基於模糊測試與 LLM 之程式修復框架

專題編號：114-1-CSIE-S002
執行期限：113 年第 1 學期至 114 年第 1 學期
指導教授：郭忠義
專題參與人員： 111590002 鄭重雨
111590038 卓柏辰

一、摘要

本專題建立一套結合模糊測試與大語言模型的自動修復框架，旨在開發者在不更動專案架構的情況下，自動偵測錯並生成修補程式碼。本框架的流程涵蓋錯誤發現、重現、修復與驗證：以 AFL++產生測資觸發漏洞，透過 Sanitizer 收集診斷資訊，並藉由 RAG 強化 LLM 在 Agentic Framework 下生成修補方案，最終經自動化驗證確保修復正確與功能完整。此方法有效降低人工介入，並提升修復效率與軟體可靠性。

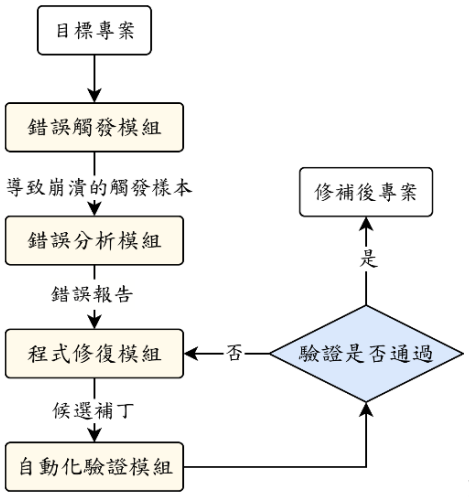
關鍵詞：模糊測試、自動修復、LLM、錯誤驗證。

二、緣由與目的

在數位時代，軟體已成為社會基石，而程式缺陷與安全漏洞則構成重大挑戰。尤其在 C/C++等具底層控制的語言中，記憶體錯誤不僅可能導致程式崩潰。雖然模糊測試能高效發現此問題，但修復仍高度依賴人工。為此，自動化程式修復（APR）技術應運而生，而近年 LLM 憑藉著程式碼理解與生成能力，為 APR 帶來新契機。然而，現有應用仍面臨兩大挑戰：其一，受限於上下文長度，難以處理跨函式或檔案的複雜錯誤；其二，缺乏將低階錯誤資訊轉化為 LLM 可理解指令的有效方法。因此，本專題旨在設計能整合模糊測試與 LLM 的自動化修復方法，以降低人工成本並提升修復效率。

三、研究架構

本框架的整體架構採用循序的管線式設計，其資料流與控制流由一個流程器進行統籌，以確保各階段的銜接。



圖(一)

四、研究範圍與技術

(一) 模糊測試

此階段在自動化生成大量測試案例，以程式執行路徑並觸發潛在錯誤。我們採用模糊測試引擎 AFL++，它能根據程式碼覆蓋率的回饋，產生變異輸入種子，從而高效地發現程式崩潰的案例。

(二) 錯誤分析

此階段任務是驗證模糊測試發現的崩潰樣本，並收集詳細診斷資訊。對於每一個崩潰樣本，啟用 AddressSanitizer 與 UndefinedBehaviorSanitizer 的環境中，嘗試重現程式崩潰。確保該錯誤並非偶發性問題，並獲取由這兩種產生的詳細錯誤報告。

(三) 程式修復

此工作流程開始會生成一個含有錯誤情境的提示詞，並透過 RAG 技術，從歷史案例庫中加入相似的解決方法作為參考。而 LLM 並非直接生成答案，而是被授權於 Agentic Framework 下自主運作，循環執行三個核心操作：

- 查閱程式碼：以請求更多相關檔案以擴展其理解。

- 紀錄觀察並總結：創建記錄以輔助後續推理。
- 提供修補方案：在資訊充足後生成最終的候選補丁。

(四) 自動化驗證

此階段構成一個自動化驗證迴圈，用以評估補丁的正確性與有效性。系統針對每個修復建議依序進行三項檢驗：

1. 修補後的程式碼能否順利編譯。
2. 是否通過原先的測試案例以維持功能完整性。
3. 能否避免最初觸發崩潰的輸入確認程式不再崩潰。

五、評估方法

為客觀評估框架的成效，將以 ARVO 資料集作為實驗基準。該資料集收錄了大量源自於 OSS-Fuzz 的真實漏洞，並提供崩潰的觸發樣本、開發者所撰寫的修復補丁以及可重新編譯的環境。

六、研究結果

整體而言，本框架在不需人工介入的情況下，達成 42.8% 的自動修復成功率，顯示 LLM 在結合模糊測試與診斷資訊後，具備實質的錯誤理解與修復能力。

而在建置層面上，平均建置成功率達 39.4%。

類型	建置成功率
AddressSanitizer: stack-use-after-return	1.000
MemorySanitizer: use-of-uninitialized-value	0.294
AddressSanitizer: heap-buffer-overflow on address	0.500
AddressSanitizer:heap-use-after-free on address	0.200
AddressSanitizer: SEGV on unknown address	1.000

七、結論

本專題成功實作一套結合模糊測試與大語言模型之自動化修復框架，完成串聯錯誤發現、重現、修復與驗證四個階段，實現從漏洞觸發到修補驗證的全自動化流程。實驗結果顯示，本系統在多種記憶體錯誤情境下皆能生成具可編譯性與功能一致性的修補方案，整體成功修復率達 42.8%，展現出高潛力的自動化修復能力。

參考文獻

[1] Bouzenia, I., Devanbu, P., & Pradel, M. (2024). Repairagent: An autonomous, llm-based agent for program repair. *arXiv preprint arXiv:2403.17134*.

[2] Xia, C. S., & Zhang, L. (2023). Conversational automated program repair. *arXiv preprint arXiv:2301.13246*.

[3] Fan, Z., Gao, X., Mirchev, M., Roychoudhury, A., & Tan, S. H. (2023, May). Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)* (pp. 1469-1481). IEEE.

[4] Jin, M., Shahriar, S., Tufano, M., Shi, X., Lu, S., Sundaresan, N., & Svyatkovskiy, A. (2023, November). Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering* (pp. 1646-1656).

[5] Mei, X., Singaria, P. S., Del Castillo, J., Xi, H., Bao, T., Wang, R., ... & Dolan-Gavitt, B. (2024). Arvo: Atlas of reproducible vulnerabilities for open source software. *arXiv preprint arXiv:2408.02153*.