Effective mutation test generation for End-to-End test programs using large language models

Anonymous Author(s)

***Abstract*—**

***Keywords*—** Mutation Testing, LLM, Software Engineering

I. Introduction

La qualité des applications web est une préoccupation majeure dans cycle de vie d'un logiciel. Elle est souvent coûteuse et difficile à garantir. Elle est par ailleurs cruciale pour les industries sensible comme la santé ou la banque. Pour tester un logiciel on utilise entre des tests End-to-End (E2E). Ces tests sont généralement réalisés en utilisant des frameworks tels que Selenium ou Cypress [1]. Les tests End-to-End (E2E) sont une approche de test qui vise à tester le comportement d'une application web dans son ensemble selon la perspective de l'utilisateur final. De ce fait, le programme sous test (PUT) est une boite noire du point de vue du test. Cette caractéristique rends entre autres difficile l'évaluation de la qualité des tests E2E tant pendant la création que la maintenance du test. Ricca et al. [2] ont identifié plusieurs défis liés à la qualité des tests E2E parmis lesquels on retrouve le problème de la fragilité. Pour évaluer la fragilité des tests en général on utilise les tests de mutation [3]. Les tests de mutation est une technique de test qui consiste à introduire des défauts dans le code source du PUT pour évaluer la qualité des tests [4]. Cette méthode est utilisé dans l'industrie pour les tests unitaires du fait de leur caractéristique de test en boite blanche. Cependant quelques travaux commencent à s'intéresser à l'application des tests de mutation pour les tests E2E. Yandrapally et al. [5] propose dans le framework MaewU qui évalue des suites de tests d'interface utilisateur (UI). Il introduit 16 opérations de mutation basées sur 250 bug reports. Leotta et al. [6] propose par la suite l'outil MUTTA qui permet d'automatiser le processus de test de mutation des tests E2E. Ces travaux ont montré que les tests de mutation sont certes l'approche la plus pertinante pour évaluer la qualité des tests E2E, mais il subsiste plusieurs défis à relever. Parmi ces défis on retourve : (i) le coût élevé des tests de mutation, (ii) l'identification et le filtre des mutants équivalents, et (iii) l'éfficacité des tests de mutation. Dans ce travail, nous appliquons la génération des tests de mutation pour répondre à la problèmatique suivante : *Comment optimiser la génération des tests de mutation pour les tests End-to-End (E2E) en surmontant les défis du coût élevé, de l'identification des mutants équivalents, et en maximisant l'efficacité des tests pour une évaluation fiable de la qualité des suites de tests ?*

Nous proposons une approche basée sur les modèles de langage pour générer des tests de mutation plus réalistes, nécessaire et diversifiés. Notre étude est conçue pour répondre aux questions de recherche suivantes:

- **RQ1:** LLM peut-il générer des tests de mutation plus efficaces que l'approche de pointe, en s'appuyant sur l'historique des erreurs
- **RQ2:** Quel est le modele de langage le plus adapté pour la génération des tests de mutation?
- **RQ3:** Peut-on identifier et réduire le nombre de mutants équivalents générés par notre approche?
- **RQ4:** Le coût (temps d'exécution et nombre de mutants) des tests de mutation générés par notre approche est-il réduit par rapport à l'approche de l'état de l'art?

II. Background

A. Mutant Generation

A **mutant** is a small change in the source code of a PUT. The change is made to simulate a real-life defect in the code [7], [8]. The aim of mutation testing is to verify that a given test is capable of detecting the mutant by failing. This is known as killing the mutant. Mutants can be divided into two broad categories:

- **Equivalent Mutants:** Mutants that do not affect the behavior of the PUT. These mutants are not useful for testing.
- **Non-Equivalent Mutants:** Mutants that affect the behavior of the PUT. These mutants are useful for testing.

La génération de mutants est une étape cruciale dans le processus de test de mutation. Il est important de noté que la génération des mutants peut être manuelle ou automatique. Les approches manuelles nécessitent évidement une intervention humaine. Alors que les approches automatiques utilisent des outils tels que PIT [6], [9], Major [10], Jumble [11], et Javalanche [12]. En revanche, ses outils peuvent générer des mutants équivalents, ce qui peut entraîner des résultats incorrects lors de l'évaluation des tests de mutation et donc une efficacité basse. Dans un recent travail Leotta et al. [6] a utilisé PIT pour générer des mutants car l'outils met à disposition un nombre élevé des mutateurs nécessaire aux tests de mutation E2E.

B. ML in Mutation Testing

C. LLM in Mutation Testing

III. Methodology

IV. data

V. evaluation

VI. Conclusion

References

[1] M. Cerioli, M. Leotta, and F. Ricca, "What 5 million job advertisements tell us about testing: a preliminary empirical investigation," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1586–1594.

[2] F. Ricca, M. Leotta, and A. Stocco, "Three open problems in the context of e2e web testing and a vision: Neonate," in *Advances in Computers*. Elsevier, 2019, vol. 113, pp. 89–133.

[3] S. Hamimoune and B. Falah, "Mutation testing techniques: A comparative study," in *2016 international conference on engineering & MIS (ICEMIS)*. IEEE, 2016, pp. 1–9.

[4] M. R. Woodward, "Mutation testing—its origin and evolution," *Information and Software Technology*, vol. 35, no. 3, pp. 163–169, 1993.

[5] R. Yandrapally and A. Mesbah, "Mutation analysis for assessing end-to-end web tests," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 183–194.

[6] M. Leotta, D. Paparella, and F. Ricca, "Mutta: a novel tool for e2e web mutation testing," *Software Quality Journal*, vol. 32, no. 1, pp. 5–26, 2024.

[7] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," *Mutation testing for the new century*, pp. 34–44, 2001.

[8] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[9] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "Pit: a practical mutation testing tool for java," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 449–452.

[10] R. Just, "The major mutation framework: Efficient and scalable mutation analysis for java," in *Proceedings of the 2014 international symposium on software testing and analysis*, 2014, pp. 433–436.

[11] S. A. Irvine, T. Pavlinic, L. Trigg, J. G. Cleary, S. Inglis, and M. Utting, "Jumble java byte code to measure the effectiveness of unit tests," in *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007, pp. 169–175.

[12] D. Schuler and A. Zeller, "Javalanche: Efficient mutation testing for java," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 297–298.