LogBERT: A BERT-Driven Approach to Log Instruction Quality Assessment

*

Sam Singh
Dept. of Software Engineering
Rochester Institute of Technology
Rochester, US
ss8650@rit.edu

Sushant Borse
Dept. of Software Engineering
Rochester Institute of Technology
Rochester, US
sb8195@rit.edu

Aayush Panchal
Dept. of Software Engineering
Rochester Institute of Technology
Rochester, US
ap9115@rit.edu

I. Introduction

A log is a string that provides contextual information about a process during its runtime. It plays an important role in various software maintenance tasks, such as failure diagnosis, auditing, and profiling. Typically, a log message contains three elements: 1) static text describing an event, 2) variable text providing dynamic information about the event, and 3) a log level (e.g., info, error, warning) representing the developer's assessment of the event's severity. Logs are useful in bridging the gap between code developers and system operators, who use log messages to monitor processes and troubleshoot errors without interacting with the underlying code. Poorly written log instructions can lead to confusion, hinder efficient troubleshooting, and increase maintenance costs.

Fig. 1. Log Messages

Therefore, having high-quality logs is essential. However, this is not an easy task. There is a lack of comprehensive and consistent guidelines for developers on logging best practices. Chen et al.[1] show that developers make frequent log-related commits. In many cases, log messages are written as "afterthoughts" after a failure occurs.

Yuan et al. [2] quantify the log pervasiveness and the benefits of software logging in C/C++ systems while proposing proactive logging strategies. It also aims to determine the extent of logging, rate the effectiveness of log messages, and comprehend how developers alter logs. However, due to the study's focus on just four sizable open-source projects, not all software logging practices may be properly represented, and moreover, the analysis ignores the addition of completely new logging code and instead focuses on changes to the existing logging code. The conclusions also may not apply to software or languages other than open-source server applications written in C/C++. Similar findings are reported for the Java [1, 3, 4, 6, 7] and Android software platforms. Another paper [4] by Kabina et al. evaluates the stability of logging statements for four different open-source programs, but at the same time, it ignores the changes and external factors with respect to logging configurations. The methodology used might also not directly apply to the proprietary software without access to the source code history.

Various logging libraries, like syslog and log4j, offer enhanced logging interfaces compared to general-purpose output functions. However, they are language-dependent and still leave decisions regarding what and where to log to developers. To address these decision-making challenges, Ding et al. [5] studied developers' logging considerations and proposed investigating automated logging techniques. Log Enhancer [9], another notable tool, automatically suggests variable values to record in existing log messages with a notable accuracy of 95.1%. Yet, its capability to capture domain-specific or context-specific details may be limited.

Li et al.[10] introduced a deep-learning-based automated approach to determine logging locations, achieving an 80.1% accuracy, and in subsequent work [11], proposed a method for suggesting appropriate log levels with an 83.7% accuracy. Despite these advancements, achieving high-quality log messages requires a combined approach that considers both correct log levels and context.

Our current research builds upon the foundational work of QuLog by Bogatinovski et al. [8], which aimed to assess log quality using neural networks. Our approach introduces the LogBERT methodology, which advances QuLog by combining its core principles and leveraging pre-trained models. LogBERT is specifically designed to evaluate two critical dimensions of log instructions: the accuracy of log-level assignment and the comprehensiveness of the static text in conveying the event's details. We achieve this by training neural networks on diverse data, encompassing a wide range of developer writing styles and logging practices. This ensures that LogBERT is well-versed in the intricacies of different logging approaches and can provide nuanced recommendations.

In a nutshell, the transition from QuLog to LogBERT in

our research represents a significant step forward in automated log analysis. By focusing on both the technical and linguistic aspects of logs, LogBERT promises to enhance the quality of logging practices in software development, thereby improving the overall efficiency and effectiveness of software maintenance tasks.

II. Study Design

A. Preliminary Empirical Study

The empirical study in QuLog focuses on assessing log instruction quality properties, specifically examining log-level assignment and linguistic quality assessment.

For log level assignment, the study observes that static text often contains words that hint at the log level's appropriateness. They use information theory to measure the uncertainty of words' association with log levels and found that most static texts have low entropy, indicating that they can effectively discriminate log levels.

In the case of linguistic quality assessment, the study emphasizes that good log instructions should provide clear and concise event descriptions. To validate this, they perform an experiment where they extract linguistic structures from static text using part-of-speech tagging. They group similar linguistic structures together and have experienced developers assess whether these linguistic groups contain enough information to comprehend the event description.

The study concludes that log quality assessment, in terms of log-level assignment and linguistic quality, can be done automatically based on the content of log instructions. Other quality properties like variable selection, log instruction placement, safeness, and completeness depend on factors like programming languages and source code structure, making them more challenging to assess automatically. Therefore, the study narrows its focus to log-level assignment and linguistic quality assessment.

B. Approach Overview

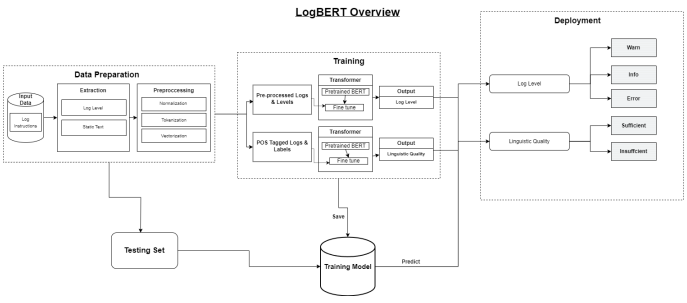

Fig. 2. Approach Overview

In our approach, we build upon the initial findings from the preliminary study to develop a system-agnostic method for assessing log instruction quality. Our focus is two-fold: first, evaluating the accuracy of log-level assignment, and second, assessing the linguistic adequacy of the log messages. Utilizing pre-trained BERT models, we fine-tune them to learn specific textual characteristics relevant to log levels and linguistic structures. This process is conducted on a dataset comprising systems known for their quality logging practices. Our framework, named LogBERT, has two main components: (1) data preparation, which extracts and formats log instructions for deep learning analysis, and (2) a deep learning framework consisting of two neural networks, each trained on one of the two assessment tasks. Through this training, the networks can distinguish log instructions across different log levels and identify those with sufficient linguistic structure. This method allows us to compare the model's predictions with developer-assigned log levels and linguistic quality, enabling an objective assessment of log instruction quality.

1) Data Preparation: In the data preparation module, log instructions are processed in a suitable learning format. The process involves separating the static text and log level from each log instruction. The static text is preprocessed using Spacy, which includes splitting words, removing ASCII special characters, eliminating stopwords from the Spacy English dictionary, and converting words to lowercase. Additionally, a pre-trained POS tagging model from Spacy is employed to extract the part-of-speech tags of each word, thereby creating the linguistic structure of the static text. The final output of this module is a set of tuples, each containing the static text of the instruction, its linguistic structure, and the log level.

2) Training: LogBERT adapts the QuLog architecture with a key enhancement: the integration of pre-trained BERT models, fine-tuned for specific log analysis tasks. It maintains two distinct neural networks, each dedicated to assessing one of the two core quality properties: log-level assignment and linguistic structure of log instructions. The architecture of each network comprises:

- Embedding Layer: In this layer, LogBERT distinguishes itself from QuLog by utilizing BERT's pre-trained embeddings. These embeddings are adapted to log instructions, providing a rich, context-aware representation of the static text or linguistic structures. This adaptation enhances the model's ability to comprehend and process the subtleties in log messages.
- Transformer Neural Network: Central to this layer is the Transformer encoder. It uses BERT's bidirectional training to analyze complex relationships within log messages, crucial for understanding context. The multi-head self-attention mechanism is adept at handling various dependencies and nuances in the text, making it highly effective for detailed log analysis.
- Output Layer: This layer interprets the embeddings from the Transformer encoder to make final predictions. The output dimensions correspond to the log levels (info, warning, error) or linguistic structure qualities (sufficient, insufficient), with a softmax function providing class score estimates.

3) Deployment: In the deployment phase, the trained models are saved and used to determine the quality of a given log message. While classifying log levels, it differentiates between

'info', 'error', and 'warning'. It also defines the linguistic quality as 'sufficient' or 'insufficient'.

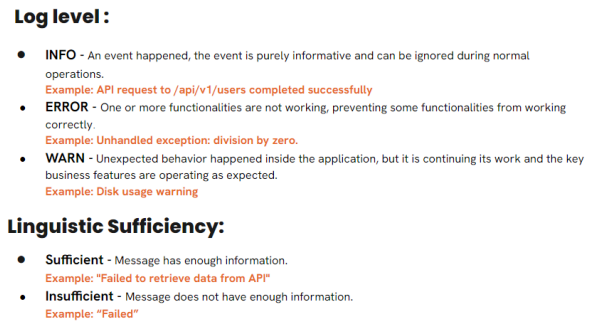

Fig. 3. Log Message Quality

III. Experiment and Result

We created an experiment to evaluate the performance of our suggested LogBERT technique with the previously published QuLog model to obtain a thorough evaluation of it.

A. Experiment Design

We collected data from the QuLog provided dataset, which included two subsets:

1) The first dataset contained static text and log levels extracted from various popular GitHub repositories. This subset aided in assessing the automated approach in determining the appropriate log level.
2) The second dataset comprised static text with labels of 'sufficient' or 'insufficient,' manually annotated by the QuLog authors, aimed at evaluating the linguistic sufficiency of log messages.

Using these datasets, we sought to answer the following research questions:

- RQ1: To what extent can an automated approach accurately determine the appropriate log level in log instruction quality assessment?
- RQ2: What is the effectiveness of the proposed automated methodology in accurately determining the linguistic sufficiency of log messages?
- RQ3: How does the efficacy of the proposed approach in log instruction quality assessment compare to that of the existing methodologies?

B. Results

1) Log Level Assessment: Our analysis for RQ1 revealed that LogBERT outperformed the QuLog model in determining the appropriate log level with a 23% increase in accuracy. When looking at the average Area Under the Curve (AUC) scores across different projects, LogBERT achieved a score of 0.94, with its binary classification version (info-error), LogBERT2, achieving a score of 0.98. This demonstrates significant improvements over QuLog's average score of 0.93. LogBERT2 also showed an impressive performance in terms of accuracy, scoring an average of 0.96, which is a notable 34% improvement over QuLog's average accuracy of 0.62.

	AUC			Accuracy		
Project	**LogBERT**	**LogBERT2**	**Qu-Log8**	**LogBERT**	**LogBERT2**	**Qu-Log8**
Hbase	0.95	0.99	0.94	0.85	0.96	0.63
Jmeter	0.93	0.98	0.93	0.8	0.94	0.59
wicket	0.92	0.95	0.94	0.85	0.92	0.62
cassandra	0.93	0.98	0.91	0.79	0.93	0.59
karaf	0.95	0.99	0.92	0.86	0.95	0.59
flink	0.96	0.99	0.93	0.87	0.94	0.58
kafka	0.95	0.98	0.93	0.83	0.94	0.63
Zookeeper	0.93	0.98	0.94	0.81	0.95	0.75
elasticsearcl	0.93	0.96	0.92	0.83	0.9	0.59
Average	**0.94**	**0.98**	**0.93**	**0.85**	**0.96**	**0.62**

Fig. 4. Log Level Results

2) Linguistic Structure Analysis: Addressing RQ2, our approach utilized two strategies: one employing part of speech (POS) tags as features and the other using static text. The F1 score, a measure of a test's accuracy, was perfect (1.0) when using POS features, indicating exceptional precision and recall balance. The specificity metric reached the maximum score (1.0), suggesting flawless identification of true negatives. When using static text features, the F1 score remained high at 0.94, with specificity slightly reduced to 0.94, signifying a strong ability to discern the linguistic quality of log messages.

Metrics	Using POS features	Using static text features	QuLog
F1	1	0.94	0.98
Specificity	1	0.94	0.99

Fig. 5. Linguistic Structure Results

3) Comparative Assessment: In response to RQ3, the efficacy of LogBERT's approach in assessing log instruction quality was robust when juxtaposed with the QuLog methodology. The LogBERT model demonstrated substantial advancements across all projects compared to QuLog, as illustrated by the AUC and accuracy metrics. Particularly for the Kafka and Zookeeper projects, LogBERT's accuracy exceeded QuLog by over 20 percentage points, indicating a superior capability in handling diverse software systems and their logging practices.

IV. Analysis

After conducting a thorough experiment, we were able to gather valuable insights on how LogBERT and QuLog compare in terms of performance. In this detailed analysis, we will highlight the key observations we made while reviewing the results.

A. Linguistic Structure Analysis

The results showed that the LogBERT model performed significantly better for linguistic structure evaluation. The analysis

makes the following important observations and suggestions for improving the methodology:

- Dataset Mislabeling: To determine if inaccurate labeling was contributing to the model's poor performance, we manually validated a subset of the data where the model predictions differed from the provided labels. Two new annotators independently classified a 10% sample of these messages as sufficient or insufficient. The annotators also tagged cases they found difficult to categorize definitively. As shown in Fig. 6, the annotators often agreed more with the model's predictions than the original dataset labels. Many cases were flagged as hard to categorize definitively, indicating ambiguity in the labeling. Lastly, notable disagreement existed between the two annotators on many items. This mislabeling could lead to an underestimation of the model's actual performance.

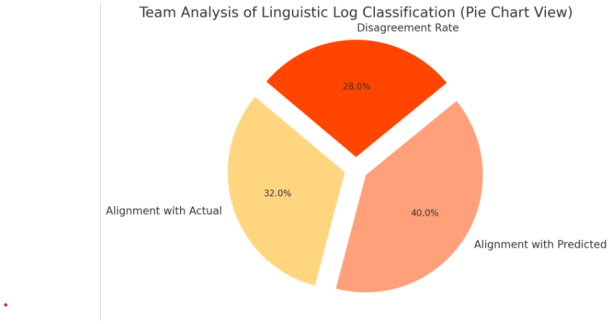

Fig. 6. Linguistic Structure Results

- Character Length Consideration: We observed that the character length of the static text is often directly proportional to its sufficiency. Longer messages tended to be more informative:
  - Sufficient Example: "Failed to retrieve data from API."
  - Insufficient Example: "Failed."
- Inclusion of Dynamic Variables: Dynamic variables could substantially enrich the analysis. Variable names, which are frequently descriptive in natural language, can provide critical context: An example where inclusion could be beneficial: "Waiting for [variablename]"
- Case Sensitivity: The preprocessing step of lowercasing all data presented challenges. Case sensitivity can be vital for BERT to discern nuances in meaning and context: An example of a problematic case: "stopping defaultleaderretrievalservice."

B. Log Level Analysis

In evaluating log levels, the study's findings were two-fold:

- Dataset Mislabeling: As with linguistic classification, we performed similar annotation experiments to determine if the dataset was misclassified. We noted a higher agreement rate with the predictions of LogBERT than with the actual labels from the dataset. This suggests that mislabeling could be an issue affecting the model's reported accuracy.

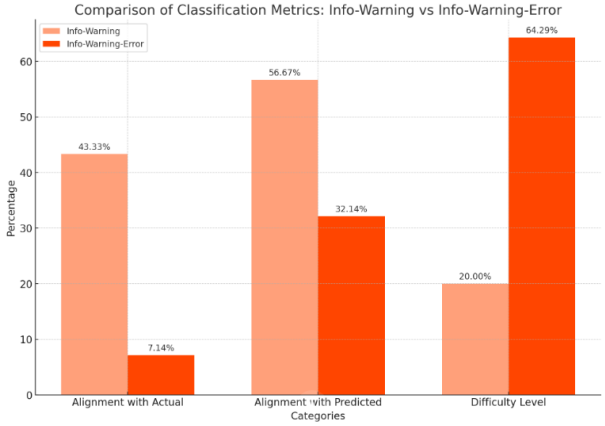

Fig. 7. Log Level Results

- Complexity of Log Level Classification: The classification of log messages into discrete levels – info, warning, and error – proved complex. We found that sometimes multiple log levels could apply to a single log, and human analysts also faced difficulties in distinguishing between these categories. However, the difficulty in classification dropped significantly when faced with binary classification, that is, info and error.

V. Related Work

A. Logging practices

In the field of software development, it is essential to comprehend and improve logging procedures. This is demonstrated by numerous studies concentrating on various software systems and programming languages, each offering a distinct perspective on the subtleties of efficient logging.

In the area of C/C++ systems, Yuan et al.'s noteworthy study [2] emphasized the comprehensive nature of program logging. The authors underscored the need of proactive logging tactics, pointing out that although developers frequently modify log levels, static texts, and log settings, they hardly ever change the location of log instructions. This result points to a degree of rigidity in the spatial logging component, which our study aims to investigate further by assessing the effect of log instruction placement on the overall program quality.

In the study [27], Minaee et al. provide a comprehensive review of more than 150 deep learning-based text classification models developed in recent years, and they discuss their technical contributions, similarities, and strengths in their paper. BERT was discovered to be one of the best text classification models. They also stated that using pre-trained models can significantly improve the accuracy of text classification models. We decided to use a pre-trained model and fine-tune it for log assessment using this.

Studies concentrating on Android [13] and Java [1, 3, 4, 6, 7] systems have made parallel observations. By adding new dimensions including log bug resolution time [1], log

instruction update kinds [3], and the evolution of logging settings [7], these studies broadened the scope of logging research. The findings of Kabina et al. [4], which showed that 20–45% of log instructions are prone to changes over a system's lifespan, are particularly noteworthy. The fact that log instructions are dynamic is consistent with our study goal of evaluating and improving the stability and pertinence of log messages over time.

Numerous studies [14, 15, 16] analyzing the logging practices of industrial systems demonstrate the significance of logging practices beyond the purview of academic study to the industry. Li et al.'s [5] comparison of logging expenses from the perspectives of researchers and developers offered a distinctive viewpoint. These observations highlight the real-world difficulties in logging, which our study attempts to solve by putting forth strategies for the automated evaluation and enhancement of log instruction quality.

The findings are consistent across studies and programming languages, indicating a common feature of logging techniques and emphasizing the necessity of using a cross-examination method to assess the quality of log training. This strategy is very pertinent to our work since it serves as the foundation for our suggested approaches, including the QuLog framework, which aims to improve logging quality automatically using deep learning techniques.

B. Automatic Logging Enhancement

Numerous cutting-edge strategies have been implemented in the field of automatic logging enhancement with the goal of raising the caliber and efficacy of log instructions. These techniques can be roughly divided into two groups: those that deal with where to log instructions and those that concentrate on choosing pertinent logging data (what-to-log).

Where-to-log Strategies: A number of research works [15, 17, 18, 19] have examined how best to arrange log instructions within the code. This part of the logging process is very important since it establishes the relevancy and context of the log messages. These tactics are consistent with our findings, especially when considered in light of our empirical study's focus on evaluating the placement and content of log instructions.

Which Approaches to Log: Conversely, a substantial amount of research [20, 21, 22, 23] has focused on determining what data ought to be recorded. Three subgroups can be further separated out of this category: log level suggestion [24, 12], relevant variables placement [25], and log message generation [26]. Automatic system-agnostic log instruction quality assessment is the primary goal of our study, and each of these subcategories helps to improve the informativeness and clarity of log messages.

C. Relation to Our Study:

Unlike other research, our study develops an automatic method for evaluating the quality of log instruction by taking advantage of shared linguistic qualities throughout various software systems. We highlight two important points: 1) Log level assignment: we examine how well log levels are selected to represent the seriousness of the incident; 2) Sufficient language structure assessment: we consider if the log messages adequately and clearly describe the events.

This method of improving automatic logging is essential since it goes beyond the traditional techniques of just adding and formatting log instructions. Our study intends to build a more comprehensive and flexible framework for log instruction quality assessment by incorporating lessons from these current approaches and customizing them to the specific requirements of various software systems.

VI. Threat To Validity

A. Bias from Assumed Quality of Training Data

One possible source of bias is the absence of an agreed-upon format for log statement writing. The project is predicated on the idea that the training data follows best practices for logging and comprises high-quality source code. This may not always be the case because different projects can have logging that varies greatly in quality. A model that is biased towards specific styles or practices that are not widely recognized as "high-quality" may result from training data that is skewed in that direction, which could limit the model's usefulness and applicability.

B. Potential for Annotation-Related Artifacts

While the presence of two annotators is meant to reduce bias, the possibility of annotation-related artifacts is still there. Annotators' annotations may be influenced by their own innate prejudices or preferences in how they perceive log statements. Although adding a layer of cross-validation with two annotators lowers this risk, it doesn't completely eliminate the chance that subjective factors could alter the annotations.

C. Applicability to Different Programming Languages

By choosing systems with varying sizes, logging statement counts, and domains, the study aims to make its conclusions more broadly applicable. Still, it's possible that the findings won't apply directly to systems built in languages that weren't studied. Findings from one set of programming languages may not transfer well to another since different programming languages may have different logging systems, norms, and practices. This constraint may have an impact on how well the suggested approaches work in various programming environments and how broadly the study's conclusions may be applied.

VII. Future Work

Our results point to several directions for future investigation and improvement of automated log assessment techniques.

A. Enhanced Feature Extraction

Subsequent versions of our model will investigate incorporating further characteristics to enhance the system's capacity for prediction. In particular, incorporating dynamic variables and character length in the log messages will be examined.

Character length has demonstrated promise as a linguistic sufficiency indicator, and dynamic variables may offer a more comprehensive framework for the model's study.

B. Addressing Dataset Quality

It will be necessary to examine datasets with mislabeled data and explore ways to modify them in order to improve the accuracy of the labels. This will enable more robust models to be trained and ensure a more reliable evaluation of the model's performance.

C. Case Sensitivity and Contextual Awareness

Our work demonstrated the significance of case sensitivity in text analysis. Future iterations of the model will aim to maintain the text's original case in order to fully utilize BERT's capacity to comprehend and interpret the meaning of the case in the language. To give the model a more complete picture of the log message environment, we also intend to include more extensive contextual data, including neighboring log lines.

REFERENCES

[1] Boyuan Chen and Zhen Ming (Jack) Jiang. 2017. Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation. Empirical Software Engineering 22 (2017), 330–374.

[2] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. 2012. Characterizing Logging Practices in Open-Source Software. In Proceedings of the 34th International Conference on Software Engineering. IEEE Press, River Street, USA, 102–112.

[3] Boyuan Chen and Zhen Ming Jiang. 2019. Extracting and Studying the LoggingCode-Issue- Introducing Changes in Java-Based Large-Scale Open Source Software Systems. Empirical Softw. Engg. 24 (2019), 2285–2322.

[4] Suhas Kabinna, Cor-Paul Beze[28] Heng Li,Weiyi Shang, Brayan Adams, Mohammed Sayagh, and Ahmed E. Hassan. 2020. A Qualitative Study of the Benefits and Costs of Logging from Developers'

[5] Heng Li,Weiyi Shang, Brayan Adams, Mohammed Sayagh, and Ahmed E. Hassan. 2020. A Qualitative Study of the Benefits and Costs of Logging from Developers' mer, Weiyi Shang, Mark D. Syer, and Ahmed E.

[6] Weiyi Shang, Zhen Ming Jiang, Bram Adams, Ahmed E. Hassan, Michael W. Godfrey, Mohamed Nasser, and Parminder Flora. 2014. An exploratory study of the evolution of communicated information about the execution of large software systems. Journal of Software: Evolution and Process 26 (2014), 3–26.

[7] Chen Zhi, Jianwei Yin, Shuiguang Deng, Maoxin Ye, Min Fu, and Tao Xie. 2019. An Exploratory Study of Logging Configuration Practice in Java. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE Press, New York, USA, 459–469. https://doi.org/10.1109/ICSME.2019.00079

[8] J. Bogatinovski, S. Nedelkoski, A. Acker, J. Cardoso and O. Kao, "QuLog: Data-Driven Approach for Log Instruction Quality Assessment," 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), Pittsburgh, PA, USA, 2022, pp. 275-286, doi: 10.1145/3524610.3527906.

[9] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2012. Improving Software Diag

[10] Zhenhao Li, Tse-Hsun (Peter) Chen, and Weiyi Shang. 2020. Where Shall We Log? Studying and Suggesting Logging Locations in Code Blocks. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. Association for Computing Machinery, New York, NY, USA, 361–372.

[11] Zhenhao Li, Tse-Hsun (Peter) Chen, and Weiyi Shang. 2020. Where Shall We Log? Studying and Suggesting Logging Locations in Code Blocks. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. Association for Computing Machinery, New York, NY, USA, 361–372.

[12] Zhenhao Li, Heng Li, Tse-Hsun Chen, and Weiyi Shang. 2021. DeepLV: Suggesting Log Levels Using Ordinal Based Neural Networks. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE Press, NJ, USA, 1461–1472. https://doi.org/10.1109/ICSE43902.2021.00131.

[13]] Yi Zeng, Jinfu Chen, Weiyi Shang, and Tse-Hsun (Peter) Chen. 2019. Studying the characteristics of logging practices in mobile apps: a case study on F-Droid. Empirical Software Engineering 24 (2019), 3394–3434

[14] Titus Barik, Robert DeLine, Steven Drucker, and Danyel Fisher. 2016. The Bones of the System: A Case Study of Logging and Telemetry at Microsoft. In 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). Association for Computing Machinery, New York, NY, USA, 92–101

[15] Jeanderson Cândido, Haesen Jan, Maurício Aniche, and Arie van Deursen. 2021. An Exploratory Study of Log Placement Recommendation in an Enterprise System. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE Computer Society, Los Alamitos, CA, USA, 143–154. https://doi.org/10.1109/MSR52588.2021.00027

[16] Antonio Pecchia, Marcello Cinque, Gabriella Carrozza, and Domenico Cotroneo. 2015. Industry Practices and Event Logging: Assessment of a Critical Software Development Process. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 2. IEEE Press, New York, USA, 169–178

[17] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. 2013. Event Logs for the Analysis of Software Failures: A Rule-Based Approach. IEEE Transactions on Software Engineering 39 (2013), 806–821.

[18] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. 2014. Where Do Developers Log? An Empirical Study on Logging Practices in Industry. In Companion Proceedings of the 36th International Conference on Software Engineering. Association for Computing Machinery, New York, NY, USA, 24–33.

[19] Heng Li, Tse-Hsun (Peter) Chen, Weiyi Shang, and Ahmed E. Hassan. 2018. Studying Software Logging Using Topic Models. Empirical Softw. Engg. 23 (2018), 2655–2694.

[20] Jian Li, Yue Wang, Michael R. Lyu, and Irwin King. 2018. Code Completion with Neural Attention and Pointer Networks. In Proceedings of the 27th International Joint Conference on Artificial Intelligence. AAAI Press, 4159–25.

[21] Zhenhao Li, Heng Li, Tse-Hsun Chen, and Weiyi Shang. 2021. DeepLV: Suggesting Log Levels Using Ordinal Based Neural Networks. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE Press, NJ, USA, 1461–1472. https://doi.org/10.1109/ICSE43902.2021.00131.

[22] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be Conservative: Enhancing Failure Diagnosis with Proactive Logging. In 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). USENIX Association, Hollywood, CA, 293–306.

[23] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2012. Improving Software Diagnosability via Log Enhancement. ACM Trans. Comput. Syst. 30, 1 (2012), 3–14.

[24] Anu Han, Chen Jie, Shi Wenchang, Hou Jianwei, Liang Bin, and Qin Bo. 2019. An Approach to Recommendation of Verbosity Log Levels Based on Logging Intention. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, New York, USA, 125–134.

[25] Xu Zhao, Kirk Rodrigues, Yu Luo, Michael Stumm, Ding Yuan, and Yuanyuan Zhou. 2017. Log20: Fully Automated Optimal Placement of Log Printing Statements under Specified Overhead Threshold. In Proceedings of the 26th Symposium on Operating Systems Principles. Association for Computing Machinery, NewYork

[26] Pinjia He, Zhuangbin Chen, Shilin He, and Michael R. Lyu. 2018. Characterizing the Natural Language Descriptions in Software Logging Statements. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Association for Computing Machinery, New York, NY, USA, 178–189

[27] Deep Learning–based Text Classification: A Comprehensive Review.Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jian fengGao.2021. Deep Learning–based Text Classification : A Comprehensive Review. ACM Comput. Surv. 54, 3, Article 62 (April 2021), 40 pages. https://doi.org/10.1145/3439726.