Property-Based Testing with Large Language Models

Chenkai Ma*
Xu Zhao*
chenkai.ma@u.nus.edu
xu.zhao@u.nus.edu
National University of Singapore
Singapore

Abstract

Property-based testing (PTB) has proven its effectiveness in software testing, but its adoption in practice is still limited, largely due to the challenges in writing diverse input generators and defining meaningful properties. Recent approaches have leveraged large language models (LLMs) to automatically generate PBTs from API documentation, but documentation alone often lacks the detail and accuracy needed to derive comprehensive properties. In this work, we propose an enhanced methodology that incorporates source code analysis alongside API documentation to address these limitations. By first focusing on deriving properties directly from source code, we try to ensure a more accurate and granular understanding of the API's behavior. Additionally, we explore multi-step prompting techniques using LLMs to generate robust property-based tests, integrating insights from both the code and documentation to enhance validity, soundness, and property coverage

CCS Concepts

• **Software and its engineering** → *Software testing and debugging*;
• **General and reference** → Design; Evaluation.

Keywords

Property-Based Testing, Large Language Models, Code Analysis

ACM Reference Format:
Chenkai Ma and Xu Zhao. 2024. Property-Based Testing with Large Language Models. In *Proceedings of CS6223 Advanced Topics in Software Testing (CS6223).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Software testing is a crucial process in software engineering as it ensures the correctness, efficiency, and security of software, so failures could be caught and addressed at an early stage[11, 13]. Automatic testing, in particular, is a prevalent method in this process because it provides consistent and efficient test inputs, unlike manual testing, which is more time-consuming and prone to human error [8]. Among automatic testing approaches, property-based testing (PTB) is a powerful method that tests the properties of a program [4], and has proven effective in identifying many bugs [2, 3, 6, 7], leading to derivative works [9, 10, 12].

Despite the success in research, PBT has not been widely adopted in practice, due to major obstacles, such as the challenges of writing random data generators and specifying meaningful properties [5]. Recently, large language models (LLMs) have become a popular and effective choice for various software testing tasks [17], making it natural to explore their potential for PBT. A promising approach is to leverage API documentation for automatically generating tests with the help of LLMs [16], as API documentation often provides detailed descriptions of inputs and desired properties, as shown in Figure 1.

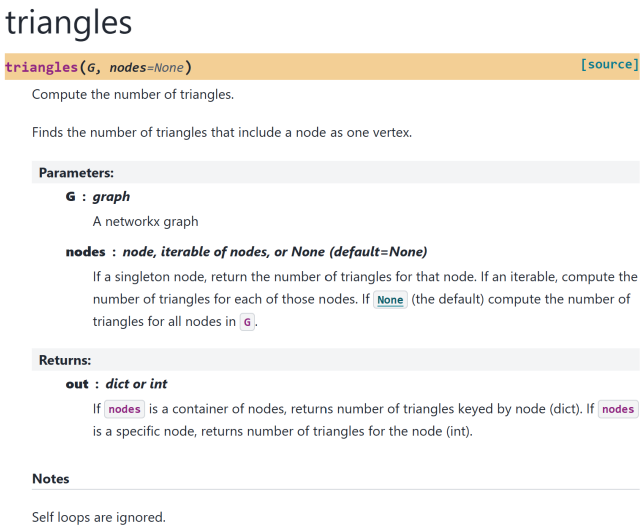

Figure 1: API documentation for `networkx.triangle`, which specifies properties of the return value.

Although leveraging large language models (LLMs) to generate property-based tests (PBT) from API documentation is a promising approach, it has certain limitations. First, API documentation may not always be detailed or accurate, lacking comprehensive information needed to extract relevant properties, such as preconditions. Secondly, documentation can sometimes be outdated or contain errors, resulting in test cases are not aligned with the actual implementation of the API. Additionally, existing methods, such as those proposed in [16], rely on mutation testing to calculate property coverage, which can result in the generation of equivalent

*Both authors contributed equally to this research.

CS6223, Aug - Dec, 2024, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

mutants when based solely on documentation. To address these shortcomings, we propose deriving properties directly from source code (Fig 2). Source code provides a more granular and accurate representation of API's behavior, offering insights that may not be captured in the documentation alone. Moreover, cross-validating properties extracted from API documentation against the source code can help us identify discrepancies or bugs. Furthermore, we will explore techniques that combine both source code and API documentation, leveraging the synergy between them. By developing techniques that incorporate both contexts, we aim to further improve the validity, soundness, and overall effectiveness of PBTs with the help of LLMs.

```python
@not_implemented_for("directed")
@nx._dispatchable
def triangles(G, nodes=None):
    if nodes is not None:
        # If `nodes` represents a single node, return only its number of triangles
        if nodes in G:
            return next(_triangles_and_degree_iter(G, nodes))[2] // 2

        # if `nodes` is a container of nodes, then return a
        # dictionary mapping node to number of triangles.
        return {v: t // 2 for v, d, t, _ in _triangles_and_degree_iter(G, nodes)}

    # if nodes is None, then compute triangles for the complete graph

    # dict used to avoid visiting the same nodes twice
    # this allows calculating/counting each triangle only once
    later_nbrs = {}

    # iterate over the nodes in a graph
    for node, neighbors in G.adjacency():
        later_nbrs[node] = {n for n in neighbors if n not in later_nbrs and n != node}

    # instantiate Counter for each node to include isolated nodes
    # add 1 to the count if a nodes neighbor's neighbor is also a neighbor
    triangle_counts = Counter(dict.fromkeys(G, 0))
    for node1, neighbors in later_nbrs.items():
        for node2 in neighbors:
            third_nodes = neighbors & later_nbrs[node2]
            m = len(third_nodes)
            triangle_counts[node1] += m
            triangle_counts[node2] += m
            triangle_counts.update(third_nodes)

    return dict(triangle_counts)
```

Figure 2: Source code for `networkx.triangle` method.

2 Proposed Solution

The workflow for our method is as follows:

(1) Construct the input for the LLM, incorporating relevant information from source code and/or API documentation.
(2) Call the LLM to generate PTBs based on the provided input.
(3) Post-process the generated PTBs, filtering out invalid or unsound test cases.
(4) Create mutants and calculate property coverage.

Due to resource constraints, we do not plan to pre-train or fine-tune LLMs. Additionally, most current LLMs are sufficiently powerful, making further tuning unnecessary. Therefore, we propose a prompting-based method for generating PTBs from API documentation and corresponding source code.

The input to the LLMs will include the following key elements:

(1) API documentation for a function.
(2) Corresponding source code for the function.
(3) Instructions for generating PTBs.

We will explore multi-step or hierarchical prompting, where the LLM first generates an intermediate representation or understanding of the API before generating the property-based tests. Additionally, we may include a system prompt instructing the LLM to act as a testing expert and a prompt specifying the output format.

3 Experiment Planning

The goal of our experiment is to address the following research questions:

(1) Can LLMs generate PTBs that are sound, valid, and has adequate property coverage based on source code?
(2) What is the synergy between API documentation and source code?
(3) How do different prompt methods affect the quality of generated PTBs?

The details of our tentative experiment setup are as follows:

- Model: We will experiment with both proprietary models, such as ChatGPT [1], Claude[1], and Gemini [14], as well as open-source models like Llama [15].
- Baseline methods: we will compare our method with the API documentation based method in [16]. Additionally, we will compare our prompting approach with their one-stage and two-stage prompt techniques.
- Data: We will use python functions from both native libraries (e.g., `datetime`) and third-party libraries (e.g., `networkx`).

Additionally, we plan to do controlled experiment to investigate the impact of different prompts, and perform ablation studies to understand the synergy between API documentation and source code.

4 Project Schedule

The tentative timeline for our project is as follows (Table 1):

Milestones	Est. Completion Time
Research proposal	6 Sep
Literature review	16 Sep
Method development (Source code)	1 Oct
Method development (API & Source code)	19 Oct
Experiment and analysis	8 Nov
Writing and revision	22 Nov

Table 1: Project Schedule

The expected outcomes of our project (contributions) are as follows:

(1) A method that prompts large language models to generate property-based tests from source code and/or API documentation.
(2) Empirical validation of the effectiveness of the proposed method.
(3) Analysis of the effects of different prompts and the synergy between API documentation and source code.

[1]https://www.anthropic.com/news/claude-3-family

Acknowledgments

We extend out sincere gratitude to Professor Manuel Rigger for his guidance and feedback throughout the project.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Thomas Arts, John Hughes, Joakim Johansson, and Ulf Wiger. 2006. Testing telecoms software with quviq QuickCheck. In *Proceedings of the 2006 ACM SIGPLAN Workshop on Erlang* (Portland, Oregon, USA) *(ERLANG '06)*. Association for Computing Machinery, New York, NY, USA, 2–10. https://doi.org/10.1145/1159789.1159792

[3] Thomas Arts, John Hughes, Ulf Norell, and Hans Svensson. 2015. Testing AUTOSAR software with QuickCheck. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 1–4. https://doi.org/10.1109/ICSTW.2015.7107466

[4] Koen Claessen and John Hughes. 2000. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*. Association for Computing Machinery, New York, NY, USA, 268–279. https://doi.org/10.1145/351240.351266

[5] Harrison Goldstein, Joseph W. Cutler, Daniel Dickstein, Benjamin C. Pierce, and Andrew Head. 2024. Property-Based Testing in Practice. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 187, 13 pages. https://doi.org/10.1145/3597503.3639581

[6] John Hughes. 2016. Experiences with QuickCheck: testing the hard stuff and staying sane. In *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*. Springer, 169–186.

[7] John Hughes, Benjamin C. Pierce, Thomas Arts, and Ulf Norell. 2016. Mysteries of DropBox: Property-Based Testing of a Distributed Synchronization Service. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 135–145. https://doi.org/10.1109/ICST.2016.37

[8] Divya Kumar and Krishn Kumar Mishra. 2016. The impacts of test automation on software's cost, quality and time to market. *Procedia Computer Science* 79 (2016), 8–15.

[9] Leonidas Lampropoulos, Michael Hicks, and Benjamin C. Pierce. 2019. Coverage guided, property based testing. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 181 (oct 2019), 29 pages. https://doi.org/10.1145/3360607

[10] David R. MacIver, Zac Hatfield-Dodds, and Many Other Contributors. 2019. Hypothesis: A new approach to property-based testing. *Journal of Open Source Software* 4, 43 (2019), 1891. https://doi.org/10.21105/joss.01891

[11] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing*. John Wiley & Sons.

[12] Rohan Padhye, Caroline Lemieux, and Koushik Sen. 2019. Jqf: Coverage-guided property-based testing in java. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 398–401.

[13] Mauro Pezzè and Michal Young. 2008. *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons.

[14] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).

[15] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[16] Vasudev Vikram, Caroline Lemieux, Joshua Sunshine, and Rohan Padhye. 2024. Can Large Language Models Write Good Property-Based Tests? arXiv:2307.04346 [cs.SE] https://arxiv.org/abs/2307.04346

[17] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing with Large Language Models: Survey, Landscape, and Vision. arXiv:2307.07221 [cs.SE] https://arxiv.org/abs/2307.07221