International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Intelligent Test Automation: A Multi-Agent LLM Framework for Dynamic Test Case Generation and Validation

Pragati Kumari

Abstract

Automated software testing is essential in modern software development, ensuring stability and resilience. This study describes a unique technique for using the capabilities of Large Language Models (LLMs) via a system of autonomous agents. These agents collaborate to dynamically generate, validate, and execute test cases based on specified requirements [1, 2]. By iteratively improving test cases via agent-to-agent communication, the system improves accuracy and effectiveness. Our implementation, which uses AutoGen and Python's unittest framework, shows how this method helps to maintain excellent software quality. Experimental evaluations across a variety of test scenarios demonstrate the versatility and efficiency of our framework, **Intelligent Test Automation (ITA)**, emphasizing its promise for increasing automated software testing [3, 4].

Keywords: Intelligent Test Automation (ITA), Large Language Models (LLMs), Multi-Agent Systems, Automated Software Testing, Test Case Generation

1. Introduction

Software testing is an essential part of software development that guarantees code correctness [5], stability, and robustness against regressions. Conventional test case generation methods can be very labor-intensive and are unable to cope with evolving requirements [6, 7]. Recent developments in Large Language Models (LLMs) have created new avenues for automated test generation [8, 9]. Yet, most current frameworks depend on external dependencies, which can add complexity and introduce potential bugs [10, 11].

In this research, we present Intelligent Test Automation (ITA), a novel multi-agent system that employs LLMs to create, verify, and run test cases based purely on provided criteria. ITA guarantees ongoing test case optimization and extensive test coverage by enabling dynamic collaboration among autonomous agents [12]. This system strives to demystify the software testing process by reducing human effort while enhancing flexibility and accuracy due to evolving development needs [13].

2. Related Work

Past work in automated test generation has predominantly revolved around static test case generation, where typically predefined rules or manual input have been required [14, 15]. External knowledge sources are employed by some to augment test generation, but dependence on such leads to potential consistency and dependency issues [16].

Our strategy stands out in employing several independent agents that cooperatively create, verify, and run test cases within an isolated setting [17]. While earlier research has touched on applying Large Language

Models (LLMs) to generate tests, not many have adopted an agent-based system that repeatedly refines test cases for better correctness and responsiveness [18, 19]. By providing dynamic interaction among the agents, our **ITA** framework improves automated software testing's reliability and versatility [20].

3. Proposed Framework

The framework, ITA, proposed in this work has three autonomous specialist agents that together produce, polish, and validate test cases for extensive software testing. The execution flow of ITA is presented in figure 1 and is as follows:

- **Test Case Generation Agent (TCGA):** The agent is responsible for extracting requirements from the specification document (**REQ_SPE.pdf**) and creating initial test cases (**test_cases.py**). It also creates a preliminary program (**calculator.py**) from the given requirements.
- **Test Case Modification Agent (TCMA):** After the initial test cases and program are created, TCMA fine-tunes the test cases and, if needed, adjusts the program to correct any inconsistencies. The revised test cases (**modified_test_cases.py**) and the modified program (**modified_calculator.py**) are then run to ensure the correctness of the implementation.
- **Test Case Validation Agent (TCVA):** TCVA validates the updated test cases and confirms that they meet the software requirements. It checks whether the test cases adequately cover the anticipated functionality and generates a final validation report certifying the test results.
- These agents operate in an iterative process, constantly updating the test cases and the program until the validation requirements are fulfilled. Automating this process reduces manual intervention to a minimum, increases test precision, and increases adaptability to changing requirements.

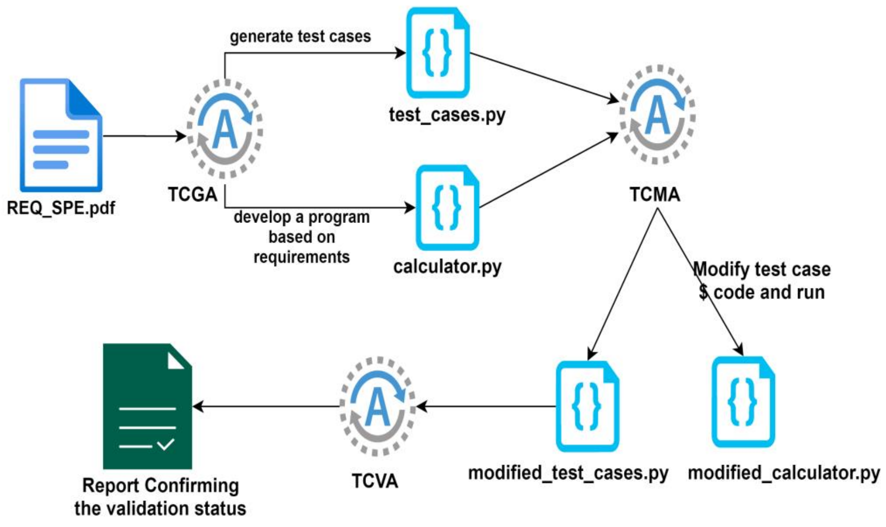

Fig 1: Execution flow of the proposed framework

4. Experimental Setup and Evaluation

To install **ITA**, first make sure Python version 3.8 or later is installed and then install using pip. Set up the API endpoint by creating environment variables or a JSON file. For the Large Language Model (LLM), set parameters such as configuration list, temperature, timeout, and cache seed. For interactions, instantiate AssistantAgent and UserProxyAgent. Utilize the Software Tester Agent (STA) to initiate conversations and create test cases from the "REQ_SPE.pdf" file. Go ahead and implement the program based on these specifications. Run test cases subsequently using Python's unittest module, with the level of verbosity changed as necessary. Finish setting up by checking the test cases using the Test Case Validation Agent (TCVA) to verify they correctly identify pass or fail situations.

The framework was tested on several software requirements to show its ability to:

- Automatically extract requirements from a structured document.
- Generate appropriate test cases from given requirements only.
- Iteratively adjust test cases to improve coverage and precision.
- Dynamically adjust code according to validation feedback.
- Run and validate test cases automatically.

High test coverage and adaptability are indicated by results, demonstrating the practical usability of **ITA** in automated software testing.

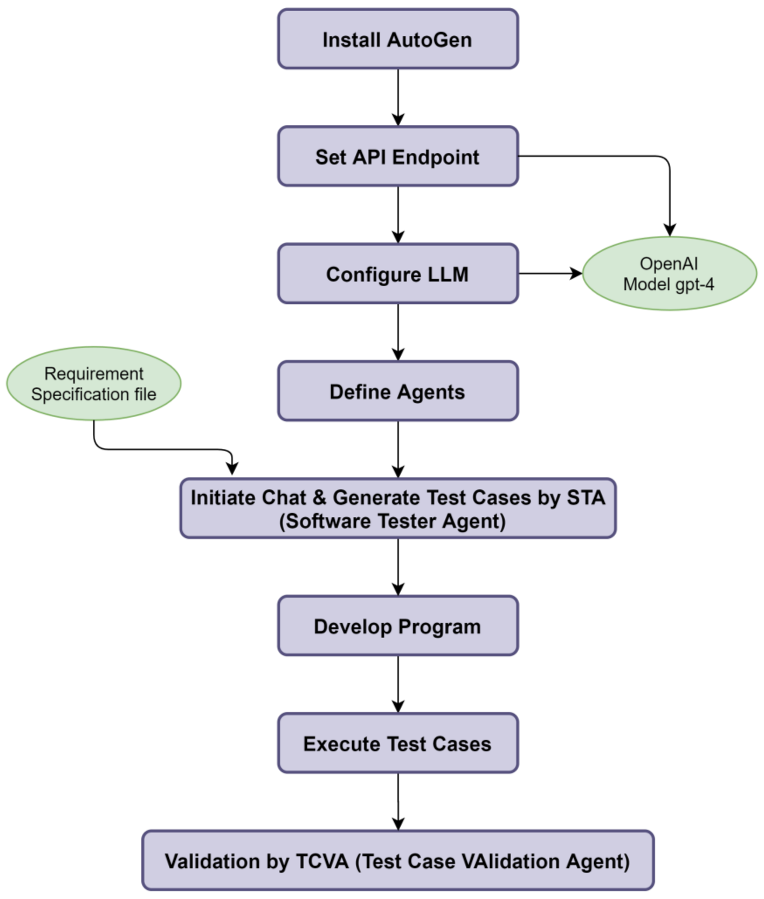
Install AutoGen
Set API Endpoint
Configure LLM
OpenAI Model gpt-4
Requirement Specification file
Define Agents
Initiate Chat & Generate Test Cases by STA (Software Tester Agent)
Develop Program
Execute Test Cases
Validation by TCVA (Test Case VAlidation Agent)

Fig. 2: Flow Diagram of Intelligent Test Automation

5. Discussion

The suggested framework has some very important benefits that lead to the efficiency, accuracy, and flexibility of software testing in an automated manner.

- **Automated Requirement Processing:** The primary goal of this method lies in the fact that it can automatically interpret and extract software requirements from specification documents. By avoiding the creation of test cases manually, the framework lowers the level of human intervention, reduces error rates, and speeds up testing. This automation is especially useful in large projects or commonly changing projects where the generation of test cases manually would prove to be time-consuming and error-ridden.
- **High Adaptability:** Contrary to the conventional testing methodologies, which may find it challenging to cope with changing software specifications, this methodology utilizes autonomous agents that continuously optimize test cases. The agents adaptively examine and modify test cases in response to validation outcomes so that they constantly remain in line with the new specifications. Such adaptability adds strength to the software testing process, making it especially suitable for agile development scenarios.
- **Test Case and Code Refinement:** The model extends basic test generation by having an iterative refinement process. The **Test Case Modification Agent (TCMA)** actively analyzes test cases, performing required modifications on both the test scripts and the implementation of the software. Through this ongoing process, edge cases are discovered, test accuracy is enhanced, and overall software quality is improved, lowering the chance of hidden bugs in production.
- **Comprehensive Execution Flow:** The systematic process of execution guarantees that all test cases created cover the software's functional requirements comprehensively. **Test Case Validation Agent (TCVA)** is vital in determining the validity and completeness of test cases prior to final validation. With the systematic process, gaps in test coverage are reduced, and confidence in the reliability of the software is improved.

6. Limitations and Future Work

Although the framework offers benefits, it does pose certain issues that should be resolved:

- **Computational Overhead:** As accuracy is improved through iterative refinement, computational overhead increases. Frequent changes in test cases and program code [21] are time- and resource-consuming processes, which have the potential to affect performance, particularly in mass-scale software programs [22]. Future research will aim at streamlining the process by achieving more efficient agent coordination and caching intelligence to mitigate redundant computations.
- **Dependence on Initial Requirement Clarity:** As the framework is based on requirement extraction, errors or vagueness in the specification document can result in poor-quality test cases. Increasing the reliability of requirement interpretation—perhaps using more sophisticated natural language processing (NLP) methods—would help to overcome this difficulty and enhance the quality of test cases produced [23].

- **Scalability Considerations:** The present framework behaves well in highly controlled test cases, but it is yet to be proven for use in realistic, multi-module software systems. Parallelizing operations among agents and the integration of the framework within CI/CD pipelines will be future work toward achieving scalability and practical applicability [24].

Overall, the proposed framework provides a fully automated, adaptive, and optimal method of software testing, correcting most of the shortcomings of classical approaches. While computational efficiency and scalability are the areas that require improvement, the continuous research and optimizations will extend its applicability to real-world scenarios [25].

7. Conclusion

This work presents a new multi-agent, LLM-based system for automated test generation and execution that offers a highly adaptive and efficient method of software testing. Through the use of interactive collaboration between autonomous agents, the system ensures higher accuracy in test case generation with less human intervention. The iterative refinement process enables ongoing improvements in test coverage and software correctness, ultimately leading to increased software reliability.

The outcomes prove that this method not only simplifies the testing procedure but also maximizes its efficacy in dynamic and changing development environments. By minimizing reliance on fixed test case generation and outside knowledge bases, the framework creates a more independent and extendable testing model. These results point to the potential of multi-agent LLM-based systems in revolutionizing conventional software testing methodologies and propelling automation in quality assurance.

Appendix

A. Experimental Setup and Code Implementation

This section provides technical details about setting up **Intelligent Test Automation** using **AutoGen**, API configurations, multi-agent interactions, and automated test execution.

A.1. AutoGen Installation and API Configuration

- **Python Requirement:** AutoGen requires Python **3.8 or higher**. Install it using:
  pip install autogen
- **Setting API Endpoint:**
  - AutoGen loads configurations from an **environment variable** or a **JSON file** using config_list_from_json().
  - It first checks the environment variable **"OAI_CONFIG_LIST"**, which must contain a **valid JSON string**.
  - If missing, it searches for a JSON file with the same name.
  - Only **GPT-4** models are retained for execution.

A.2. LLM Configuration Parameters

A dictionary called **LLM Config** contains several parameters that need to be set for a model, including:

- **config_list** → Additional model settings.
- **temperature** → Controls randomness in responses.
- **timeout** → Maximum wait time for a model response.
- **cache_seed** → Used for caching consistency.

A.3. Multi-Agent Configuration

The framework defines two agents:

- **AssistantAgent (assistant):** Interacts with the user and processes tasks.
- **UserProxyAgent (user_proxy):** Handles user input and task management.

Agent behavior includes:

- **Data input via user_proxy**, with a **TERMINATE** message upon task completion.
- A maximum of **10 consecutive auto-replies** is allowed.
- The **working directory** is set to "web" via code_execution_config.
- **Language modeling parameters** are configured for both agents.

A.4. Test Case Generation Process

- A **chat session** is initiated.
- The **Software Tester Agent (STA)** reads the **"REQ_SPE.pdf"** file and extracts the **requirements**.
- A structured **prompt** is generated for the **STA** to create **test cases**.
- The system **automatically saves** the test cases in a structured file.

A.5. Program Development and Execution

- The **AssistantAgent** outlines **step-by-step actions** for developing a program based on requirements.
- The user executes the program and **runs test cases** using Python's unittest module.
- **If test cases fail**, the system **automatically modifies** the program and test cases.
- If all test cases pass, no modification is required.

A.6. Test Case Validation

- The **Test Case Validation Agent (TCVA)** checks the **pass/fail** status of test cases.
- If failures occur, the agent suggests modifications.

B. Full Implementation Code Repository

For complete implementation details, including the **Python implementation** of Intelligent Test Automation, visit the **GitHub repository**:

➞ https://github.com/Kumari-Pragati/Intelligent-Test-Automation

This repository includes:

- The **Python implementation** of Intelligent Test Automation.
- Sample **input requirement documents** and **generated test cases**.
- Automated **test execution scripts**.

Acknowledgement

I would like to express my sincere gratitude to Dr. Novarun Deb, Assistant Professor at the University of Calgary, Canada for his invaluable guidance and support. I am honored to begin my PhD under his supervision at the University of Calgary, Canada. I would also like to acknowledge Dr. Souvick Das, Research Associate at Ca' Foscari University, Venice, Italy, for his insightful discussions and continuous encouragement in my research journey. Their mentorship and support have been instrumental in shaping my academic and research pursuits.

Authors' Biography

Pragati Kumari received an MTech degree in Information Architect and Software Engineering from School of Computer Science & Information Technology, Devi Ahilya Vishwavidyalaya, Indore in 2017. She worked as a Project Associate at IIIT Vadodara from Oct 2022 to July 2024, contributing to an ISRO-sponsored project on Safe Ship Navigation. Additionally, she served as a Teaching Assistant from Jan 2022 to Dec 2022, assisting in courses related to Software Engineering and Database Management Systems. She has been admitted to the PhD program at the University of Calgary, Canada, for the May 2025 session. Her research interests include software testing automation, requirement engineering, artificial intelligence, large language models, and multi-agent systems. She is currently engaged in independent research on intelligent test automation using LLMs.

References

1. S. C. Allala, J. P. Sotomayor, D. Santiago, T. M. King, and P. J. Clarke, “Towards transforming user requirements to test cases using MDE and NLP,” in *Proceedings - International Computer Software and Applications Conference*, IEEE Computer Society, Jul. 2019, pp. 350–355. doi: 10.1109/COMPSAC.2019.10231.
2. M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, “An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation,” Feb. 2023, [Online]. Available: http://arxiv.org/abs/2302.06527
3. J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, “Software Testing With Large Language Models: Survey, Landscape, and Vision,” *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 911–936, Apr. 2024, doi: 10.1109/TSE.2024.3368208.
4. H. Ayenew and M. Wagaw, “Software Test Case Generation Using Natural Language Processing (NLP): A Systematic Literature Review,” *Artificial Intelligence Evolution*, pp. 1–10, Jan. 2024, doi: 10.37256/aie.5120243220.
5. Y. Cai *et al.*, “Automated Program Refinement: Guide and Verify Code Large Language Model with Refinement Calculus,” *Proceedings of the ACM on Programming Languages*, vol. 9, no. POPL, pp. 2057–2089, Jan. 2025, doi: 10.1145/3704905.

6. N. Marques, R. R. Silva, and J. Bernardino, "Using ChatGPT in Software Requirements Engineering: A Comprehensive Review," Jun. 01, 2024, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/fi16060180.
7. N. S. Mathews and M. Nagappan, "Test-Driven Development and LLM-based Code Generation," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, New York, NY, USA: ACM, Oct. 2024, pp. 1583–1594. doi: 10.1145/3691620.3695527.
8. Z. Xue *et al.*, "LLM4Fin: Fully Automating LLM-Powered Test Case Generation for FinTech Software Acceptance Testing," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA: ACM, Sep. 2024, pp. 1643–1655. doi: 10.1145/3650212.3680388.
9. S. Wang, Y. Yu, R. Feldt, and D. Parthasarathy, "Automating a Complete Software Test Process Using LLMs: An Automotive Case Study," Feb. 2025.
10. Alex, Liu, Vivian, and Chi, "From Defects to Demands: A Unified, Iterative, and Heuristically Guided LLM-Based Framework for Automated Software Repair and Requirement Realization," Dec. 2024, [Online]. Available: http://arxiv.org/abs/2412.05098
11. S. M. Taghavi Far and F. Feyzi, "Large language models for software vulnerability detection: a guide for researchers on models, methods, techniques, datasets, and metrics," *Int J Inf Secur*, vol. 24, no. 2, p. 78, Apr. 2025, doi: 10.1007/s10207-025-00992-7.
12. M. Haeggström, "Hands-on Use and Adaptation of AI in Developing and Testing Software Applications."
13. H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future," Aug. 2024.
14. J. A. S. de Cerqueira, M. Agbese, R. Rousi, N. Xi, J. Hamari, and P. Abrahamsson, "Can We Trust AI Agents? An Experimental Study Towards Trustworthy LLM-Based Multi-Agent Systems for AI Ethics," Oct. 2024, [Online]. Available: http://arxiv.org/abs/2411.08881
15. D. Cambaz and X. Zhang, "Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, New York, NY, USA: ACM, Mar. 2024, pp. 172–178. doi: 10.1145/3626252.3630958.
16. Q. Wang, J. Wang, M. Li, Y. Wang, and Z. Liu, "A Roadmap for Software Testing in Open-Collaborative and AI-Powered Era," *ACM Transactions on Software Engineering and Methodology*, Dec. 2024, doi: 10.1145/3709355.
17. S. Khan and M. Daviglus, "AI-Driven Automation in Agile Development: Multi-Agent LLMs for Software Engineering," 2025, doi: 10.13140/RG.2.2.20682.89281.
18. S. Das, N. Deb, A. Cortesi, and N. Chaki, "Extracting goal models from natural language requirement specifications," *Journal of Systems and Software*, vol. 211, May 2024, doi: 10.1016/j.jss.2024.111981.
19. E. Jabbar, S. Zangeneh, H. Hemmati, and R. Feldt, "Test2Vec: An Execution Trace Embedding for Test Case Prioritization," Jun. 2022, [Online]. Available: http://arxiv.org/abs/2206.15428
20. M. Boukhlif, N. Kharmoum, and M. Hanine, "LLMs for Intelligent Software Testing: A Comparative Study," in *Proceedings of the 7th International Conference on Networking, Intelligent*

Systems and Security, New York, NY, USA: ACM, Apr. 2024, pp. 1–8. doi: 10.1145/3659677.3659749.

21. M. Tufano, A. Agarwal, J. Jang, R. Z. Moghaddam, and N. Sundaresan, "AutoDev: Automated AI-Driven Development," Mar. 2024.
22. C. Cao, F. Wang, L. Lindley, and Z. Wang, "Managing Linux servers with LLM-based AI agents: An empirical evaluation with GPT4," *Machine Learning with Applications*, vol. 17, p. 100570, Sep. 2024, doi: 10.1016/j.mlwa.2024.100570.
23. A. Abo-eleneen, A. Palliyali, and C. Catal, "The role of Reinforcement Learning in software testing," Dec. 01, 2023, *Elsevier B.V.* doi: 10.1016/j.infsof.2023.107325.
24. N. Rao, K. Jain, U. Alon, C. Le Goues, and V. J. Hellendoorn, "CAT-LM Training Language Models on Aligned Code And Tests," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, Sep. 2023, pp. 409–420. doi: 10.1109/ASE56229.2023.00193.
25. H. Yin, H. Mohammed, and S. Boyapati, "Leveraging Pre-Trained Large Language Models (LLMs) for On-Premises Comprehensive Automated Test Case Generation: An Empirical Study," in *2024 9th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, IEEE, Nov. 2024, pp. 597–607. doi: 10.1109/ICIIBMS62405.2024.10792720.