Report: AI in Software Testing

Artificial Intelligence (AI) is increasingly being adopted in software testing to enhance efficiency, accuracy, and scalability. AI tools can assist with test case generation, data creation, bug detection, execution, and reporting. This report outlines how AI helps in software testing, the tools available both online (cloud-based) and offline (local LLMs), and the practical challenges faced when integrating AI with modern testing protocols such as the Model Context Protocol (MCP).

1. How AI Helps in Software Testing

1. Automated Test Case Generation: AI models can create unit, integration, and end-to-end test cases.
2. Test Data Generation: AI can generate realistic and edge-case data to enhance coverage.
3. Bug Detection: AI-powered static code analysis tools can identify potential vulnerabilities.
4. Natural Language Processing: Converting requirements or user stories directly into test scenarios.
5. Visual Testing: AI can detect UI discrepancies automatically.
6. Flowchart and Documentation: Generating clear visual representations of test flows.

2. Online Cloud-based AI Tools

Cloud-based AI solutions provide access to powerful LLMs without requiring local hardware. Commonly used tools include: - ChatGPT, Gemini, Claude: For test case generation, data preparation, and flowcharts. - GitHub Copilot: AI pair programming for creating unit tests. - AI-assisted platforms like Testim, Functionize, and Mabl for autonomous testing.

3. Offline / Local AI Tools

Running AI locally gives more control and privacy but requires significant hardware resources. For effective MCP integration, at least a 7B parameter LLM is recommended, which needs ~16GB RAM with GPU support. With 8GB RAM, smaller models can be tested but may not handle complex workflows efficiently. Examples: - Ollama + Local LLMs (e.g., LLaMA, Mistral) - LM Studio for running models offline - Playwright MCP for automated browser testing connected with LLMs

4. Challenges and Limitations

1. Execution Overhead: AI-driven execution of test cases can be slower than manual runs.
2. Hardware Limitations: Running a 7B+ model locally requires at least 16GB RAM and GPU.
3. Integration Complexity: Setting up MCP and bridging AI with tools like Playwright is time-consuming.
4. Cost: Cloud-based solutions may become expensive with high API usage.
5. Reliability: AI-generated test cases may miss edge cases or generate redundant tests.

5. Conclusion

AI is a valuable assistant in software testing, particularly for generating test cases, preparing data, and automating documentation. Cloud-based tools offer convenience and power, while local AI solutions offer control and privacy. However, challenges such as hardware requirements, execution overhead, and integration complexity limit full automation of testing workflows. A hybrid approach—using AI for test creation and manual/automated execution—currently provides the best balance of efficiency and reliability.