The Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)

QAagent: A Multiagent System for Unit Test Generation via Natural Language Pseudocode (Student Abstract)

Akhil Deo

Johns Hopkins University
3400 N. Charles St
Baltimore, MD 21218
adeo1@jhu.edu

Code — https://github.com/AkhilDeo/QAagent

Introduction

Unit testing plays a crucial role in software development, as it ensures that individual components of a system work as intended by testing them in isolation. These tests help developers catch bugs early, maintain code quality, and streamline future development by making refactoring safer and more reliable (Tosun et al. 2018). However, creating unit tests is often time-consuming, repetitive, and labor-intensive. As a result, many developers tend to de-prioritize writing comprehensive tests (Daka and Fraser 2014), leading to technical debt and decreased software reliability over time.

In recent years, the advent of large language models (LLMs) has revolutionized many aspects of software engineering, from code completion to full-scale code generation (Jiang et al. 2024). This technological shift has also highlighted the growing need for automated testing systems that can complement and aid these advances. For example, many code generation systems rely on automated test generation and execution to iteratively develop code (Huang et al. 2024a; Chen et al. 2022; Huang et al. 2024b). However, current test creation approaches lack the accuracy and code coverage to fully support developers or be a reliable indicator for code generation tasks. As software systems become more complex and developers increasingly rely on LLM-generated code, the ability to automatically generate high-quality unit tests is becoming more important than ever. This work proposes QAagent, a multiagent system that leverages LLMs' abilities to write pseudocode for functions in code to improve the coverage of LLM-generated unit tests.

Methodology

QAagent and its overall pipeline are depicted in Figure 1. The process begins by providing information, such as a function header and a comment describing its function, into the code architect agent. Once the code architect agent has generated pseudocode in natural language of how the function is most likely to be implemented, the test generator agent creates a comprehensive suite of test cases to cover basic and edge cases, predicated on the function header, the comment describing the goal of the function, and the generated pseudocode.

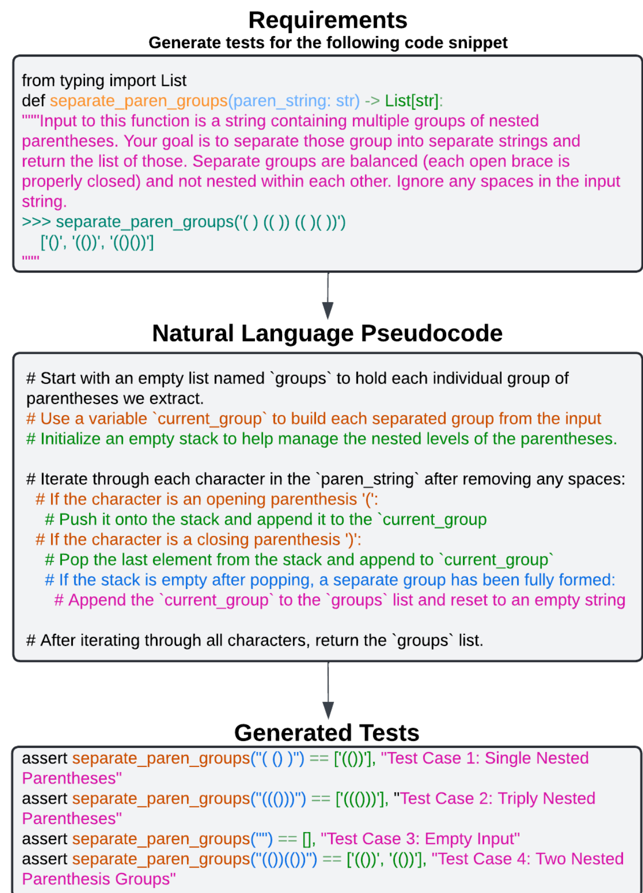

Figure 1: An Illustration of the QAagent Workflow with an Example from the HumanEval Dataset

Code Architect Agent: Natural Language Code Planning

The code architect agent employs a LLM to generate a natural language plan of how a function will be implemented. By omitting the implementation step, the system mitigates

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

the mistakes that LLMs often make during the code generation process. The agent is prompted to utilize a Chain-of-Thought approach (Wei et al. 2023) to methodically break down the task of planning how a function will be implemented into smaller, granular efforts. For example, the agent typically considers the function signature and the provided comment, and then fleshes out the idea. Following this, the LLM generates the pseudocode in natural language. The pseudocode is structured as comments to be placed into the code without issues at the test execution stage.

Test Generator Agent: General and Edge Cases

Before the test generation step, the natural language pseudocode is appended to the function header and comments describing the function's aims and any other pertinent information. The test generation agent also utilizes an LLM. The agent prompts the model to cover base and edge cases. Similarly, this agent employs a Chain-of-Thought process to break down the consideration of the various parts of the input, as well as base and edge cases, into many steps.

Evaluation

In this section, experiments are conducted in order to answer the following: What is the code coverage of the tests created by QAagent? And how accurate are the tests created by QAagent?

Experimental Setup

Benchmarks In this work, QAagent's effectiveness is evaluated with two widely used code generation datasets, HumanEval and MBPP (Chen et al. 2021; Austin et al. 2021). Though these benchmarks evaluate the effectiveness of generated code and not tests, many prior works using HumanEval and MBPP have reported test generation metrics.

Baselines We report the performance of GPT-4 Turbo, utilizing a zero-shot prompt, on the two benchmarks. Additionally, we compare the performance of QAagent against two code generation frameworks: CodeCoT and AgentCoder (Huang et al. 2024a,b).

Metrics Test suites are evaluated on line coverage and accuracy. To compute these values, we execute the generated tests on the canonical solution for each problem. The coverage results are collected by Coverage.py[1]. Line coverage is defined as the percentage of lines of code covered by the produced tests, and accuracy is the percentage of tests that pass the canonical solution.

What Is the Code Coverage of the Tests Created by QAagent?

The coverage results are shown in Table 1. We observe that the coverage from the tests generated by QAagent is 96.2% and 96.6% for HumanEval and 87.8% and 87.9% for MBPP, respectively. These results indicate that QAagent allows for better test coverage than other test generation mechanisms, for the most part. For example, AgentCoder scored 84.7% and 87.5% for HumanEval and 85.3% and 89.5% for MBPP. AgentCoder scores slightly higher on test coverage when all tests are considered for MBPP.

Strategies	**HumanEval**	**MBPP**
GPT-4 Turbo	77.0 / 79.2	35.5 / 35.9
CodeCoT	74.7 / 77.2	79.3 / 82.9
AgentCoder	84.7 / 87.5	85.3 / **89.5**
QAagent	**96.2 / 96.6**	**87.8** / 87.9

Table 1: Code Coverage of Generated Tests. We report code coverage as the percentage of code lines covered by the first five test cases / all test cases

How Accurate Are the Tests Created by QAagent?

Strategies	**HumanEval**	**MBPP**
GPT-4 Turbo	76.2	42.6
CodeCoT	67.1	79.0
AgentCoder	87.8	**89.9**
QAagent	**88.6**	52.7

Table 2: Accuracy of the Generated Test Cases

The accuracy results are shown in Table 2. We observe that the accuracy from the tests generated by QAagent is 88.6% for HumanEval and 52.7% for MBPP. Though the HumanEval accuracy score is higher than AgentCoder and CodeCoT, QAagent's accuracy on MBPP is drastically lower than the accuracies observed by AgentCoder and CodeCoT.

Conclusion

This work presents QAagent, a multi-agent system designed to improve the generation of unit tests by leveraging the capabilities of LLMs to create pseudocode for functions. Through a series of experiments conducted on the HumanEval and MBPP datasets, we demonstrate that QAagent achieves superior code coverage compared to other test generation frameworks such as AgentCoder and CodeCoT. Specifically, QAagent's tests produced significantly higher coverage on HumanEval and competitive results on MBPP.

However, while QAagent excelled in code coverage, test accuracy showed mixed results. Though QAagent's accuracy was the highest out of the three strategies on HumanEval, the observed accuracy on MBPP was notably lower compared to other approaches. This discrepancy highlights a key area for future research—enhancing the robustness of QAagent in generating accurate tests across different datasets. As the difference between HumanEval and MBPP is mainly that HumanEval provides at least one test case and a correct answer for this test case, future work could focus on determining valid answers for just one or two basic test cases without having access to anything more than a function definition and a comment describing its aims.

[1] https://coverage.readthedocs.io/en/7.6.7/

References

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; and Sutton, C. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732.

Chen, B.; Zhang, F.; Nguyen, A.; Zan, D.; Lin, Z.; Lou, J.-G.; and Chen, W. 2022. CodeT: Code Generation with Generated Tests. arXiv:2207.10397.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Daka, E.; and Fraser, G. 2014. A Survey on Unit Testing Practices and Problems. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 201–211.

Huang, D.; Bu, Q.; Qing, Y.; and Cui, H. 2024a. CodeCoT: Tackling Code Syntax Errors in CoT Reasoning for Code Generation. arXiv:2308.08784.

Huang, D.; Zhang, J. M.; Luck, M.; Bu, Q.; Qing, Y.; and Cui, H. 2024b. AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation. arXiv:2312.13010.

Jiang, J.; Wang, F.; Shen, J.; Kim, S.; and Kim, S. 2024. A Survey on Large Language Models for Code Generation. arXiv:2406.00515.

Tosun, A.; Ahmed, M.; Turhan, B.; and Juristo, N. 2018. On the effectiveness of unit tests in test-driven development. In *Proceedings of the 2018 International Conference on Software and System Process*, ICSSP '18, 113–122. New York, NY, USA: Association for Computing Machinery. ISBN 9781450364591.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.