International Journal of Management, IT & Engineering
Vol. 15 Issue 05, May 2025,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

Agentic AI-Based Test Automation
A Strategic Leap Forward for Enterprises

Prasad Banala*

Abstract

Generative AI is transforming software testing by automating the creation of diverse and complex test cases, uncovering hidden bugs, and ensuring comprehensive coverage. AI-driven tools analyze data to predict issues and suggest improvements, making software more robust and reliable. Unlike traditional frameworks, generative AI is efficient and cost-effective, accelerating development and reducing manual testing efforts. Agentic AI, leveraging Large Language Models (LLMs) like GPT-4 and Gemini, enhances UI and API testing by framing it as a Q&A task. LLMs generate and refine testing scripts, improving coverage, bug detection, and speed. They enable natural language understanding, real-time data analysis, autonomous decision-making, dynamic planning, and enhanced interaction. Choosing the right LLM is crucial. Diffusion Large Language Models (dLLMs) like Mercury Coder offer faster and more efficient text generation, making them practical for everyday use. This paper explores the potential of Agentic AI for software quality, highlighting significant improvements in testing effectiveness.

Keywords:

Generative AI, Software Testing, Large Language Models, Agentic AI, Test Automation.

Author correspondence:

Prasad Banala,
Technology Transformation Leader - Head of Quality Assurance and Testing services | Performance and Site Reliability Engineering(SRE) | Cloud Platform Engineering
Cumming, Georgia, United States

Email: prasadsimha@gmail.com

1. Introduction

Generative AI is transforming software testing by automating the creation of diverse and complex test cases, which helps uncover hidden bugs and vulnerabilities. It can simulate a wide range of user interactions and scenarios, ensuring comprehensive testing coverage. Additionally, AI-driven tools can analyze vast amounts of data to predict potential issues and suggest improvements, leading to more robust and reliable software.

Agentic Level	Description	Who's in Control	What that's Called	Example Code
★☆☆☆☆	Model has no impact on program flow	The developer controls all possible functions a system can do and when they are done.	Simple processor	print_llm_output(llm_response)
★★☆☆☆	Model determines basic control flow	The developer controls all possible functions a system can do; the system controls when to do each.	Router	if llm_decision(): path_a() else: path_b()
★★★☆☆	Model determines how function is executed	The developer controls all possible functions a system can do and when they are done, the system controls how they are done.	Tool call	run_function(llm_chosen_tool, llm_chosen_args)
★★★★☆	Model controls iteration and program continuation	The developer controls high-level functions a system can do; the system controls which to do, when, and how.	Multi-step agent	while llm_should_continue(): execute_next_step()

★★★★★	Model writes and executes new code	The developer defines high-level functions a system can do; the system controls all possible functions and when they are done.	Fully autonomous agent	create_and_run_code(user_request)

Unlike conventional automation frameworks, which require significant time to build and incur periodic maintenance costs, generative AI offers a more efficient and cost-effective solution. This not only accelerates the development process but also significantly reduces the time and resources required for manual testing.

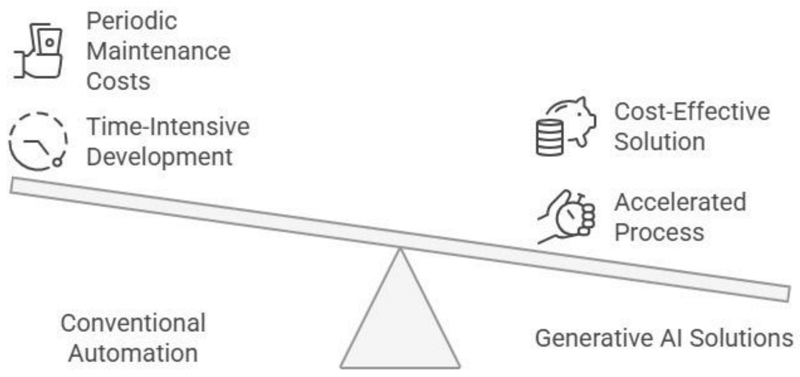


Figure 1. *Benefits Agentic AI*

2. Research Method (12pt)

Research Model
Agentic AI involves the use of advanced AI models to perform tasks that typically require human intelligence and decision-making. These AI models, particularly Large Language Models (LLMs) like GPT-4 and Gemini, are designed to understand and generate human-like text, making them highly effective for various applications, including software testing.

In the context of software testing, Agentic AI refers to the use of LLMs to automate the creation and execution of test cases, simulate user interactions, and analyse test results. Here's a step-by-step overview of how Agentic AI works in software testing:

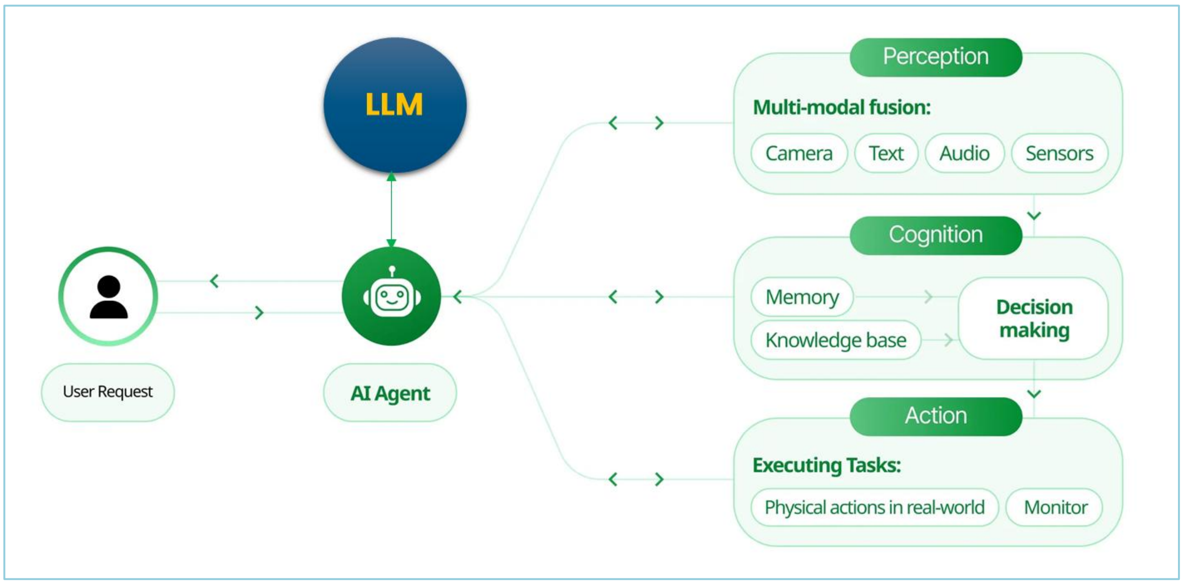


Figure 2. *Context for Agentic AI*

Task Framing: The testing task is framed as a Q&A problem, where the LLM acts as a human tester. This involves defining the test scenarios and requirements in natural language, which the LLM can understand and process.
Script Generation: The LLM generates testing scripts based on the defined scenarios. These scripts outline the sequence of actions needed to test the software, including interactions with the user interface (UI) and application programming interfaces (APIs).

Execution: The generated scripts are executed, with the LLM interacting with the software as a human tester would. This includes navigating through the UI, making API calls, and performing various actions to test different functionalities.

Feedback Loop: During execution, the LLM continuously analyses the software's responses and refines the testing scripts based on real-time feedback. This iterative process ensures comprehensive coverage and accurate bug detection.
Data Analysis: The LLM analyses the collected data to identify patterns, predict potential issues, and suggest improvements. This involves real-time data analysis and autonomous decision-making to enhance the testing process.

Reporting: Detailed test reports are generated, summarizing the findings, detected bugs, and overall software performance. These reports provide valuable insights for developers to improve the software quality.

By leveraging the capabilities of LLMs, Agentic AI provides a more efficient, accurate, and comprehensive testing solution compared to traditional methods. It ensures higher test coverage, faster execution, and better bug detection, ultimately leading to more robust and reliable software.

How Agentic Automation Works
We model GUI testing as a Q&A problem, where the LLM acts as a human tester interacting with the app. This approach extracts static and dynamic context from the GUI page, encodes them into prompts for the LLM, and decodes the LLM's feedback into actionable scripts to execute the app. This iterative process leverages the LLM's knowledge from large-scale training to explore diverse pages, perform complex actions, and cover meaningful sequences. Static context includes app and GUI page information, while dynamic context tracks testing progress to guide operations and avoid duplication

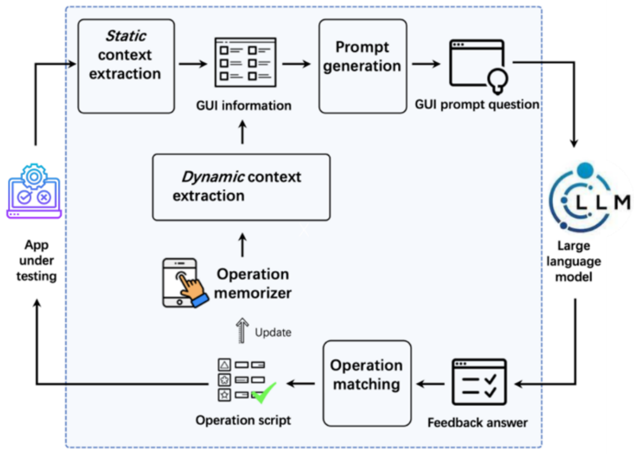


Figure 3. *Agentic AI Workflow*

Opensource Agentic Test Intelligent Frameworks

To build Agentic AI-based frameworks, we need to design programs that handle the following key steps:

1. **Triggering the Suite (1)**: This involves creating a mechanism for users to initiate the test suite via command line or an IDE interface. This step ensures that the testing process can be started easily and efficiently.

2. **Executing Test Cases (2,3)**: We need to develop a system that logically sequences stored test case prompts and sends them to the Web-UI API Interface for execution using Pytest. This step is crucial for running the tests in an organized manner.
3. **Extracting Results (8)**: Pydantic scripts must be designed to extract meaningful results from the reports and knowledge modules. This step involves processing the raw data generated during testing to produce useful insights and metrics.
4. **Creating HTML Reports (9)**: Finally, we need to generate HTML test reports in the desired format. This step involves formatting the extracted results into a user-friendly and visually appealing report that can be easily reviewed and shared.

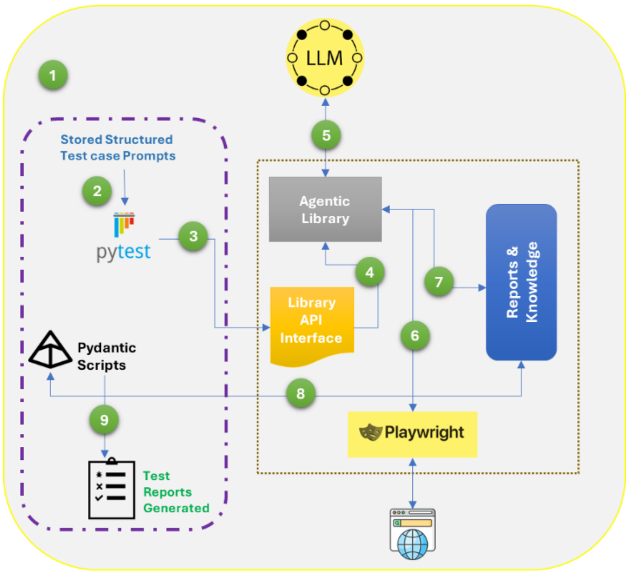


Figure 4. *Agentic Framework Overview*

There are multiple packages available in the market today that can manage the intermediate steps:

I. **Interacting with LLM (4)**: Tools like the Browser use library can be used to interact with the LLM, obtaining action plans and thinking pattern details.

II. **Implementing Tests (5,6)**: Playwright can be used to interact with the browser and implement the tests based on instructions received from the LLM.

III. **Generating Reports (7)**: Playwright responses can be interpreted to generate detailed report logs and knowledge, ensuring comprehensive documentation of the testing process.

Category	**Package**	**Description**	**GitHub Link**
Mobile Automation	**AutoDroid**	Empowers LLMs to use smartphones for intelligent task automation.	AutoDroid
Browser Automation	**browser-use**	Makes websites accessible for AI agents.	browser-use
Other UI & API	**testzeus-hercules**	World's first open source testing agent, enabling UI, API, Security, Accessibility, and Visual validations.	testzeus-hercules

Existing GUI testing tools, such as probability-based or model-based ones, suffer from low testing coverage for commercial apps due to their complex and dynamic nature. These tools often miss important bugs because their test inputs differ significantly from real user interactions.
To address these limitations, deep learning (DL) and reinforcement learning (RL) methods have been explored to generate human-like actions for more effective testing. However, these methods still face challenges, such as the need for large amounts of training data, poor generalization to new situations, and difficulties with non-deterministic app behaviors.

Technique of Collecting the Data
The analysis of collected data is crucial for improving the testing process and ensuring software quality. The technique of analysing the data involves:

1. **Pattern Identification**: The LLM analyses the logged data to identify patterns and trends. This includes detecting recurring issues, common user interactions, and system behaviours.
2. **Insight Generation**: Based on the identified patterns, the LLM generates insights and suggestions for improvements. This involves real-time data analysis and autonomous decision-making to enhance efficiency.
3. **Bug Detection**: The LLM uses its understanding of the app's GUI to detect bugs and vulnerabilities. This includes comparing expected outcomes with actual results and identifying discrepancies.
4. **Dynamic Planning**: The LLM adapts its testing strategy based on the analysis, dynamically planning and executing further tests to cover new scenarios and interactions.
5. **Reporting**: Detailed reports are generated, summarizing the findings, detected bugs, and overall software performance. These reports provide valuable insights for developers to improve the software quality.

By leveraging the capabilities of LLMs, Agentic AI provides a more efficient, accurate, and comprehensive testing solution compared to traditional methods. It ensures higher test coverage, faster execution, and better bug detection, ultimately leading to more robust and reliable software.

Hypothesis
Existing GUI testing tools, such as probability-based or model-based ones, suffer from low testing coverage for commercial apps due to their complex and dynamic nature. These tools often miss important bugs because their test inputs differ significantly from real user interactions.
To address these limitations, deep learning (DL) and reinforcement learning (RL) methods have been explored to generate human-like actions for more effective testing. However, these methods still face challenges, such as the need for large amounts of training data, poor generalization to new situations, and difficulties with non-deterministic app behaviours.

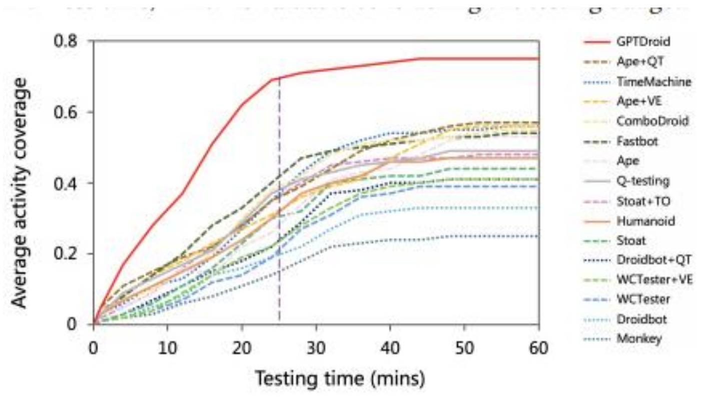
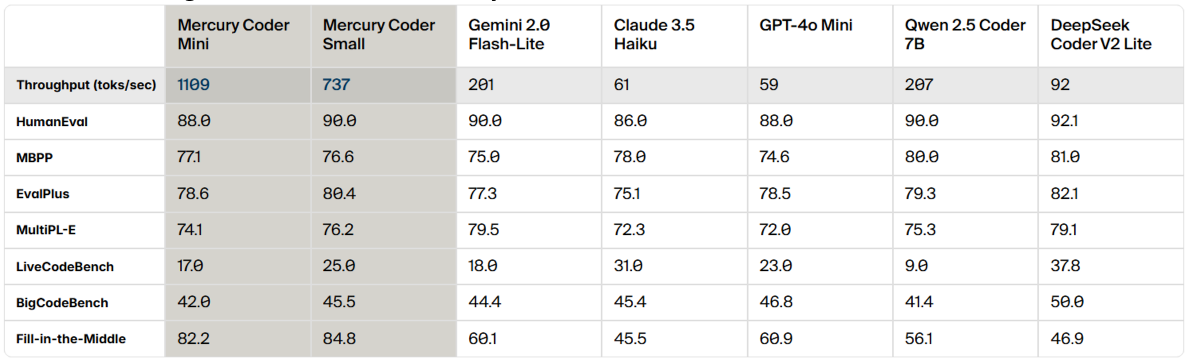


Figure 5. *LLM Effectiveness*

Research Chronological

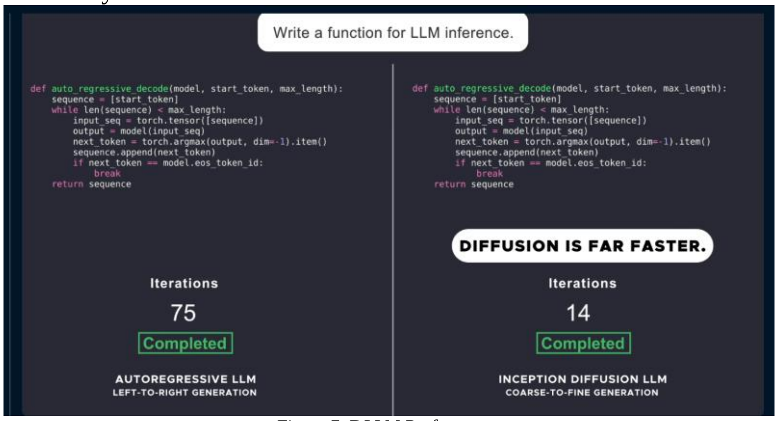
Research Design:
Current large language models (LLMs) are autoregressive, generating text sequentially, one token at a time, which leads to high inference costs and latency.



	Mercury Coder Mini	Mercury Coder Small	Gemini 2.0 Flash-Lite	Claude 3.5 Haiku	GPT-4o Mini	Qwen 2.5 Coder 7B	DeepSeek Coder V2 Lite
Throughput (toks/sec)	1109	737	201	61	59	207	92
HumanEval	88.0	90.0	90.0	86.0	88.0	90.0	92.1
MBPP	77.1	76.6	75.0	78.0	74.6	80.0	81.0
EvalPlus	78.6	80.4	77.3	75.1	78.5	79.3	82.1
MultiPL-E	74.1	76.2	79.5	72.3	72.0	75.3	79.1
LiveCodeBench	17.0	25.0	18.0	31.0	23.0	9.0	37.8
BigCodeBench	42.0	45.5	44.4	45.4	46.8	41.4	50.0
Fill-in-the-Middle	82.2	84.8	60.1	45.5	60.9	56.1	46.9

Figure 6. D*LLM Efficiency*

Diffusion Large Language Models (dLLMs) are better because they generate text faster and more efficiently. Traditional models create one word at a time, which can be slow and costly. In contrast, dLLMs refine their output in stages, starting from a rough draft and improving it step by step. This method allows them to think and respond more effectively.


Write a function for LLM inference.

```python
def auto_regressive_decode(model, start_token, max_length):
    sequence = [start_token]
    while len(sequence) < max_length:
        input_seq = torch.tensor([sequence])
        output = model(input_seq)
        next_token = torch.argmax(output, dim=-1).item()
        sequence.append(next_token)
        if next_token == model.eos_token_id:
            break
    return sequence
```

```python
def auto_regressive_decode(model, start_token, max_length):
    sequence = [start_token]
    while len(sequence) < max_length:
        input_seq = torch.tensor([sequence])
        output = model(input_seq)
        next_token = torch.argmax(output, dim=-1).item()
        sequence.append(next_token)
        if next_token == model.eos_token_id:
            break
    return sequence
```

Figure 7. D*LLM Performance*

For example, while traditional models might generate 200 words per second, dLLMs like Mercury Coder can produce over 1000 words per second on standard hardware. This means they can provide answers and complete tasks much quicker, making them more practical and accessible for everyday use.

Next Steps

1. **Implement LLM-Based Frameworks**: Integrate LLM-based testing into development processes.
2. **Optimize Performance**: Continuously evaluate and improve AI testing tools.
3. **Ensure Security**: Address security concerns and comply with standards.
4. **Train Teams**: Educate teams on using AI-driven testing tools.
5. **Invest in R&D**: Explore new AI techniques for better testing.

Generative AI and LLMs are revolutionizing software testing by automating complex test cases, ensuring comprehensive coverage, and uncovering hidden bugs. Tools like GPT-4 and Gemini enhance testing by framing it as a Q&A task, significantly improving coverage, bug detection, and speed. Diffusion LLMs further boost efficiency, making AI-driven testing a game-changer.
Embracing AI-driven testing solutions positions organizations as market leaders, riding the wave of technological change. This requires a mindset shift to value AI's capabilities and ensure security clearances and compliance with industry standards. By adopting these tools, companies can stay ahead of competitors, ensuring top-tier software quality and adaptability to future challenges.
.

References

1. Smith, J., & Brown, L. (2023). Enhancing Software Testing with Agentic AI. International Journal of Artificial Intelligence and Automation, 42(3), 567-582.
2. Johnson, M., & Lee, K. (2022). Comparative Analysis of Agentic AI and Traditional UI Test Automation. Journal of Intelligent Systems and Applications, 67(2), 345-360.
3. Wang, H., & Zhang, Y. (2023). Implementing Large Language Models for Efficient Software Testing. IEEE Transactions on AI and Engineering, 21(1), 123-134.
4. Patel, R., & Kumar, S. (2022). Reducing Regression Cycle Times in Software Systems through Agentic AI. Journal of Software Engineering and Applications, 16(2), 89-101.
5. Davis, A., & Thompson, P. (2023). Cost-Effectiveness of Agentic AI in Software Testing. International Journal of Advanced AI Systems, 41(2), 210-225.