2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) | 979-8-3503-0039-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-NIER58687.2023.00023

Auto-Logging: AI-centred Logging Instrumentation

Jasmin Bogatinovski
Technical University Berlin, Berlin, Germany
jasmin.bogatinovski@tu-berlin.de

Odej Kao
Technical University Berlin, Berlin, Germany
odej.kao@tu-berlin.de

***Abstract*—Logging in software development plays a crucial role in bug-fixing, maintaining the code and operating the application. Logs are *hints* created by human software developers that aim to help human developers and operators in identifying root causes for application bugs or other misbehaviour types. They also serve as a bridge between the Devs and the Ops, allowing the exchange of information. The rise of the DevOps paradigm with the CI/CD pipelines led to a significantly higher number of deployments per month and consequently increased the logging requirements. In response, AI-enabled methods for IT operation (AIOps) are introduced to automate the testing and run-time fault tolerance to a certain extent. However, using logs tailored for human understanding to learn (automatic) AI methods poses an ill-defined problem: AI algorithms need no hints but structured, precise and indicative data. Until now, AIOps researchers adapt the AI algorithms to the properties of the existing human-centred data (e.g., log sentiment), which are not always trivial to model. By pointing out the discrepancy, we envision that there exists an alternative approach: the logging can be adapted such that the produced logs are better tailored towards the strengths of the AI-enabled methods. In response, in this vision paper, we introduce auto-logging, which devises the idea of how to automatically insert log instructions into the code that can better suit AI-enabled methods as end-log consumers.**

***Index Terms*—software engineering, logging, AIOps**

I. Introduction

Traditionally, in IT systems, developers write code while system operators run the code mostly as a black box. The connection between the two worlds is frequently established with log messages (logs). The developer hints to the (unknown) operator for the root cause of an issue, and vice versa the operator reports bugs during operation. To enable logging, developers instrument the source code with log instructions. Log instructions are structured texts, commonly composed of a logger object, log level (e.g., info", "error"), static text (IP {} cannot be reached), and dynamic variables (e.g. IP {}). The flexibility of inserting the log instructions within any part of the source code enables the recording of important events, making logs a common choice for tasks such as troubleshooting, debugging, system comprehension [1], and tasks with legal implications [2]. Therefore, logging is important for successful system development and operation.

The central dogma in software logging is that instructions are written from *humans* for *humans*, i.e., the logging is *human-centric*. The *human-centric* logging has two key assumptions. The first assumption is that the *expected log consumers* are humans. Therefore, the log messages are represented in a *human*-friendly form. The second assumption suggests that developers insert log instructions based on *sufficient understanding* of the system and production environment complexities and the running time configurations. Although prior knowledge of all the system and running time complexities is seldom available, the human-centric view is the gold standard for logging instrumentation.

As envisioned by the human-centric view, the operational activities with the produced logs are to be performed by humans. However, in modern IT systems, there is a strong development and adoption of log-based methods for automating operational activities [3], [4]. Automatic methods are intelligent methods (e.g., machine learning methods) that use the logs from the existing log instrumentation to learn models for operational tasks (e.g., anomaly detection). We point out that by changing the log consumer of the produced logs, the first assumption of the human-centric logging view is violated.

Furthermore, automatic methods[1] are learned on data initially meant to be understood by humans. Accordingly, they are tailored to accommodate human-friendly properties that are not always trivial to model (e.g., log sentiment [5]). While designed to replace humans in system operation, automatic methods intrinsically share properties different from humans. For example, some automatic methods (e.g., LSTM [6]) can easily learn subtle differences in long log sequences (up to several hundred characters), as opposed to human operators. For humans, it is usually harder to track conditional sequence changes of several tens of characters (e.g., caused by branching conditions). In addition, automatic methods can work on arbitrary data types, while human operators predominantly need a textual description of the events.

As the adoption of automatic methods is expected to grow [4], inevitably, the majority of the log consumers of system logs will be entities other than humans. Recognizing that there exist different properties between the two means that some properties may be more favourable for the automatic methods, while others for human operators, potentially imposing shortcomings on the other. Eminently the question of what is the most optimal input for the automatic methods as consuming *entities* emerges. While in the human-centric view, the human directly limits the form of logging content (e.g., its textual nature [7], [8]), it is unclear if there exists logging instrumentation better suited for the properties of the automatic methods. Furthermore, even in the human-centre logging, there are many documented cases from the area of "how-to-log" [9],

[1]We use the term *automatic* for AI-enabled log-based operation methods. We use *auto-logging* for autonomous logging instrumentation references.

showing that developers make inappropriate judgments on log locations [8], or log instruction content [7]. The studies point out that if the log instructions were placed correctly or augmented sufficiently, they would have made the logs more useful in problem resolution [10], [11]. Thereby, developers do not have *sufficient understanding* of all complexities, which violates the second assumption [12]–[15]. This becomes even more weight considering the pace of system software and hardware evolution, the changing execution models, and storage paradigms to name just a few.

Motivated by the eminent violation of the two central assumptions of the human-centric logging instrumentation, in this paper, we introduce the idea of **auto-logging**. Auto-logging aims to accommodate the existing discrepancy between human-centric instrumentation and its utilization for automatic operation methods. As a concept, it refers to automatically instrumenting the source code with log instructions such that the automatic properties are better emphasized. The logging automation enables log adoption irrespective of the developer's experience level. It also liberates the developers from taking care of logging instrumentation while simultaneously it is optimized for the automatic methods. By setting the focus on automatically inserting logging instructions in the context of improving the performance of the automatic methods, we envision that auto-logging may improve the overall quality of the logging code and system operation. In the remaining, we introduce the auto-logging idea and envision possible approaches on how to achieve it.

II. Software Logging Process Analysis

A. The Current State of Logging

Currently, the logging instrumentation is performed by developers. They follow a sequential three-stage choice procedure for the 1) logging approach, 2) logging utility, and 3) logging code composition [15]. The key assumptions are that the developers have a complete understanding of the system complexities and the purpose of logging. In addition, the logs are expected to be of sufficient quality as they are omnipresent over many operational tasks [16]. To impose quality on the logging process, many companies specify logging guidelines [14].

However, studies on logging practices point out that the practical state of logging instrumentation undermines the assumption that developers always have *sufficient understanding* for the logging purpose [9], [13]–[15], [17]. To begin with, modern software systems are commonly developed by many developers with different levels of expertise and overview of the system complexity, often collaborating remotely. The misalignment in expertise and lack of shared understanding leads to misinterpretation of the logging purpose. For example, Chen et al. [9] imply this by finding that the source files modified by developers other than the original contributor are more prone to introduce ambiguities for troubleshooting than non-modified files. Next, the logging information may be incorrect or insufficient. For example, Salfner et al. [18] reported that the misinterpretation of the characters `be` as a verb instead of the hexadecimal value for the number 190, required several hours to diagnose the problem. Many other log-related issues exist on issue management systems, such as Jira. With regard to system operation, the developers and operators may have different backgrounds and understandings of the systems introducing further ambiguity. Studies show that even in leading software companies it is difficult to find thorough and complete guidelines to improve developers' logging behaviours [14]. Although there are different public resources such as blog posts [19], that share the best practices of experienced developers, they are very high-level and application-specific. Due to the presence of subjectivity in log instruction writing, the optimality of the current human-centric logging is questionable even for well-maintained projects [9], [10] irrespective of their origin (i.e., industry or open source). Notably, the guidelines are not designed with the automatic methods as the intended log consumers in mind.

Another important aspect of the current logging state is that with the emergence of learning-to-log [10], logging instrumentation can be achieved semi-automatically. These methods change the entity (an algorithm collaborating with the developer) that inserts log instructions. The problem of learning to log is composed of two sub-problems concerned with *where-to-log* [20]–[23] and *what-to-log* (log level [16], [24], [25], static text [7], [26], [27], and variables [11], [28], [29]. Many of the proposed semi-automatic logging methods assume the prior existence of some human-centred logging instrumentation. The semi-automatic methods are evaluated by comparing the novel instrumentations against the existing ones, and they improve the performance on different operational tasks. *This gives grounds for validity for the semi-automatic instrumentation*. However, the diagnosis is executed by a human, not automatic operation methods. Thereby, it is not clear if and how much improvement the semi-automatic instrumentation brings to the automatic methods. As these approaches are human-centric, they are not tailored for automatic methods as log consumers.

B. Towards Automatic-centred Logging

Although imperfect, human-centred logging is useful for system development and operation and is de facto the golden standard. However, in modern IT systems, there is significant adoption of automatic log-based methods for system operation [4], [30]. The main goal of the latter is to replace and improve the human-centric system operation. Inevitably, the current automatic methods are trained with the same data as an operator would use if she would have conducted the analysis. The wide adoption of automatic operation methods suggests that human-centric instrumentation is also useful for them.

The introduction of an entity different from humans for log analysis invites the question if alternative logging instrumentation is better suited for automatic methods. For humans, the natural text of the logs is informative for not just identifying failures but obtaining clues for their resolution. However, practical experiences show that analysis of long log textual sequences often becomes cumbersome for humans [31]. This is particularly emphasised if the failure-related logs span

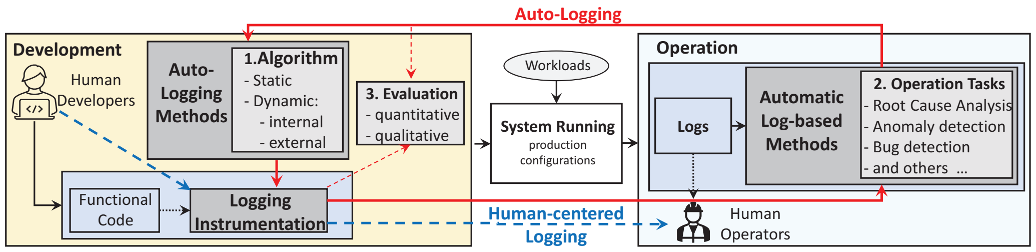

Fig. 1. Auto-logging is part of the overall system development and operation cycle. The grey colour denotes the three parts of the **strategy**.

multiple lines, separated by tens of other logs. In contrast, automatic methods can process arbitrary data types, i.e., they are not limited to a textual representation. Furthermore, the automatic methods can model and are sensitive to changes in sequences of many characters (where one character is one event). Therefore, they share different properties from humans.

Additionally, maximizing the information by combining the static text and parameters when generating the log, although useful for developers and human interpretation, causes significant challenges for the automatic methods. For example, log parsing is a predecessor of many automatic tasks as it reduces the noise in the generated output logs. However, log parsers are error-prone end can degrade the performance of the automatic methods (e.g., up to 40% performance drop for anomaly detection [32], [33]). This means that the automatic methods are severely impaired by the need to account for human-friendly representation. The existence of cases where the two log consumers behave differently introduces ambiguity on the optimality of human-centric logging for automatic methods. This prompts the need to envision alternative source code instrumentation better suited to automatic methods.

III. Auto-Logging: Logging Instrumentation for AI-enabled Methods for System Operation

Fig. 1 depicts the differences between auto-logging and human-centred logging. In auto-logging, the logging process is centred on automatic log-based methods as log consumers, i.e., it is automatic-centred. Automatic-centred logging can be performed by both humans and auto-logging methods. Under the term auto-logging, we further consider that the instrumentation is to be performed by auto-logging methods. The auto-logging method is an intelligent method that places the logging instructions in the most suitable locations as evaluated by the automatic method, which is conducting the operational activities. Notably, the auto-logging process is weakly-coupled with the functional code, as the log instructions should still capture the intention for the logging.

The key advantage of auto-logging is that unifies the logging making it non-dependant on the different levels of developers' expertise. The auto-logging methods can be trained on diverse data sources from within the project, including different data types alongside the source code (e.g., documentation, comments, specifications and similar). This enables the auto-logging methods to have a broad overview of the project, reducing the impact of the insufficient understanding that human developers may have. To ensure trustfulness, developers can still insert logging instructions as they found fit. However, auto-logging provides additional information tailored for the automatic methods humans are not required to analyze. The autologging idea is closely related to several works already address the problem of automatically inserting logging instructions (e.g., Log4Pref [21] or Log20 [20]). In the related works the target goal is to aid the developers as log consumers. In this regard, auto-logging has slight differences as it envisions the automatic methods as the main log consumers. Specifically, this enables to instrument the logging code through an interplay between two automatic methods: 1) the automatic instrumentation method on one side and 2) the automatic operational methods on the other side.

The idea of auto-logging envisions potential improvement during both system development and operation. One key question is how to realize auto-logging. The fundamental aspect in this regard is the choice of the **strategy** on how to instrument the source code to better suit the automatic methods. The strategy embodies an *algorithm* that instruments the source code such that the *performance* of the automatic methods on *different operational tasks* is equal to or better than human-centred logging. Given the aforenamed definition of the strategy for *auto-logging instrumentation*, three key aspects can be identified, i.e., 1) constructing an *algorithm*, 2) choices of an *automatic operational task*, and 3) measuring the *logging quality* improvement. We discuss them in the following.

1) Auto-Logging Algorithm Design: The **algorithm** provides a mechanism on how to instrument the source code. In the most general form, the algorithm receives as input non-instrumented and outputs instrumented source code. The special case where human-centred instrumentation exists is also subject to auto-logging. To derive potential solutions, we make an assumption about the relation between the system (i.e., the functional code) and its environment (with the automatic methods as its main constituent). Specifically, we examine whether the system is allowed to interact with the automatic methods during the instrumentation. Accordingly, we envision two possible strategies: 1) *static* and 2) *dynamic*.

The *static* strategies assume that the logging code instrumentation is done in isolation, i.e., without influence from the environment (despite for evaluation). Approaches from intelligent code synthesis are examples of this algorithmic category [34], [35]. The *dynamic* strategies assume that the system may interact with its environment. The interaction is achieved by workloads on which the system exercises certain behaviours (e.g., logs generated from verification tests). The existence of diverse workloads is one precondition for dynamic strategies. Additionally, automatic methods for the operational tasks that interact with the system may exist. As the system is allowed to interact with its environment, inevitably, the algorithm (”Agent 1”) can reuse the fingerprint of its behaviour in the logging instrumentation (e.g., evaluated by the automatic methods as ”Agent 2”). The ”Agent 1” may start with some initial instrumentation (e.g., random logging placement within the source code). Afterwards, the software system runs iteratively against the workloads, while ”Agent 1” applies an *algorithm-designed objective* to refine the logging instrumentation until logging improvement is achieved.

Conditioned on the availability of automatic methods during logging instrumentation, we identify two possible *algorithmic-design objectives*: 1) *internal* and 2) *external*. The *internal* objective utilizes just system behaviour in response to the different workloads irrespective of the automatic methods. One type of intrinsic objective is to arrange the log instructions that minimize the number of log instructions while enabling to reach each method in no more than n-steps. Such behaviour can be obtained by examining the invocation chains in response to the workloads. The second optimization criteria group are the *external* ones. The external objectives directly evaluate the logging quality for desired properties. Compared to the *internal*, the *external* objectives receive direct feedback from the environment on the goodness of the current instrumentation. In the simplest case, the feedback is given by the automatic method for a single automatic task. The log data of the current instrumentation in response to the workloads are evaluated by the capability of the automatic method (”Agent 2”) to achieve high performance on the operational task assessed by some performance criteria. This value is presented to ”Agent 1”. ”Agent 1” uses this information to exploit similar configurations if ”Agent 2” is showing good performance, or completely changes it in the opposite case.

2) Automatic Operational Tasks: One strong aspect of human-centric logging is the universality of its textual representation. Its textual nature abstracts various concepts that hint to operators not just how to detect but also how to resolve problems, i.e., they are multi-purposeful. In the ideal case, the auto-logging should improve the performance of the automatic methods on all of the **operational tasks**. Therefore, an important dimension of auto-logging is the degree of logs’ multi-purposefulness. In the case of auto-logging, the multi-purposefulness depends on the considered operational tasks. Although having one instrumentation for the source code is attractive in terms of efficiency, another possible instrumentation is to instrument the source code for each task. By choosing potentially different log positions or event descriptions for the tasks different log instructions may be relevant. Different log instructions for different tasks introduce the idea of log instructions classes, where each log is used for the targeted task. The idea of logging classes can potentially reduce the analysis time as it restricts the relevant logs to analyze. Notably, the idea of log classes goes beyond the debugging/operational classes introduced by the current log levels because it clearly states the *operational* task the log is used for (anomaly detection, safety analysis and similar).

3) Logging Quality Evaluation: We identify two quality evaluation criteria 1) quantitative, and 2) qualitative. Quantitative criteria characterize explicit performance improvement. When logging is tailored for a single task, logging quality can be evaluated as the difference in the performance (assessed by some metric) of the automatic methods when trained with logs from auto-logging over human-centred logging instrumentation. In the case of multiple automatic tasks, the quality evaluation is more challenging as different automatic tasks may be evaluated by different metrics. In this case, the performance evaluation can be found as a Pareto Optimal set of solutions, meaning several optimal configurations may exist. Furthermore, auto-logging should not introduce performance overhead concerning execution time and space occupancy. Inserting too many instructions may improve quality but can increase the monitoring overhead affecting the system functionality. In addition, automatic logging should preserve the integrity and safety of the system. It should ensure that sensitive information is adequately encoded. Another view on quality evaluation can be qualitative. Bogatinovski et al. [16] introduced a data-driven framework to qualitatively evaluate the properties of the log instructions. However, this evaluation is concerned with human-centric logging but may give guidelines for qualitative auto-logging evaluation.

In summary, auto-logging is a non-trivial problem. A potential solution involves several subproblems related to text generation, relevance ranking, optimal placement, automatic text understanding, automatic task selection, and representation learning. At the same time, auto-logging methods should introduce low-performance overhead and preserve system integrity. All of the above criteria should be considered when evaluating the logging quality of the auto-logging methods.

IV. Future Plans

As a first step to enable the development of an auto-logging agent, we aim to create a publicly available benchmark environment that includes target software systems, automatic operation methods and auto-logging instrumentation algorithms. The benchmark will be the gold standard that supports the development and evaluation of auto-logging approaches while enabling their fair comparison.

In Section III, we formulated the dynamic strategy as a sequential decision-making problem. By doing so, we open the application of ideas from reinforcement learning [36] to learn the auto-logging method. The inclusion of an ”Agent 2” (the automatic method) as a critique on the ”goodness” of the intermediate logging instrumentation (generated by ”Agent 1”), paired with ideas such as building a learning curriculum and constructing surrogate quality evaluation functions enable informative guidance on what best suits the automatic methods to achieve better instrumentation (similar to the solution for matrix multiplication [37]). Furthermore, given the textual nature of the problem, we are interested in combining NLP methods as agents (e.g., Transformer neural network [38]), and the aforenamed ideas from reinforcement learning to model the interaction between ”Agent 1” and ”Agent 2” as the most natural fit for the problem of auto-logging.

V. Conclusion

The recent adoption of log-based automatic methods for system operation requires enriched logs that differ greatly from the current human-centric ones. In this paper, we introduce the idea of auto-logging, which entangles the logging instrumentation with automatic methods to tailor the logs towards their specific properties. We believe that auto-logging instrumentation is very important as it can lead to faster and better system development and operation.

References

[1] 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*, pp. 1–17, 2020.

[2] O. Michael. (2002) ”Sarbanes-Oxley Act of 2002”. USA Senat. [Online]. Available: ”https://www.govtrack.us/congress/bills/107/hr3763”

[3] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, “A survey on automated log analysis for reliability engineering,” *ACM Comput. Surv.*, vol. 54, 2021.

[4] P. Notaro, J. Cardoso, and M. Gerndt, “A survey of aiops methods for failure management,” *ACM Trans. Intell. Syst. Technol.*, vol. 12, 2021.

[5] J. Bogatinovski, S. Nedelkoski, G. Madjarov, J. Cardoso, and O. Kao, “Leveraging log instructions for log-based anomaly detection,” in *2022 IEEE International Conference on Services Computing (SCC)*, 2022, pp. 321–326.

[6] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, p. 17351780, 1997.

[7] P. He, Z. Chen, S. He, and M. R. Lyu, “Characterizing the natural language descriptions in software logging statements,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018, p. 178189.

[8] J. Cândido, H. Jan, M. Aniche, and A. van Deursen, “An exploratory study of log placement recommendation in an enterprise system,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2021, pp. 143–154.

[9] B. Chen and Z. M. (Jack) Jiang, “Characterizing logging practices in java-based open source software projects – a replication study in apache software foundation,” *Empirical Software Engineering*, vol. 22, pp. 330–374, 2017.

[10] 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*. IEEE Press, 2015, p. 415425.

[11] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, “Improving software diagnosability via log enhancement,” *ACM Trans. Comput. Syst.*, vol. 30, 2012.

[12] A. R. Chen, T.-H. Chen, and S. Wang, “Demystifying the challenges and benefits of analyzing user-reported logs in bug reports,” *IEEE Transactions on Software Engineering*, vol. 26, no. 8, 2021.

[13] B. Chen and Z. M. J. Jiang, “Extracting and studying the logging-code-issue-introducing changes in java-based large-scale open source software systems,” *Empirical Software Engineering*, vol. 24, pp. 2285–2322, 2019.

[14] 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 *Companion Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2014, p. 2433.

[15] B. Chen and Z. M. J. Jiang, “A survey of software log instrumentation,” *ACM Comput. Surv.*, vol. 54, 2021.

[16] J. Bogatinovski, S. Nedelkoski, A. Acker, J. Cardoso, and O. Kao, “Qulog: Data-driven approach for log instruction quality assessment,” in *30th International Conference on Program Comprehension (ICPC 22)*. New York, NY, USA,: ACM, 2022.

[17] Z. Li, T.-H. Chen, J. Yang, and W. Shang, “Studying duplicate logging statements and their relationships with code clones,” *IEEE Transactions on Software Engineering*, vol. 48, pp. 2476–2494, 2022.

[18] F. Salfner, S. Tschirpke, and M. Malek, “Comprehensive logfiles for autonomic systems,” in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004.

[19] C. Eberhardt. The art of logging. [Online]. Available: https://www.codeproject.com/Articles/42354/The-Art-of-Logging

[20] X. Zhao, K. Rodrigues, Y. Luo, M. Stumm, D. Yuan, and Y. Zhou, “The game of twenty questions: Do you know where to log?” in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2017, p. 125131.

[21] K. Yao, G. B. de Pádua, W. Shang, S. Sporea, A. Toma, and S. Sajedi, “Log4perf: Suggesting logging locations for web-based systems’ performance monitoring,” in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2018, p. 127138.

[22] 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)*, 2018, pp. 61–71.

[23] 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 *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, 2012, pp. 293–306.

[24] 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)*. NJ, USA: IEEE Press, 2021, pp. 1461–1472.

[25] A. Han, C. Jie, S. Wenchang, H. Jianwei, L. Bin, and Q. Bo, “An approach to recommendation of verbosity log levels based on logging intention,” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. New York, USA: IEEE, 2019, pp. 125–134.

[26] Z. Ding, H. Li, and W. Shang, “Logentext: Automatically generating logging texts using neural machine translation,” in *Proceedings of the 9th IEEE International Conference on Software Analysis, Evolution and Reengineering*, ser. SANER ’22. IEEE, 2022.

[27] A. Mastropaolo, L. Pascarella, and G. Bavota, “Using deep learning to generate complete log statements,” in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 2279–2290.

[28] Z. Liu, X. Xia, D. Lo, Z. Xing, A. E. Hassan, and S. Li, “Which variables should i log?” *IEEE Transactions on Software Engineering*, vol. 47, pp. 2012–2031, 2021.

[29] A. Rabkin, W. Xu, A. Wildani, A. Fox, D. Patterson, and R. Katz, “A graphical representation for identifier structure in logs,” in *Proceedings of the 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*. USA: USENIX Association, 2010.

[30] S. Locke, H. Li, T.-H. P. Chen, W. Shang, and W. Liu, “Logassist: Assisting log analysis through log summarization,” *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3227–3241, 2022.

[31] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, “Log clustering based problem identification for online service systems,” in *Proceedings of the 38th ICSE*. New York, NY, USA: Association for Computing Machinery, 2016, p. 102111.

[32] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, “Tools and benchmarks for automated log parsing,” in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 2019, p. 121130.

[33] V.-H. Le and H. Zhang, “Log-based anomaly detection with deep learning: How far are we?” in *2022 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2022.

[34] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. d. M. d’Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S.

Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals, "Competition-level code generation with alphacode," 2022. [Online]. Available: https://arxiv.org/abs/2203.07814

[35] F. Christopoulou, G. Lampouras, M. Gritta, G. Zhang, Y. Guo, Z. Li, Q. Zhang, M. Xiao, B. Shen, L. Li, H. Yu, L. Yan, P. Zhou, X. Wang, Y. Ma, I. Iacobacci, Y. Wang, G. Liang, J. Wei, X. Jiang, Q. Wang, and Q. Liu, "Pangu-coder: Program synthesis with function-level language modeling," 2022. [Online]. Available: https://arxiv.org/abs/2207.11280

[36] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[37] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, pp. 47–53, 2022. [Online]. Available: https://doi.org/10.1038/s41586-022-05172-4

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 60006010.