分布式进程的后验 LLM 支持调试

Dennis Schiese[1] and Andreas Both[1]

Web & Software Engineering Research Group
Leipzig University of Applied Sciences

Abstract. 在本文中，我们解决了手动调试的问题，这个问题至今仍然耗费大量资源，并且在某些方面仍显古老。这个问题尤其体现在日益复杂和分布式的软件系统中。因此，本工作的目标是引入一种方法，该方法可能适用于任何系统，在宏观和微观层面均能简化这一调试过程。这种方法利用系统的进程数据与生成式 AI 相结合，以生成自然语言解释。这些解释是从实际的进程数据、接口信息和文档中生成，以更有效地指导开发人员理解流程及其子流程的行为和潜在错误。在这里，我们展示了一个使用此方法应用于基于组件的 Java 系统的演示器。然而，我们的方法是与编程语言无关的。理想情况下，生成的解释将提供对过程的良好理解，即使开发者不熟悉所考虑系统的所有细节也是如此。我们的演示器作为开源网络应用程序免费提供给所有用户。

Keywords: 人工智能支持的软件开发 · 调试 · 大语言模型

1 介绍

软件开发，与其他许多领域一样，受到快速增长的自动化的影响，这对软件开发效率产生了相当大的影响。结果是，现有软件的数量和复杂性继续增加。这一点在分布式系统中尤为明显，在这些系统中，进程利用多个软件组件（例如 Web 服务），以及在开发复杂的软件架构时，错综复杂的依赖关系和庞大的代码库增加了更多挑战。然而，随着软件开发的加速，传统的流程如概念化、实现、调试或测试仍然至关重要，即使人工智能代理开始采用它们也是如此。在调试的情况下，像*自动程序修复*（APR）这样的概念提供了许多不同的技术和工具来解决这个问题 [1,7]。尽管这些方法的表现不断提高，但它们仍会遇到失败的情况。在这种情况下，手动调试成为必要过程。

在此演示中，其中一个考虑的过程——仅由两个组件（Web 服务）组成——可以触发超过 5000 次可追踪的跨组件和组件内方法调用（不包括外部库）。对这样一个过程进行手动调试是繁琐且耗时的。为了解决这一挑战，我

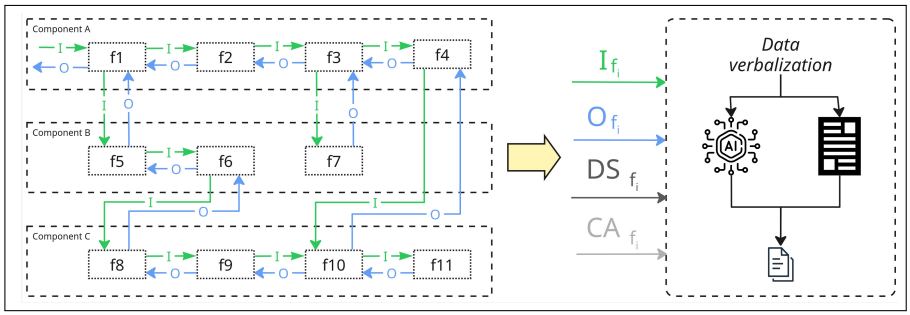

Fig. 1: 一个涉及三个相互作用组件的抽象过程。每个单元（f）可以带参数或不带参数调用另一个，并且可能接收返回值。这些调用以及输入（I_{f_i}）、输出（O_{f_i}）、文档字符串（DS_{f_i}）和调用者（CA_{f_i}）等值会被记录下来，并由一个*数据语言化器*用于生成对 f_i 的解释，使用模板或生成式 AI。

们在 [4,5] 中提出了一种在系统运行期间检查并事后对每个方法的数据进行口头描述的方法。在这里，我们扩展了这种方法，并明确关注组件导向的解释（见图 1），以简化调试过程，并实现了相应的演示器。为了减少搜索过多解释所带来的开销，采用大型语言模型（LLMs）来提取子调用方法中相关方面的内容，并将其整合到当前方法的自然语言解释中。我们旨在传播所有子过程中所有相关的方面，而不太相关的方面可能不会用于父方法，从而使得开发人员可以收到关于特定过程实际发生情况的简洁说明。这种方法或类似的方法，即 任何利用进程数据生成解释以供手动调试的工作，据我们所知目前尚未见报道。然而，在 APR 领域中，使用生成式 AI 被认为是一种主导趋势 [3,6]；[2] 提出了一种面向（自动化）调试的方法，同时旨在支持 LLM 的补丁生成。

2 基于 LLM 的组件式系统调试

在我们的方法中，我们考虑任何分布式或非分布式的（基于组件的）系统，例如 一组 Web 服务。由于我们的方法是建立在具体的流程数据之上的，我们建立了一个三元存储来持久地存储这些数据。作为示例系统，我们使用了 Qanary[1]。它被选中的原因是其架构利用了 Web 服务，这使其特别适合于分析分布式系统——并且对于调试来说是一个特殊且具有挑战性的案例。在执行过程之后，每个方法调用（任何组件中）存储的数据将用于事后生成

[1] https://github.com/WDAqua/Qanary

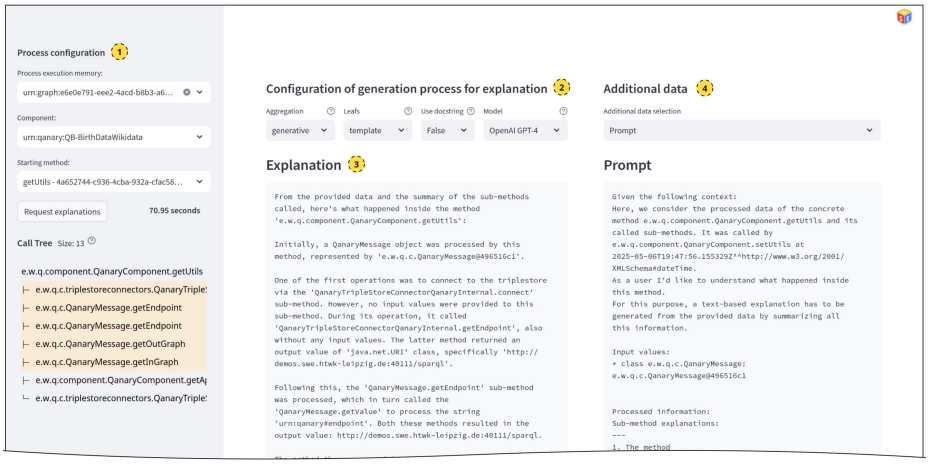

Fig. 2: 演示者的截图。它显示了所选过程、组件和所考虑（当前调试的）示例组件系统的相应方法的 LLM 生成解释及其对应的提示。

解释，使用 Java 服务 [2]。我们实现了一个基于模板的生成和一个使用用户选定的大语言模型的方法。

如图 2 所示，该演示器是一个前端，它促进了对先前执行的对所考虑的示例（但实际）基于组件的系统的具体方法的解释。它在我们的研究小组的网站 [3] 和 GitHub[4] 上可用。视频教程可在 [5] 处观看。

过程配置1– 用户从最后（已经执行的）过程中选择一个，其中至少有一个方法创建了日志条目的组件，以及开始方法（即 从整个过程调用树中，子树的根方法，这一点需要解释清楚）。为了提供应用程序行为的完整解释，有必要对所有下属的方法调用进行说明（即 那些来自子树的方法）。此外，在生成更高层次的方法说明时会使用这些说明，要么将其纳入提示中，要么作为模板的一部分。可以从调用序列中的方法（也包括从任何组件后续调用的方法）中选择来生成并展示解释。

生成过程配置说明2– 用户可以决定如何根据调用树为选定的方法生成解释。如前所述，下属调用也按需进行解释。定义了两种场景：第一种，一个不调用其他方法（调用树的叶子）的方法仅使用该方法的过程数据进行解

[2] https://github.com/WSE-research/qanary-explanation-service
[3] https://wse-research.org/llm-driven-debugging/
[4] MIT 许可证的源代码可在 https://github.com/WSE-research/frontend_aggregated_explanations 获取
[5] https://wse-research.org/LLM-supported-debugging-video

释。第二种，调用方方法通过汇总其下属调用的解释，并包括自己的处理数据来聚合这些解释。解释——无论是叶子节点还是聚合节点——都可以使用模板或选定的大语言模型生成。用户还可以选择是否包含方法的文档字符串以及使用哪个大语言模型。

说明3– 所选方法的说明如下所示。在这里，用户可以测试哪种配置能为他们的需求提供最佳解释。

额外数据4– 用户可以选择并显示其他一些数据，例如基于生成的解释提示或使用的文档字符串（如适用）。

3 结论与未来工作

在这篇论文中，我们提出了一种改进和高效的手动调试示例软件系统的方法。在应用程序执行过程中存储实际进程数据（输入和输出），并利用这些数据解释已执行的进程的想法适用于任何规模的系统。特别地，我们的方法与语言无关，并且适合解释分布式组件系统。演示者通过使用大语言模型生成各种方法、过程和组件的解释来展示此数据的有用性，从而使所考虑软件系统的高效分析成为可能。我们这个演示的主要重点是证明基于输入参数和返回值等依赖进程的数据的方法的可行性。然而，通过纳入文档字符串，我们也迈出了利用支持（静态）数据的第一步，这可能会进一步提高质量。在未来，进一步增加诸如源代码或性能信息（如果可用的话）之类的额外数据可能进一步提升质量。

References

1. Anand, A., Gupta, A., Yadav, N., Bajaj, S.: A comprehensive survey of AI-driven advancements and techniques in automated program repair and code generation. CoRR **abs/2411.07586** (2024)
2. Kang, S., Chen, B., Yoo, S., Lou, J.G.: Explainable automated debugging via large language model-driven scientific debugging. Empirical Softw. Eng. **30**(2), 45 (2024)
3. Kwon, S., Lee, S., Kim, T., Ryu, D., Baik, J.: Exploring LLM-based automated repairing of ansible script in edge-cloud infrastructures. J. of Web Eng. **22**(6) (2023)
4. Schiese, D., Perevalov, A., Both, A.: Post-hoc insights: Natural-language explanations for AI-enhanced/-integrated software systems. In: SEMANTiCS 2024 (2024)

5. Schiese, D., Perevalov, A., Both, A.: Towards LLM-generated explanations for component-based knowledge graph question answering systems. In: 23rd International Conference on WWW/Internet (ICWI). IADIS (2024)
6. Xia, C.S., Wei, Y., Zhang, L.: Automated program repair in the era of large pre-trained language models. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). pp. 1482–1494 (2023)
7. Zhang, Q., Fang, C., Ma, Y., Sun, W., Chen, Z.: A survey of learning-based automated program repair. ACM Trans. on Softw. Eng. and Methodology **33**(2) (2023)