Evaluating the Impact of Software Context on the Quality of LLM-Generated Test Oracles

Hugo Klijn Mitchell Olsthoorn Annibale Panichella

TU Delft

1. Background

- **Oracle Problem:** An oracle is the mechanism used to determine the correctness of the behaviour of a System Under Test (SUT) for a given input. The oracle problem can be defined as the challenge of, given an input for a system, distinguishing corresponding desired, correct behaviour from potentially incorrect behaviour [1].
- **Mutation Testing:** a fault-based testing technique that can evaluate the quality of a test by introducing small modifications in the source code and afterwards produces a mutation adequacy score [2]. A high mutation score and test strength indicates that more mutants were killed, thus indicating a stronger test suite since the test cases are more effective at finding real faults.
- **Context Compression:** aims to reduce the amount of information provided to an LLM, while retaining enough information for the LLM to solve the desired task [3].

2. Research Question

How does context influence the quality of LLM-generated test oracles?

Based on this research question, we aim to answer the following smaller RQs:

- RQ 1 - What types of code context, individually or in combination, contribute most to the quality of LLM-generated test oracles?
- RQ 2 - What effect does the addition of documentation to code context have on the quality of LLM-generated test oracles?
- RQ 3 - To what extent can compression methods on the best-performing context configuration retain the quality of test oracles while reducing context size?

3. Methodology

We followed the pipeline based on work by D. Molinelli [4]:

1. **Dataset:** For our dataset we used gitbug-java, a publicly available and reproducible benchmark of Java bugs featuring 199 bug-fixes sourced from 55 relevant open-source repositories from 2023 [5]. Filter if: the project is a Maven project, has a single pom.xml file, and uses JUnit as its testing framework
2. **Context & Compression:** We define the following four levels of context: (TestPrefix) as T, (InvokedMethods) as I, (FocalClass) as F, (Javadoc) as J. We define the following compression methods:LLM Summary, Test Prefix + LLM Summary, Test Prefix + Skeleton
3. **Inference:** We generate the assertions using Ollama's gpt-oss:20b-cloud model, with a random seed of 42. All other unspecified decoding parameters were left unchanged from the default settings.
4. **Evaluation:** To evaluate the LLM-generated assertions and to answer our research questions, we will use a mutation testing framework for Java called PIT. The following metrics were used: compilation rate, pass rate, mutation score, test strength.
5. **Analysis:** The results will be analyzed by comparing the performance across context levels, as well as by examining the marginal contribution of each context type.

4. Results & Discussion

The results in Table 1 show that providing additional context generally improves the quality of LLM-generated assertions compared to using only the test prefix. In particular, adding the focal class leads to the strongest improvement, suggesting that it contains the most useful information for generating accurate assertions. Adding invoked methods also helps, but to a smaller extent, while combining invoked methods with the focal class does not outperform the focal class alone, likely because of overlapping or redundant information.

Table 1

Context	Comp	Pass	Mut score (Δ)	Test str (Δ)
T	68.4%	69.1%	40.4% (+0.0)	68.8% (+0.0)
T + I	69.1%	76.7%	43.7% (+3.3)	73.1% (+4.4)
T + F	76.7%	79.2%	48.2% (+7.7)	77.4% (+8.7)
T + I + F	74.2%	83.8%	45.9% (+5.5)	75.2% (+6.4)

Note. Values in parentheses indicate percentage-point changes relative to the baseline configuration T.

The results in Table 2 show that the effect of Javadoc depends on the available context: it improves performance when little code context is provided, but gives only small benefits or even reduces quality when richer context is already available.

Table 2

Context	Comp	Pass	Mut score (Δ)	Test str (Δ)
Baseline: T				
T	68.4%	69.1%	40.4% (+0.0)	68.8% (+0.0)
T + J	67.6%	75.4%	43.8% (+3.3)	73.8% (+5.1)
Baseline: T + I				
T + I	69.1%	76.7%	43.7% (+0.0)	73.1% (+0.0)
T + I + J	63.6%	77.9%	44.3% (+0.6)	73.7% (+0.6)
Baseline: T + F				
T + F	76.7%	79.2%	48.2% (+0.0)	77.4% (+0.0)
T + F + J	74.2%	82.9%	46.7% (−1.5)	73.5% (−4.0)
Baseline: T + I + F				
T + I + F	74.2%	83.8%	45.9% (+0.0)	75.2% (+0.0)
T + I + F + J	78.7%	81.2%	47.1% (+1.1)	75.8% (+0.7)

Note. Values in parentheses indicate percentage-point changes relative to the baseline configuration shown at the start of each block.

Finally, Table 3 shows that compression methods reduce token usage considerably, but all compressed configurations perform worse than the uncompressed test prefix plus focal class baseline. This indicates a trade-off between efficiency and oracle quality, where the uncompressed configuration is best when context size is not a constraint, while compressed summaries or skeletons may be useful when reducing token usage is more important.

Table 3

Context	Comp	Pass	Mut score (Δ)	Test str (Δ)	Tokens (Δ)	% Tokens	Time
T + F	76.7%	79.2%	48.2% (+0.0)	77.4% (+0.0)	1311.7 (+0.0)	100.0%	4.31s
Summary	32.4%	73.9%	29.5% (−18.7)	67.5% (−9.9)	311.2 (−1000.4)	23.7%	2.58s
T + Summary	66.8%	76.5%	41.7% (−6.5)	74.7% (−2.8)	504.0 (−807.6)	38.4%	3.35s
T + Skeleton	73.1%	65.2%	43.6% (−4.6)	72.6% (−4.9)	725.1 (−586.6)	55.3%	3.30s

Note. Values in parentheses indicate percentage-point changes relative to the baseline configuration T + F. Time is reported in seconds.

5. Conclusion

In our dataset, the results suggest that adding relevant code context improves the quality of LLM-generated assertions compared to using only the test prefix. The inclusion of the focal class contributed most among the evaluated context types. The effect of Javadoc was mixed: it improved results most whenever the available code context was limited. However, when richer code context was available, its effect was limited and even reduced assertion quality when included with the focal class. Finally, compression methods reduced the number of tokens, but did not retain the full quality of test oracles. The uncompressed configuration performed best overall. However, when context size is important, the test prefix paired with a summary provides a reasonable trade-off. But if mutation score is prioritized, the test prefix paired with the skeleton may be preferred.

6. References

[1] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The Oracle problem in software Testing: A survey. IEEE Transactions on Software Engineering, 41(5):507–525, 11 2014.

[2] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. IEEE transactions on software engineering, 37(5):649–678, 2010.

[3] Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. arXiv preprint arXiv:2307.06945, 2023.

[4] Davide Molinelli, Luca Di Grazia, Alberto Martin-Lopez, Michael D. Ernst, and Mauro Pezzè. Do LLMs Generate Useful Test Oracles? An Empirical Study with an Unbiased Dataset. Gesellschaft für Informatik (GI), 1 2026.

[5] André Silva, Nuno Saavedra, and Martin Monperrus. Gitbug-java: A reproducible benchmark of recent java bugs. In Proceedings of the 21st International Conference on Mining Software Repositories, 2024.