Research Statement

Yintong Huo
School of Computing and Information Systems, Singapore Management University
Email: ythuo@smu.edu.sg
28 (Day) 11 (Month) 2024 (Year)

Background

Modern software systems play a crucial role in our daily lives, powering everything from search engines to communication platforms. Even minor failures in these systems can significantly impact millions of users, making software reliability a critical area of research. Traditional **software reliability engineering** methods often require extensive manual inspection of software performance data. However, the introduction of deep learning has opened new avenues for automating failure management. My research concentrates on developing **AI-based techniques** that provide reliability assurance in an extensive, practical, and generalizable way. In general, the goal of my research is to improve the efficiency and reliability for software development and operations.

To achieve the target, I have worked on language models to analyze textual software data, such as generating functional code, repairing buggy code, detecting anomalies in software logs, and automatically managing runtime failures. Additionally, I am exploring multimodal software engineering, particularly in user interface and web applications that respond to human instructions. In the following, I outline three of my main research areas, i.e., context-aware code intelligence, AIOps, and multimodal software development.

Research Areas

1. Context-aware Code Intelligence

Code production is a critical stage in software development that directly impacts software reliability. Learning-based techniques, especially LLMs, demonstrate advancements in understanding programming patterns and generating code. However, these models are trained on general textual data, neglecting domain knowledge in code practice; for example, they overlook the unique structure and syntax of programming languages. This gap can lead to compromised code quality and introduce errors. To bridge the gap, my research on context-aware code intelligence focuses on two key aspects, current capabilities of LLMs and the domain knowledge that can enhance code production. Specifically, I aim to answer three questions: How can we better understand the programming capabilities of LLMs? How can we integrate useful domain knowledge into these models for improved code generation?

Understanding code models' capabilities. To understand the programming capabilities of code models, we established a series of benchmarks across different tasks. For instance, we conducted the first empirical study on using LLMs to generate

logging statements from code inputs [1]. This study created a large-scale benchmark to assess the effectiveness and generalization of LLMs in logging. The analysis revealed limitations in the models' ability to capture complete code semantics, highlighting the necessity of incorporating broader programming contexts and control flow information to improve logging performance.

Domain knowledge for code intelligence. We aim at enhancing learning-based models by integrating domain knowledge from program analysis. More specifically, we are interested in identifying valuable programming resources and improving existing NLP techniques for code production. The first question drives me looking into knowledge mining from open forums like StackOverflow. For instance, one of our research projects [2] develop an automated API resolution tool to link diverse APIs to their official documentation, enabling swift comprehension of API usage patterns. The second question leads us to incoporate extensive static contexts [5] (such as call graphs and type information) beyond individual method boundaries for generating logging statements [3]. We also integrate domain knowledge into prompts for fixing program errors, using clustering and failure templates to repair buggy code [4].

Future Perspectives. While recent years have many works on generative code models with promising results, these LLMs struggle with complex tasks due to their limited sequential reasoning capabilities. To maximize their potential, there is a growing trend toward developing agent systems that have evolved from standalone assistants into nearly autonomous agents, capable of outsourcing tasks and adapting their behavior for more sophisticated human requirements. In this framework, an LLM acts as the main controller or "brain", managing the sequence of operations necessary to complete a task. Future work could focus on creating end-to-end agent-based solutions that assist users in completing daily engineering tasks effectively.

2. AIOps (AI for IT Operations)

Fault tolerance is a crucial aspect in reliability engineering, which involves a system's capability to sustain its intended functionality despite failures (i.e., manifestations of faults). Enhancing this resilience requires timely and accurate detection and diagnosis of failures to implement appropriate mitigations, such as recovery through redundant mechanisms or rolling back to a previous state. Traditional methods often depend on manual monitoring of software telemetry data; however, the growing complexity of systems and the vast amounts of output data make manual inspection increasingly impractical.

In my pursuit of enhancing fault tolerance, my research focuses on developing learning-based log management techniques, as logs serve as primary resources for software monitoring. These semi-structured logs contain both developer-written events and automatically generated runtime parameters. Current studies oversimplify log sequences as character strings, neglecting the intricate semantics embedded in these events. Therefore, my work aims to pioneer practical, adaptable, and scalable log analytical methods by thinking three questions: (1) How do semantics in logs facilitate log analysis? (2) Can log techniques adapt to complex and evolving software demands? (3) How do we bridge the gap between academic tools and real-world deployment?

Semantic-aware log parsers. As the fundamental step of log analysis, log parsing aims to extract structured templates and parameters from raw log messages. However, existing log parsers are syntax-based, missing valuable semantic details in logs. To address this limitation, our work, SemParser [6], extracts log semantics through a two-step process: pairing parameters and their descriptions within individual logs, and inferring implicit semantics and computing structural outputs based on historical logs. We further explore LLMs for parsing. For example, DivLog [7] harnesses LLMs' in-context learning (ICL) ability by sampling parsing examples for each target log, creating prompts for LLMs to parse logs. Additionally, LILAC [8] tackles the LLM inefficiencies problem with an adaptive parsing cache. The core idea, adaptive parsing cache, is to store previous log templates and employ tailored matching and adaptive updating operations. LILAC outperforms state-of-the-art methods in terms of parsing accuracy while maintaining higher efficiency.

Scalability and adaptivity in log analytics. In handling large daily logs, identifying critical failures among anomalies poses a key challenge for engineers. Within software evolution process, parsing errors, shifting log events, and unstable sequences further complicate the process. EvLog [9] addresses these hurdles using a multi-level representation extractor to retain log semantics and a one-to-all anomalous log classifier with an attention mechanism to handle unstable sequences. The evaluation across two systems shows that, EvLog maintains the effectiveness in analyzing evolving-software log files without further training. Besides, we developed SeaLog [10] to build an accurate, lightweight, and adaptable anomaly detection framework. The core component of SeaLog is a Trie-based Detection Agent, using a dynamically expanding Trie structure for real-time detection. It also incorporates expert feedback to handle evolving log data. Its deployment within real-world cloud company futher demonstrates practical applicability.

Log sequence simulator. Log analysis grapples with a significant challenge—insufficient practical data due to privacy constraints in industrial logs and the simplistic nature of publicly available lab logs. AutoLog [11] tackles this issue by leveraging program analysis techniques to simulate log sequences based on the execution order of logging statements. In experiments across 50 popular Java projects, AutoLog significantly outperforms existing log datasets, acquiring 9-58 times more log events from the same system and generating log messages 15 times faster using a single machine.

Future perspectives. Modern software systems produce a variety of data modalities, including logs, typologies, KPIs, and incident tickets. While these data types are often interconnected, previous research has typically addressed them in isolation. My upcoming research aims to develop multimodal operational strategies to enhance software reliability, such as performing root cause analysis by analyzing combined service dependency graphs and logs. Additionally, I plan to explore the integration of development and operations (DevOps), utilizing operational data (like logs) to formulate strategies for addressing software bugs. This approach leverages valuable feedback from software operations to help identify and eliminate potential faults during the development phase.

3. Multimodal Software Development

Websites are integral to the modern digital landscape, with over 1.11 billion active sites and approximately 252,000 new ones launched daily. The process of developing a graphical user interface (GUI) for websites typically begins with graphic designers creating the visual layout, which is then handed off to programmers to implement. This translation from design to code is often time-consuming and prone to errors, especially for those without specialized expertise.

While multimodal LLMs have shown significant advancements in handling image and code, their application to GUI understanding and generation remains unexplored. As an emerging area, developing MLLM-based GUI-to-code applicable to real-world scenarios involves several challenges. First, there lacks benchmark that mimics comprehensive and realistic development scenarios, which can include mobile app design and desktop UI design. Webpages may feature a variety of elements and visual designs, each with differing complexities. It is crucial to categorize the levels of automatic GUI generation to capture this diversity effectively. Second, there exists a modality gap between visual features in screenshots and code snippets. Webpages often include intricate layouts and styles, requiring the model to have a deep understanding of a GUI framework's components. Such understanding must be sufficient to accurately reproduce elements with detailed attributes such as colors, fonts, margins, and positioning.

Autonomous webpage development. My initial work on this problem resulted in the first MLLM-based segment-aware method for web UI generation. While existing MLLMs struggle with generating accurate UI code, our study observes that breaking down full screenshots into smaller visual segments will improve performance. This decomposition allows models to conduct more reasoning steps, each focused on a manageable sub-generation task. Motivated by the traditional “divide and conquer” algorithm, our method, DCGen [12], decomposes a complicated screenshot into smaller, doable visual segments, solves each part individually, and then combines the solutions for the original problem.

Comprehensive dataset and benchmarks. High-quality and representative web development dataset are the crucial resources and for multimodal software engineering research. We introduce a benchmark consisting of complicated real-world static webpages, as well as the first collection of user-interactive webpages [13] for this area. Upon these benchmarks, we empirically analyze the effectiveness and common failures of MLLMs-generated UI code, revealing several current limitations and implications for future improvements.

Future perspective. While code generation from natural language instructions has been extensively studied, developing software that meets multi-modal requirements remains an unexplored area. Future research could focus on the end-to-end software development process, aiming to simultaneously optimize front-end design, back-end architecture, and resource management. Additionally, the development of GUI agents presents another promising research direction. These agents aim to understand software interfaces and execute user instructions by mimicking human interactions, such as searching, clicking and typing. This area could significantly enhance user experience and streamline the software development process.

Selected Publications and Outputs

For a complete publication list, please refer to my Google Scholar or DBLP page.

1. Yichen Li*, Yintong Huo*, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, Lionel C. Briand, and Michael R. Lyu, “Exploring the Effectiveness of LLMs in Automated Logging Statement Generation: An Empirical Study.” To appear in IEEE Transactions on Software Engineering, 2024.
2. Yintong Huo, Yuxin Su, Hongming Zhang, and Michael R. Lyu, “ARCLIN: Automated API Mention Resolution for Unformatted Texts.” In Proceedings of the 2022 International Conference on Software Engineering, pp. 138-149.
3. Yichen Li, Yintong Huo, Renyi Zhong, Zhihan Jiang, Jinyang Liu, Junjie Huang, Jiazhen Gu, Pinjie He, and Michael R. Lyu, “Go Static: Contextualized Logging Statement Generation.” In Proceedings of the 2024 ACM on Software Engineering, pp. 609 – 630.
4. Yun Peng, Shuzheng Gao, Cuiyun Gao, Yintong Huo, and Michael R. Lyu, “Domain Knowledge Matters: Improving Prompts with Fix Templates for Repairing Python Type Errors.” In Proceedings of the 2024 International Conference on Software Engineering, pp. 12-24.
5. Yichen Li, Yun Peng, Yintong Huo, and Michael R. Lyu, “Enhancing LLM-based Coding Tools Through Native Integration of IDE-Derived Static Context.” In Proceedings of the 2024 IEEE/ACM International Conference on Software Engineering Workshop on Large Language Model for Code.
6. Yintong Huo, Yuxin Su, Baitong Li, and Michael R. Lyu, “SemParser: A Semantic Parser for Log Analytics.” In Proceedings of the 2023 International Conference on Software Engineering, pp. 881-893.
7. Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He, “DivLog: Log Parsing with Prompt Enhanced In-Context Learning.” In Proceedings of the 2024 International Conference on Software Engineering, pp. 2457-2468.
8. Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu, “Lilac: Log parsing using llms with adaptive parsing cache.” Proceedings of the ACM on Software Engineering, FSE, pp. 137-160.
9. Yintong Huo, Cheryl Lee, Yuxin Su, Shiwen Shan, Jinyang Liu and Michael R. Lyu. “EvLog: Identifying Anomalous Logs over Software Evolution”, Proceedings of 34th IEEE International Symposium on Software Reliability Engineering, pp. 391-402, 2023
10. Jinyang Liu, Junjie Huang, Yintong Huo, Zhihan Jiang, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Minzhi Yan, and Michael R Lyu. “Scalable and adaptive log-based anomaly detection with expert in the loop”. Arxiv, 2023.
11. Yintong Huo*, Yichen Li*, Yuxin Su, Pinjia He, Zifan Xie, and Michael R. Lyu. “AutoLog: A Log Sequence Synthesis Framework for Anomaly Detection”, Proceedings of 38th IEEE/ACM International Conference on Automated Software Engineering, pp. 497-509, 2023.
12. Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R Lyu. “Automatically generating UI code from screenshot: A divide-and-conquer-based approach”. Arxiv, 2024
13. Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zhiyao Xu, Michael R Lyu. “Interaction2Code: How Far Are We from Automatic Interactive Webpage Generation?”. Arxiv, 2024.