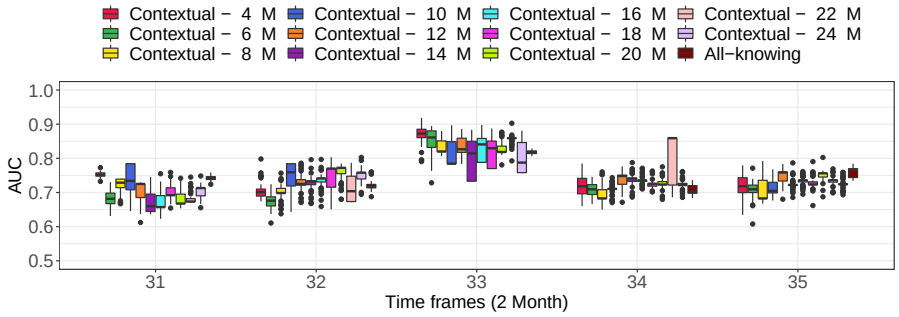
ure 14. Note that the same approach is followed for both the Shallow-LLP and the DL-LLP.

Results: We observe that at least one contextual Shallow-LLP statistically significantly outperforms the all-knowing Shallow-LLP on 94%, 82% and 86% of testing time frames for Hadoop, Spring and OpenStack respectively. We also observe that the all-knowing Shallow-LLPs statistically outperform all the contextual Shallow-LLPs in only 5%, 8%, and 0% of our Hadoop, Spring and OpenStack testing time frames, respectively.

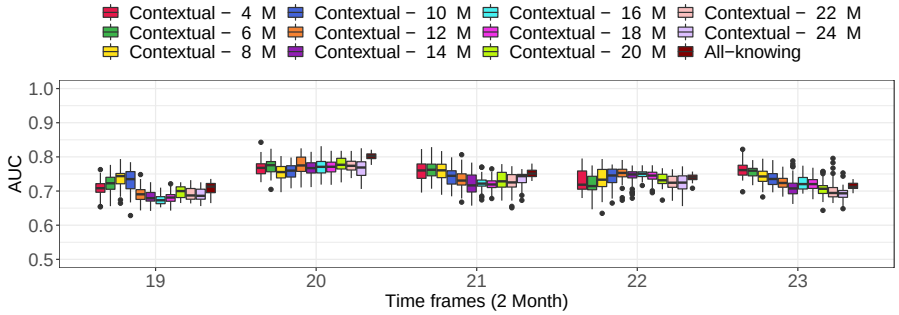
While 51%, 31% and 22% of the cases in which the contextual Shallow-LLP outperforms the all-knowing model have a large effect size (Cohen's d) for Hadoop, Spring and OpenStack respectively, we observe that 0% of the cases in which the all-knowing Shallow-LLP outperforms the contextual Shallow-LLPs with a large effect size for both Hadoop and Spring projects.

Furthermore, across the different testing time frames, we observe a median of one, three and three contextual Shallow-LLPs that outperform the all-knowing Shallow-LLP for Hadoop, Spring and OpenStack respectively . For example, three Spring contextual models (6-month, 8-month, and 10-month based contextual models) outperform the all-knowing model on the nineteenth testing time frame, while the all-knowing model outperforms the 22-month and 24-month based contextual model on the same testing time frame.

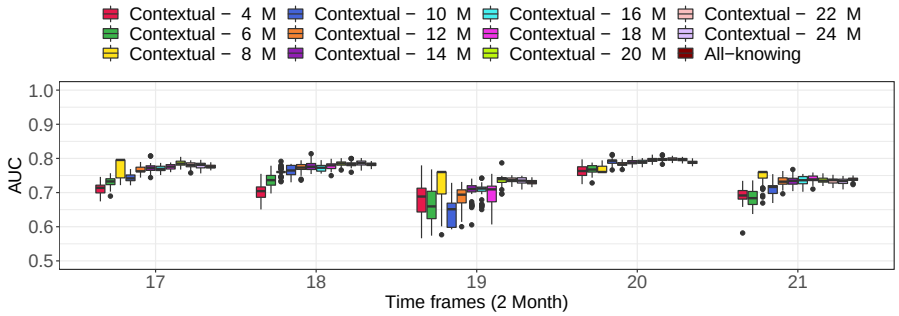
As for the DL-LLPs, we do not observe a significant performance improvement when comparing the all-knowing and the best contextual approach, as shown in Figure 16. In fact, the AUC performance difference between the best contextual DL-LLP and the all-knowing DL-LLP is negligible on 83%, 82% and 92% of the testing time frames of Hadoop, Spring and OpenStack respectively. One reason why contextual DL-LLPs are not performing as well as the Shallow-LLPs compared to their respective all-knowing LLPs might have to do with the data quality vs. quantity trade-off [5]. In fact, the contextual DL-LLPs are trained on good quality data (i.e., avoiding noise of irrelevant past data) but that data might not be large enough for a data-hungry model such as the DL-LLP.



(a) Hadoop



(b) Spring



(c) OpenStack

Fig. 15: The AUC Performance of contextual and all-knowing Shallow-LLPs on a selection of testing time frames (two-month long) - The remaining time frames (e.g., 1 to 35 for Hadoop) are available in the appendix.

As we do not observe a statistical difference between the performance of all-knowing and contextual DL-LLPs, we focus in the remainder of this RQ on contextual Shallow-LLPs for which we consistently observe at least one contextual model that exceeds the performance of the all-knowing Shallow-LLP.

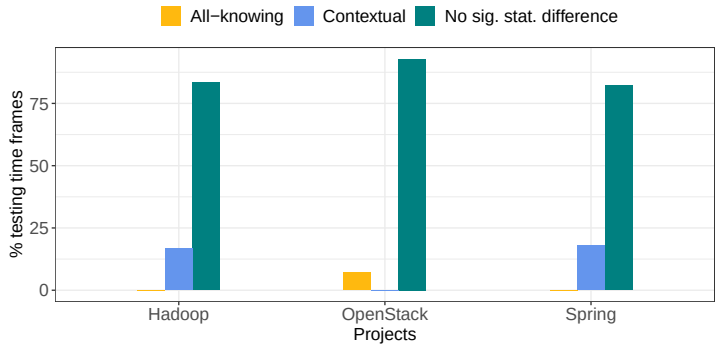


Fig. 16: Percentage of testing time frames in which one of our time-aware DL-LLPs (i.e., contextual or All-knowing) is the statistically significantly best performing model in terms of AUC.

No contextual Shallow-LLP consistently performs the best on all of our evaluated testing time frames, as shown in Figure 17. In fact, we do not observe a clear pattern about contextual models that perform the best, as six, seven and eight different contextual models perform the best on at least one testing time frame of Hadoop, Spring and OpenStack respectively. For example, the best Hadoop contextual model for the 33rd testing time frame is the four-month based-model, while it is the 16-month based model for the 34th testing time frame.

Despite performing well on some testing time frames, a contextual model can poorly perform on other time frames. For example, the four-month based contextual model of Hadoop has a median[16] AUC of 0.86 on the 33rd testing time frame as shown in Figure 15, while the same contextual model has a median[16] AUC performance of just 0.72 on the 34th time frame.

Additionally, we observe that the size of the contextual model (i.e., number of months used to train it) has a negligible to moderate correlation with the performance of our evaluated contextual models, as Spearman's correlation between the size and performance of our contextual models is -0.05, -0.3 and 0.47 for Hadoop, Spring and OpenStack respectively.

We also investigate ML modeling techniques aiming to improve the performance our time-aware Shallow-LLPs. Specifically, we implement two ML models that take advantage of the nature of our evaluated time-aware approaches. While the first is an ensemble model [60] in which our contextual Shallow-LLPs vote for the appropriate log level across all frame sizes, the varying model leverages a subset of features to train an all-knowing Shallow-LLP. We chose a threshold of six features for our varying models (i.e., a maximum of six most varying features can be removed), in order to guarantee enough

[16] Median over the 100 bootstrap samples.

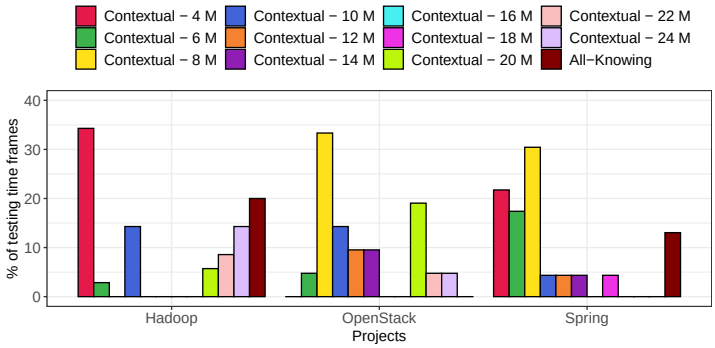


Fig. 17: Percentage of testing time frames in which the contextual Shallow-LLPs perform the best in terms of AUC.

features for training representative varying models (i.e., avoid issues merely due to lack of features).

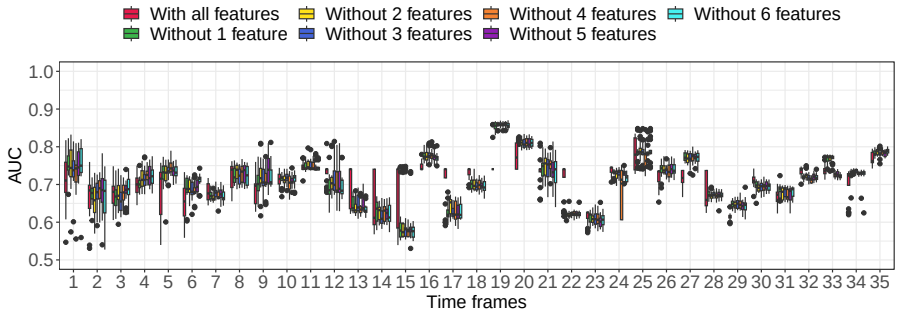
We observe that the ensemble learning approach does not bring any significant improvement to the performance. In fact, the best performing contextual Shallow-LLP outperforms the ensemble learning model in 95%, 95% and 82% of the testing time frames. Additionally, the improvement (if any) brought by the ensemble learning approach to the best performing time-aware Shallow-LLP on a given time frame is negligible for all of our studied projects except for one time frame for Hadoop project.

As for our all-knowing varying Shallow-LLPs (shown in Figure 18), we do not observe a significant difference between the performance of the model trained using all features and the performance of models trained using all features except the n features (n from 1 to 6) that vary the most. Specifically, we observe that the models omitting most varying features (one or more) outperform the model that uses all features in only 25% (Hadoop), 43% (Spring) and 55% (OpenStack) of our testing time frames.

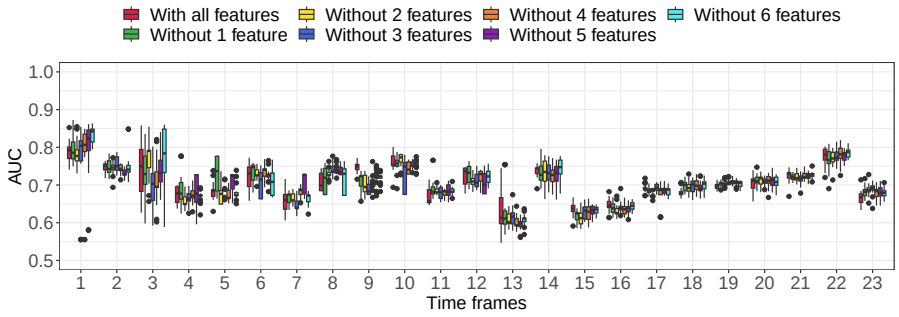
Summary of RQ3

While there is no difference in terms of performance between the contextual and all-knowing DL-LLPs, at least one contextual Shallow-LLP statistically outperforms the all-knowing Shallow-LLP on our evaluated testing time frames. **Our results suggest including the size of the contextual model as a hyperparameter to tune when experimenting with contextual LLPs.**

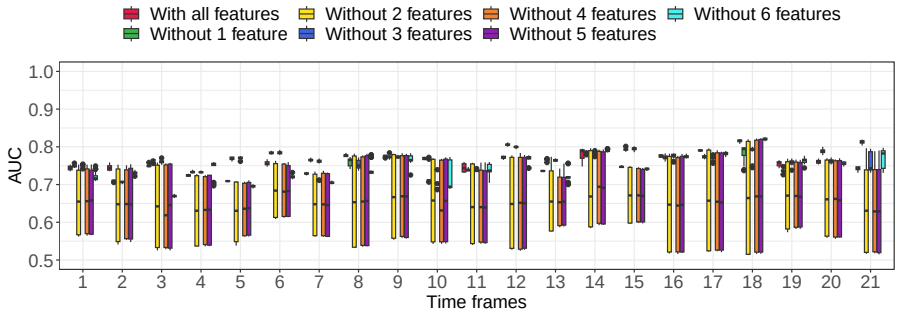
RQ4. How does the interpretability of the log level prediction models change over time?



(a) Hadoop



(b) Spring



(c) Spring

Fig. 18: The AUC Performance of the varying all-knowing Shallow-LLP on all testing time frames (two-month long)

Motivation: The goal of this research question is to quantify how the interpretability changes over time for Shallow-LLPs (which our previous RQ indicates as the more fitting for handling time-related issues), so that practitioners can better understand what impacts logging level decisions in a particular time period. In fact, previous findings also indicate that the performance of Shallow-LLPs trained using a time-aware approach can change significantly from one

time frame to the other (regardless of the training time frame size). Such changes can be due to changing logging patterns across time. Therefore, we investigate if the change in performance is accompanied with an interpretability change, so updating models is important for understanding the important factors related to log level prediction. The interpretability is relevant (especially for models not leveraging deep learning such as the Shallow-LLP) for practitioners in their decision making such that shifting or conflicting feature importance rankings might be confusing for practitioners.

Approach: To investigate log level prediction models interpretability across time, we train time-aware Shallow-LLPs following the steps below, which are highlighted in Figure 19.

We split the history of our studied project into equal time frames (TF) of size S (e.g., four months). For each time frame, we train 100 log level prediction models using 100 bootstrap samples, similarly to the other research questions.

We then extract a ranking of important features from each of the 100 models, as well as the positive or negative impact each of these features can have on the log level prediction, similar to prior work [47, 45]. For instance, a feature can have a positive impact on the prediction if increasing the value of that feature increases the prediction probability and vice-versa. To determine if a feature F has a positive or negative effect on a log level L, we first calculate the probability (P1) of predicting L using features that are set at their median values. Next, we increase the value of the feature F by one standard deviation from its median while keeping the other features at their median values, and re-predict the probability (P2) of the log level L. The two probabilities P1 and P2 are then compared to determine the type of impact feature F has on log level L.

Since each model has a different ranking of features, we cluster the 100 rankings using the Scott-Knott clustering algorithm into a single ranking. Similarly, we summarize the impact of a feature as a vote. A feature has a positive impact if that feature has a positive impact on the majority of our 100 trained models. Note that we rarely observe different impacts of the same feature on the 100 models that are trained on the same time frame.

We repeat the same analysis on each of the following time frames to obtain a ranking as well as the impact of each important feature.

We then statistically compare each pair of the obtained rankings, as well as the rankings of the pair of successive time frames using Spearman Correlation. Since we are conducting multiple comparison tests, we additionally leverage Bonferroni's correction [16]. Note that a ranking correlation ranging between 0 and 0.4 is considered weak to very weak, a ranking correlation ranging between 0.4 and 0.6 is considered moderate, and a ranking correlation higher than 0.6 is considered strong to very strong. Additionally, we leverage the information about features' impact in order to detect contradictory features (i.e., features with different impact across two or more time frames). Specifically, we compare whether a feature has a positive impact on the models of a given time frame, while it has a negative impact on the models of another time frame.

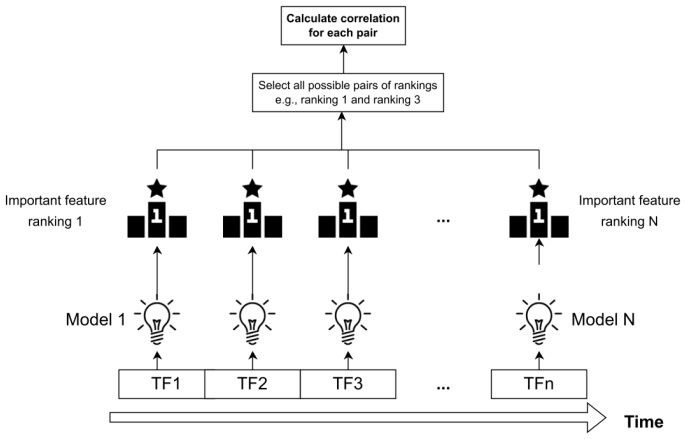


Fig. 19: Overview for comparison of interpretability across time

We repeat the same analysis with different time frame sizes, starting from four and up to 24 months similarly to the previous research questions. Note that this research question focuses on contextual Shallow-LLPs, as they outperform the all-knowing Shallow-LLP (according to our findings of RQ3).
Results: Across different time frame sizes, a median of 88%, 80% and 79% of the model pairs exhibit a very weak to weak correlation in terms of feature rankings for Hadoop, Spring and OpenStack respectively. For instance, we observe 90%, 87% and 79% of model pairs trained on a six-month time frame exhibit a very weak to weak important features correlation. Furthermore, we report a median (across different time frame sizes) of 12%, 20% and 27% of model pairs that have a strong to very strong correlation in terms of important features ranking. For example, only 14%, 20% and 20% of the six months-based model pairs have strong to very strong important features correlation for Hadoop, Spring and OpenStack, respectively.

Furthermore, successive pairs of models do not have consistent rankings of the most important features. We observe that the ranking of important features for a median (across different time frame sizes) of 31%, 40% and 0% of the successive training time frames is weak to very weak for Hadoop, Spring and OpenStack respectively. For instance, the percentage of successive frames with weakly correlated ranking of important features when considering six-month based models is 33%, 29% and 50%. Meanwhile, the percentage for successive time frames with strongly correlated features is 13%, 42% and 21% for Hadoop, Spring and OpenStack respectively.

Additionally, we observe that the median (over our evaluated time frame sizes) percentage of important features that are common across all models is 40%, 54% and 44% for Hadoop, Spring, and OpenStack. That percentage increases as the size of the frame gets smaller, as the correlation between the size of the time frame and the number of different features is strongly negatively correlated (Spearman's ρ=-0.9, -0.93 and -0.97 for Hadoop, Spring

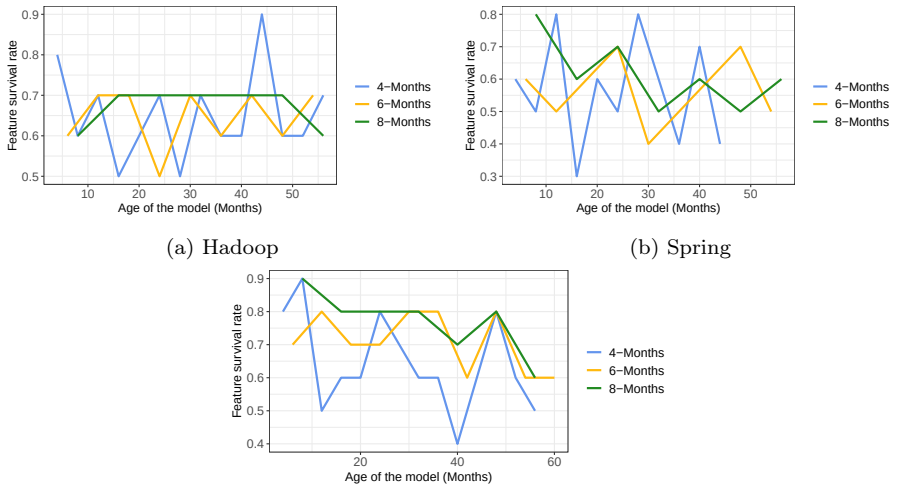
and OpenStack respectively). For example, while the percentage of common features for Hadoop models that are trained on six months time frames is 26%, that percentage is 94% for models trained on frames with a size of 48 months. We think that training models on small time frames (assuming enough data points for training) allows to unveil specific patterns to those time periods (reflected by the highly different feature importance results), as opposed to models trained on larger time frames for which feature importance becomes similar. For example, OpenStack's three models that are trained on 24 months time frames have 17 out of 18 of their important features in common.

Despite observing models trained on different time frames with common important features, these features can have a contradictory effect. In fact, the median percentage of contradictory features over shared features between a pair of models ranges from 0% to 40% (depending on the evaluated time frame size), 0% to 30% and 0% to 22% for Hadoop, Spring and OpenStack, respectively. For example, we observe that while having more variables in a logging statement increases the probability of that logging statement to have the *Info* log level according to Hadoop's six-month based-models trained between May and November 2012, the opposite is observed for the model trained on the next six months time frame, as increasing the number of variables within a logging statement reduces the probability of predicting the log level *Info*.

We observe that the percentage of important features that remain important after retraining a log level prediction model later in time (aka., feature survival rate) exhibits a different pattern than models studied by a prior study [36]. While brown test (i.e., tests that trigger false positive build failures) prediction models studied by Olewicki et al. [36] show a monotonically decreasing feature survival rate, our log level prediction models show a fluctuating feature survival rate, as shown in Figure 20. Across the different training sizes, the median feature survival rate ranges from 60% to 70%, 55% to 65% and 60% to 80% for Hadoop, Spring and OpenStack respectively.

Summary of RQ4

The interpretability of log level prediction models changes across time, as the important features are different across a median of 88%, 80% and 79% of the model pairs for Hadoop, Spring, and OpenStack respectively. The smaller the time frame, the more unique its models' interpretability. **Our results suggest the use of recently trained models for more accurate interpretability.**



(a) Hadoop

(b) Spring

(c) OpenStack

Fig. 20: Feature survival rate for log level prediction models. For clarity purpose, we only show the four, six and eight months based models.

Implications

Data leakage:

Our findings suggest leveraging time-aware approaches for shallow and DL-LLPs for more realistic performance estimations. Log level prediction models trained using the random training/testing data splitting approach can overestimate the expected performance of a time-aware log level predictor by a median AUC up to 17% (Shallow-LLP) and 27% (DL-LLP). The existence of data leakage in the state-of-the-art LLPs indicates that performances reported in literature might not be achievable when practitioners integrate LLPs into their environments. Therefore, when training LLPs on their own data, practitioners need to take into consideration data leakage. Specifically we recommend evaluating both Shallow and DL LLPs using a time-aware approach. Specifically, the train dataset needs to be chronologically before the testing and validation datasets. As for the size of the time frame of training, we recommend that practitioners include that size into the hyperparameter they tune when searching for the optimal LLP. Such a time frame size has an impact on the performance, especially for Shallow-LLP.

Action steps: Implement a chronological split strategy rather than a random one, when sampling LLP data to mirror real-live use cases (i.e., the training data should always be before the testing data). Our paper describes how one can evaluate time-aware strategies (e.g., contextual LLPs) to optimize their own LLP's performance while ensuring it is free from data leakage.

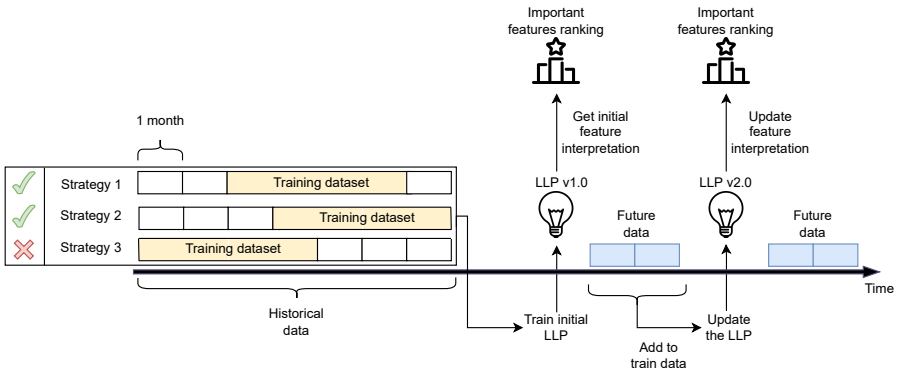


Fig. 21: Concept drift mitigation approach

Concept drift:

Our results suggest the need for continuous concept drift tracking and frequent updates to the LLPs (both Shallow-LLP and DL-LLP). Log level prediction models suffer from concept drift as their AUC performance drops on future testing time frames after a median of just 1.5, 1 and 1 testing time frames for Hadoop, Spring and OpenStack respectively. The existence of concept drift in LLPs imposes the necessity of updating them frequently in order to avoid obsolete models. While it is not possible to completely eliminate concept drift, our findings indicate that concept drift does not become significant until 2 months after the training dataset for both Shallow and DL LLPs. Therefore, for training an initial LLP we propose to use historical data that ends two months at most before the testing phase, as shown in Figure 21. For instance, if one wishes to use an LLP to predict log levels starting March 2024, the end date of the data leveraged for training the model should not be older than December 2023. Furthermore, the trained LLP should be updated every two months. For example, the model trained in March 2024 will need to be updated come May 2024, with data from the March and April months added to the training dataset. Note that retraining the LLPs frequently can be costly (especially for the DL-LLPs), yet, we believe that the benefit brought by having an LLP that is robust to logging changes outweighs the efforts for model maintenance (e.g., the periodic LLP retrain could be automated).

Action steps: (1) Update the training data for LLPs at intervals dictated by our concept drift detection methodology (e.g., intervals of less than two months, as a rule of thumb based on our empirical findings) to mitigate concept drift impact. (2) Continuously monitor LLPs' performance over time and adjust the retrain frequency based on observed performance changes.

Time-aware models:

We advise practitioners to leverage contextual models (i.e., time-aware models trained on a subset of the available data), rather than the all-knowing models (i.e., time-aware models that leverage all available data), as the former option shows (1) better performances for Shallow-LLPs, (2) at least as good as the all-knowing model in terms of performance for the DL-LLP, and it is easier to retrain (especially for DL-LLPs) which makes model updates and hyperparameter tuning activities easier.

We also recommend that practitioners update the ranking of important features whenever they update the LLP, as shown in Figure 21. The feature importance ranking for time-aware models is typically different. Thus, when opting for a best time-aware performing model (i.e., either a contextual model or all-knowing model), practitioners should consider regenerating the most important features.

Action steps: (1) Leverage contextual LLPs to guarantee optimal performance and facilitate easier retraining (crucial for concept drift mitigation) and hyperparameter tuning. (2) Update the feature importance rankings each time the LLP is retrained to maintain the relevancy of the LLP interpretability in light of new data.

5 Threats to validity

5.1 External validity

One external threat to the validity of our work concerns the generalization of our results to other software systems as well as models. Even if our study covers three popular, large software projects maintained for a long period, and our experiments targeted different time frames and time frame sizes, we do not generalize our results to other software systems or other machine learning models. We also encourage future studies to replicate our study on other software engineering models as well as software systems.

5.2 Internal validity

One internal threat to the validity of our work regards the time frame sizes used to train and test our models. For instance, leveraging a different time-frame size can show a different result. To mitigate this threat, we leverage different time-frame sizes ranging from four to 24 months and we leverage 100 bootstrap samples for each trained model to make our analysis statistically robust. Similarly, our comparison between the all-knowing models and the contextual ones can be different with different contextual models sizes. To mitigate this threat, we also compared the all-knowing models to different

contextual models, each of which is trained on a different sample size of four to 24 months. Another internal threat to validity regards the quality of the log levels selected by developers. In fact, developers can introduce inaccurate log level choice which can impact the overall quality of our datasets. However, we observe that the logging statements in our datasets are not changed frequently (3.8% to 7.5% of the logging statements across our evaluated projects).

6 Conclusion

Log level prediction models proposed by prior work [28, 29] leverage a set of metrics to predict the appropriate log level for a logging statement. While their models (Shallow-LLP and DL-LLP) show a good performance, the way these are evaluated can suffer from data leakage and concept drift, i.e., two time-related issues. Due to the change in log level choice decisions across time as well as the context differences (e.g., domain, data) between the LLPs and previously studied models (i.e., defect prediction and AIOps models), one might be unclear whether LLPs are less impacted by data leakage and concept drift (e.g., stagnant logging practices), or whether they can be severely impacted by time-related issues (e.g., abrupt changes to logging strategy).

Since prior work [58, 4] indicates that time-related issues (e.g., concept drift) impact software engineering models differently. Therefore, in this paper, we quantify the impact of data leakage and concept drift on the performance and interpretability of log level prediction models.

Our findings indicate that a data leakage risk exists for both the Shallow and DL-LLPs. Furthermore, we found that LLPs (especially DL-LLPs) need to be updated frequently as they can suffer from concept drift as soon as a couple of months post training.

As a means of mitigating time-related issues, we evaluated two types of time-aware models: *a)* contextual models that leverage data from a time frame of the history, and *b)* the all-knowing model that leverages all the data available at the moment of its creation. Through the comparison of the performance of these time-aware models, we observe that the all-knowing Shallow-LLP – despite using more data– can perform significantly worse than some contextual Shallow-LLPs. Yet, no specific contextual Shallow-LLP consistently outperforms all the other contextual Shallow-LLLPs. Therefore, we encourage considering the size of the contextual model as a hyperparameter to tune when training log level prediction models spanning through time.

Finally, we investigated how the important features for log level prediction change for a time-aware model. We observe that while some features are shared between all the contextual models, a larger portion (i.e., up to 40%) of the features are exclusive to a set of the contextual models. Furthermore, even when two contextual models share some important features, they may have contradictory effect on the prediction of a specific log level.

Our results suggest paying attention to time-related issues when leveraging log level prediction models. To mitigate the effect of these time-related issues,

we recommend using contextual models that are time-aware (i.e., not susceptible to data leakage), easier to train than the global model and –given the right window size– perform at least as good as the global models. While finding the best performing shallow model is relatively fast (i.e., real-time retraining), we seek to develop in future work an approach to find the best contextual model for DL-LLPs.

Acknowledgements: We would like to thank the anonymous reviewers for their insightful comments.
Data Availability Statement: The datasets generated during and/or analysed during the current study are available online [38].

Declarations:

Conflicts of Interests: The authors declare that they have no conflict of interest.

References

1. *rms: Regression Modeling Strategies.*
2. A. Agrawal and T. Menzies. Is ai different for se? *NC State Univ.*, 12 2019.
3. H. Anu, J. Chen, W. Shi, J. Hou, B. Liang, and B. Qin. An approach to recommendation of verbosity log levels based on logging intention. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019.
4. K. E. Bennin, N. bin Ali, J. Börstler, and X. Yu. Revisiting the impact of concept drift on just-in-time quality assurance. *2020 IEEE 20th Int. Conf. on Software Quality, Reliability and Security (QRS)*, 2020.
5. T. Bertram, J. Fürnkranz, and M. Müller. Quantity vs quality: Investigating the trade-off between sample size and label reliability, 2022.
6. R. Chahar and D. Kaur. A systematic review of the machine learning algorithms for the computational analysis in different domains. *International Journal of Advanced Technology and Engineering Exploration*, 7(71):147, 2020.
7. Chakkrit Tantithamthavorn. *ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test*, 2018.
8. J. B. Ekanayake, J. Tappolet, H. C. Gall, and A. Bernstein. Tracking concept drift of software projects using defect prediction quality. *2009 6th IEEE Int. Working Conference on Mining Software Repositories*, pages 51–60, 2009.
9. Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. Where do developers log? an empirical study on logging practices in industry. In *Proceedings of the 36th International Conference on Software Engineering(ICSE'14)*, page 24–33, 2014.

10. J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. volume 8, pages 286–295, 09 2004.
11. J. a. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014.
12. G.Ditzler, M.Roveri, C.Alippi, and R.Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.
13. D. J. Hand and R. J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Mach. Learn.*, 45(2):171–186, 2001.
14. M. Harel, K. Crammer, R. El-Yaniv, and S. Mannor. Concept drift detection through resampling. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1009–II–1017. JMLR.org, 2014.
15. F. E. Harrell. *Regression Modeling Strategies*. Springer International Publishing, 2001.
16. W. Haynes. *Bonferroni Correction*, pages 154–154. Springer New York, 2013.
17. J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 2003.
18. Z. Jia, S. Li, X. Liu, X. Liao, and Y. Liu. Smartlog: Place error log statement by deep understanding of log intention. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 61–71, 2018.
19. J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan. The impact of correlated metrics on defect models. *arXiv preprint arXiv:1801.10271*, 2018.
20. J.Lu, A.Liu, F.Dong, F.Gu, J.Gama, and G.Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.
21. S. Kabinna, W. Shang, C. Bezemer, and A. E. Hassan. Examining the stabity of logging statements. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 326–337, 2016.
22. S. Kaufman, S. Rosset, and C. Perlich. Leakage in data mining: Formulation, detection, and avoidance. volume 6, pages 556–563, 01 2011.
23. D. Lee, G. K. Rajbahadur, D. Lin, M. Sayagh, C.-P. Bezemer, and A. E. Hassan. An empirical study of the characteristics of popular minecraft mods. *Empirical Software Engineering*, pages 1–23, 2019.
24. D. Lee, G. K. Rajbahadur, D. Lin, M. Sayagh, C.-P. Bezemer, and A. E. Hassan. An empirical study of the characteristics of popular minecraft mods. *Empirical Software Engineering*, 09 2020.
25. H. Li, T.-H. P. Chen, W. Shang, and A. E. Hassan. Studying software logging using topic models. *Empirical Software Enggineering*,

23(5):2655–2694, 2018.
26. H. Li, T.-H. P. Chen, W. Shang, and A. E. Hassan. Studying software logging using topic models. *Empirical Softw. Engg.*, 23(5):2655–2694, Oct. 2018.
27. H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan. A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.
28. H. Li, W. Shang, and A. Hassan. Which log level should developers choose for a new logging statement? *Empirical Software Engineering*, page 1684–1716, 2017.
29. Z. Li, H. Li, T.-H. Chen, and W. Shang. Deeplv: Suggesting log levels using ordinal based neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1461–1472, 2021.
30. Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao, M. Chintalapati, and D. Zhang. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 480–490, New York, NY, USA, 2018. Association for Computing Machinery.
31. A. Liu, Y. Song, G. Zhang, and J. Lu. Regional concept drift detection and density synchronized drift adaptation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17.
32. Z. Liu, X. Xia, D. Lo, Z. Xing, A. E. Hassan, and S. Li. Which variables should i log? *IEEE Transactions on Software Engineering*, pages 1–1, 2019.
33. N. Lu, J. Lu, G. Zhang, and R. Lopez de Mantaras. A concept drift-tolerant case-base editing technique. *Artificial Intelligence*, 230(C):108–133, 2016.
34. Y. Lyu, H. Li, M. Sayagh, Z. Jiang, and A. E. Hassan. An empirical study of the impact of data splitting decisions on the performance of aiops solutions. *ACM Transactions on Software Engineering and Methodology*, 01 2021.
35. Y. Lyu, G. K. Rajbahadur, D. Lin, B. Chen, and Z. M. J. Jiang. Towards a consistent interpretation of aiops models. 31(1), 2021.
36. N. M. Olewicki, Doriane and B. Adams. Towards language-independent brown build detection. In *Proc. of the 44th Int. Conf. on Software Engineering(ICSE'22)*, 2022.
37. A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Communications of the ACM*, page 55–61, 2012.
38. Y. E. Ouatiti. The impact of concept drift and data leakage on log level prediction models - appendix. https://zenodo.org/records/10898284, 2024.
39. Y. E. Ouatiti, M. Sayagh, N. Kerzazi, and A. E. Hassan. An empirical study on log level prediction for multi-component systems. *IEEE Transactions on Software Engineering*, 2023.

40. A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo. Industry practices and event logging: Assessment of a critical software development process. In *Proceedings of the 37th International Conference on Software Engineering*, volume 2, pages 169–178, 2015.
41. G. K. Rajbahadur, S. Wang, G. Ansaldi, Y. Kamei, and A. E. Hassan. The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering*, pages 1–1, 2021.
42. S. Ramrez-Gallego, B. Krawczyk, S. Garca, M. Woniak, and F. Herrera. A survey on data preprocessing for data stream mining. *Neurocomput.*, 239(C):39–57, 2017.
43. B. A. S. McIntosh, Y. Kamei and A. E. Hassan. The impact of code review coverage and code review participation on software quality. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, page 292–201, 2014.
44. F. Sarro, R. Moussa, A. Petrozziello, and M. Harman. Learning from mistakes: Machine learning enhanced human expert effort estimates. *IEEE Transactions on Software Engineering*, 2022.
45. M. Sayagh, Z. Dong, A. Andrzejak, and B. Adams. Does the choice of configuration framework matter for developers? empirical study on 11 java configuration frameworks. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2017.
46. W. Shang, M. Nagappan, and A. E. Hassan. Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Enggineering*, 20(1):1–27, 2015.
47. E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan. Is lines of code a good measure of effort in effort-aware models? *Information and Software Technology*, 2013.
48. R. Subramanyam and M. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects. *IEEE Transactions on Software Engineering*, 2003.
49. C. Tantithamthavorn and A. E. Hassan. An experience report on defect modelling in practice: Pitfalls and challenges. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP'18)*, page To Appear, 2018.
50. C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, 2018.
51. C. Tantithamthavorn, J. Jiarpakdee, and J. Grundy. Actionable analytics: Stop telling me what it is; please tell me what to do. *IEEE Software*, 2021.
52. C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 45(07):683–711, 2019.
53. P. Thongtanunam and A. Hassan. Review dynamics and its impact on software quality. *IEEE Transactions on Software Engineering*, pages 1–13, 2018.

54. D. Yuan, Y. D., Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *Proceedings of the 11th Conference on Operating Systems Design and Implementation, Systems Design and Implementation*, pages 249–265, 2014.
55. D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12.
56. D. Yuan, S. Park, and Y. Zhou. Characterizing logging practices in open-source software. In *Proceedings of the 34th International Conference on Software Engineering(ICSE'12)*, pages 102–112, 2012.
57. D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage. Improving software diagnosability via log enhancement. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, page 3–14. Association for Computing Machinery, 2011.
58. D. Zhang and J. Tsai. Machine learning and software engineering. 2002.
59. J. Zhang, V. S. Sheng, J. Wu, and X. Wu. Multi-class ground truth inference in crowdsourcing with clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2016.
60. Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman amp; Hall/CRC, 2012.
61. J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang. Learning to log: Helping developers make informed logging decisions. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE'15, pages 415–425, 2015.