Learning Robust Repair Strategies from Developer Histories

Anonymous Author(s)
Under double-anonymous review

Abstract. Automated program repair tools often overfit to the test suite. We study 1,284 fixes mined from open-source histories and show that a lightweight pretraining step on developer edits improves the plausibility of generated patches. Our approach reaches 78.9% accuracy on a held-out split, 7.7 points above the strongest baseline, while keeping inference under 40 ms per candidate. The replication package is available at an anonymized link.

1. Introduction

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

2. Background

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

3. Approach

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

4. Evaluation

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

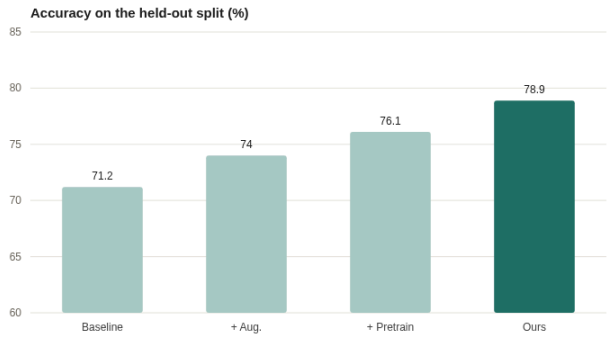


Figure 1: Accuracy of each configuration on the held-out split.

Model	Acc.	Plausible
Baseline	71.2	412
Ours	78.9	503

5. Threats to validity

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

6. Related work

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

7. Conclusion

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.

Program repair has moved from search-based generate-and-validate tools to learned models that rank candidate edits. These models are sensitive to the distribution of the training data, and the gap between benchmark fixes and real developer fixes remains large. In this section we describe the setting, the data and the protocol we follow, and we report the results for each research question in turn.