基于大语言模型的软件日志生成与静态调用上下文技术综述

1. 引言

1.1 日志在现代软件系统中的作用

现代软件系统更加庞大，实现的功能更加精细、复杂，一个功能往往需要多个模块间的协同。日志作为软件系统的重要辅助信息，在程序调试、错误定位以及系统维护中都是不可或缺的一环。在软件开发阶段，日志信息可以帮助开发者追踪程序的执行情况；在运维阶段，日志可以监测系统的运行状态乃至识别性能瓶颈；在系统故障时，日志可以回溯、定位故障点，提高诊断效率。

1.2 传统日志实践的局限性

在传统的日志实现中，往往依靠开发者人工插桩，日志的质量高度依赖开发者的经验。不同开发者对于日志的重视程度、设计思路都有不同的理解，而对于一个庞大的软件系统，日志需要多位人员合作实现，难以保证日志内容的一致性和完整性。部分日志存在冗余，增加了分析时的负担；而大部分日志则与程序上下文贴合不紧密，仅记录单一节点的运行状态，无法为复制的故障提供有效的诊断依据；

1.3 大语言模型赋能日志生成

近年来，大语言模型（LLM）发展迅速。LLM通过大量数据预训练，拥有极强的语义理解与生成能力。从代码生成角度，LLM 可以根据自然语言描述，自动生成语法正确、功能完整的代码片段；从代码补全角度，LLM可以基于上下文推测开发者意图，提供精确的代码补全建议；从注释生成与代码总结角度，模型能够提炼代码核心功能，生成通俗易懂的自然语言描述[1]。这证明了LLM对程序语法结构、语义逻辑的理解能力，为日志生成任务中的应用奠定了坚实的基础。

1.4 本文综述的研究范围

因此，本文将对近年来基于LLM的日志生成研究进行综述，重点分析静态程序分析技术在构建日志生成上下文中的作用，尤其聚焦调用图（Call Graph）的构建与多跳调用上下文提取技术。本文不系统讨论配置系统本身或日志分析、异常检测等下游应用，仅围绕 “日志生成” 与 “静态调用上下文构建” 两大核心展开。

2. 软件日志生成的传统方法

2.1 基于人工经验的日志插桩方法

早期软件系统中的日志生成主要依赖开发者手工插桩。该方法的优势在于日志设计可以贴合具体的业务需求，具有较强的针对性。但是日志的位置、内容取决于开发者的主观判断，不同开发者的设计风格差异大；且随着软件系统的规模发展，人工插桩的工作量呈指数级增长，难以覆盖所有关键的执行路径。该类方法难以满足现代软件系统对日志的需求。

2.2 基于规则与程序分析的日志生成方法

为解决人工插桩的弊端，研究者提出了基于规则与程序分析的自动化日志生成方法。该类方法通过解析源代码生成的抽象语法树、控制流图等中间表示，同时结合预设规则或程序结构分析，能够自动识别日志的插入点，并生成相应的日志语句。相较于人工插桩，此类方法显著提升了日志的自动化程度，确保日志风格一致性，但受限于预设规则的完整性，难以扩展至多样化的业务场景。

2.3 传统方法的不足与总结

可见，传统的日志生成方法存在三个通病：程序上下文信息有限、难以表达复杂执行语义、不适合复杂调用关系场景。亟待新的日志生成方法，以突破人工经验与预设规则的束缚。

3. 基于大语言模型的软件日志生成研究

3.1 基于大语言模型的日志生成方法

3.1.1 基于局部代码上下文的日志生成方法

早期基于LLM的日志生成方法多采用局部代码上下文作为输入，即将包含日志插入点的单个方法、代码块或代码片段提供给LLM。比如，直接将缺失日志的函数代码作为prompt提供给GPT-3.5、Codex等模型，让模型根据函数内部的变量操作、条件判断等逻辑生成相应日志[2]。优势在于实现简单、计算成本低，无需复杂的上下文预处理，但受限于上下文范围，生成的日志往往只能反映局部执行状态，难以满足复杂系统故障诊断的需求。

3.1.2 基于扩展程序上下文的日志生成方法

为弥补局部上下文的不足，研究者开始探索引入扩展程序上下文的日志生成方法，如以SCLogger为代表的技术路线[3]。该类方法通过静态程序分析技术提取跨函数的上下文信息，包括目标函数的调用者、被调用者、相关变量的传播路径等，将这些信息与局部代码上下文结合后，作为LLM的输入，从而提升日志的语义完整性与关联性。例如，在配置相关日志生成中，LLM不仅能获取配置解析函数的局部代码，还能获取通过静态分析得到的目标函数的配置加载函数、依赖的参数验证函数等上下文信息[4]。研究表明，引入跨方法的程序上下文信息能够显著提升日志内容的相关性和表达能力，使生成的日志更能反映系统运行的全局背景[2]。

3.2 思维链与上下文学习在日志生成中的作用

思维链与上下文学习技术进一步优化了基于LLM的日志生成效果。思维链通过引导LLM分步推理程序执行逻辑，从而明确复杂函数调用关系中的关键节点，比如先分析程序当前执行阶段、再识别核心

变量与业务含义、最后确定日志级别与描述内容[2]；上下文学习则通过在prompt中加入高质量的日志生成示例，让LLM学习目标项目的日志风格、格式规范与关键信息捕捉方式，减少了人工规则的设计[4]。这两项技术尤其适用于复杂调用关系的场景，生成更精准、规范的日志语句。

4. 静态程序分析驱动的调用上下文构建

4.1 程序上下文在日志生成中的重要性

高质量的软件日志不仅要求反应当前节点的运行状态，还需要反应该节点的执行路径、调用背景、依赖关系等信息。在复杂、大型的软件系统中，同一变量的取值或执行结果可能来自不同的调用路径，因此，对应的日志信息不能仅仅依靠局部上下文。比如函数调用上下文，可以体现当前函数的调用和被调用信息、帮助追踪调用链之间的参数传递信息，是业务逻辑的重要体现。因此，基于函数调用上下文生成的日志，可以反应更多的予以信息，帮助开发者在快速诊断或追踪执行路径。

4.2 基于调用图的多跳调用上下文建模

调用图（Call Graph）指描述程序中函数间调用关系的有向图结构，每个节点表示一个函数或方法，每条有向边代表函数间的调用关系[1]。在日志生成任务中，调用图是提取跨函数调用上下文的基础，通过遍历调用图，可以精准地获取目标函数的调用者、间接调用者等关键上下文信息[5]。基于调用图的多跳调用上下文建模是平衡日志信息量与计算成本的关键策略。其中，一跳调用上下文指直接调用目标函数的函数，能够提供目标函数的直接执行背景；在此基础上，两跳调用上下文则进一步包含直接调用者的调用者，能够呈现更完整的执行路径[4]。研究表明，随着调用上下文跳数的增加，日志的语义丰富度会显著提升，但会增加信息冗余度，会增加LLM输入负担，还可能干扰LLM捕捉核心信息[5]。在实际应用中，限制调用上下文的跳数（一般为两跳）可以在信息完整性与计算成本之间取得平衡，是目前主流的上下文建模方式[4]。

4.3 C++程序中调用图构建的特点与挑战

相较于其他语言，C++因为支持虚函数与动态绑定，在静态编译期难以确定虚函数的具体调用目标，需要通过类层次分析（CHA）等技术补全潜在调用关系[1]；同时，模板实例化机制导致同一模板会生成多个不同的实例化函数，需要准确识别这些实例化函数及其调用关系，避免调用遗漏；而且C++中的函数指针等特性会产生隐式调用关系，无法通过简单的语法分析识别，需要结合代码模式匹配与数据流分析进行补全[4]。因此，C++程序的调用图构建更加复杂，对静态分析技术的精准性要求更高。

4.4 基于两跳调用上下文的基线方法设计

结合上述分析，本文设计了基于两跳调用上下文的基线方法，为大语言模型日志生成提供高质量上下文输入。该方法的核心流程如下：首先，输入目标方法，通过 Clang 等前端工具解析C++源码，生成抽象语法树（AST）与函数调用关系，构建完整的程序调用图[1]；其次，基于调用图进行逆序遍历，从目标方法出发，提取一跳上下文和两跳上下文；最后，从所有有效两跳调用上下文组合中随机采样，返回其完整方法体，作为目标方法的扩展上下文[4]。该基线方法为基于大语言模型的日志生成

提供了一种轻量级且可扩展的上下文构建方式，既保证了上下文信息的丰富性，又避免了过度冗余，兼顾了C++程序的复杂特性[1]。

5. 基于大语言模型日志生成方法的优势与不足

基于大语言模型的日志生成方法具有两大核心优势：一是语义表达能力强，相较于传统规则驱动方法，能够生成更加自然、贴合业务逻辑的日志描述[2]；二是自动化程度高，无需人工设计大量规则，通过上下文学习即可适配不同项目的日志风格与规范[4]。

但该类方法也存在不足：首先，日志质量与输入的调用上下文完整性、冗余性强相关，可能会生成与执行逻辑不符的错误日志[2]；其次，大型闭源LLM依赖商业API，调用成本高，且存在隐私泄露的风险。小型开源模型虽然可以实现本地部署，但对复杂调用关系的理解能力有限[4]。

6. 挑战与未来研究方向

当前基于LLM的软件日志生成与静态调用上下文技术仍然面临诸多挑战，未来可从以下方向展开深入研究：一是更精准的调用上下文选择，目前多采用固定跳数的上下文提取策略，缺乏对业务场景的自适应调整，未来可以结合业务逻辑与故障模式，动态选择关键上下文节点；二是静态分析与LLM的深度结合，现有研究多将静态分析作为上下文提取工具，未来可探索让LLM直接理解调用图等静态分析结果，自主判断上下文相关性[1]；三是进一步优化小型开源模型的微调策略，降低模型部署成本[4]；四是构建专门的日志质量评估体系，减少当前对人工或传统文本相似度指标的依赖[2]。

7. 总结

本文系统综述了基于大语言模型的软件日志生成与静态调用上下文技术的研究进展。首先回顾了传统日志生成方法的特点与局限性，指出上下文信息不足是其核心短板；随后重点分析了基于大语言模型的日志生成研究，强调扩展程序上下文对提升日志质量的关键作用；深入探讨了静态程序分析驱动的调用上下文构建技术，详细阐述了调用图的核心作用、多跳上下文建模策略以及C++程序的适配挑战，并介绍了基于两跳调用上下文的基线方法；最后通过综合分析指出了当前研究的优势与不足，展望未来研究方向。

综合来看，静态调用上下文与大语言模型的结合是软件日志生成技术的重要发展方向。静态程序分析为日志生成提供精准、完整的跨函数上下文支撑，LLM则将上下文信息转化为高质量、自然的日志描述，两者的有机结合，有效解决了传统方法的局限性。未来随着静态分析技术的精准化、LLM的轻量化与适配性提升，软件日志生成技术将朝着更自动化、高质量、工程化的方向发展，为软件系统的维护与故障诊断提供更强大的支撑。

参考文献

[1] Hou X, Zhao Y, Liu Y, et al. Large Language Models for Software Engineering: A Systematic Literature Review[J]. ACM Transactions on Software Engineering and Methodology, 2024.

[2] Li Y, Huo Y, Jiang Z, et al. Exploring the Effectiveness of LLMs in Automated Logging Statement Generation[J]. IEEE Transactions on Software Engineering, 2024.

[3] Li Y, Zhang S, Wang X, et al. Go Static: Contextualized Logging Statement Generation[J]. IEEE Transactions on Software Engineering, 2025.

[4] Zhong R, Li Y, Yu G, et al. Beyond LLMs: An Exploration of Small Open-source Language Models in Logging Statement Generation[J]. arXiv preprint arXiv:2505.16590, 2025.

[5] Huo Y, Li Y, Su Y, et al. AutoLog: A Log Sequence Synthesis Framework for Anomaly Detection[J]. arXiv preprint arXiv:2308.09324, 2023.