



Automatic generation of end-to-end tests using large language models (LLM)

Faculty of Applied Sciences, University of West Bohemia

Ing. Milan Janoch

Consultant: Ing. Tomáš Rojík

Supervisor: Ing. Martin Dostal, Ph.D.

Motivation & Problem

Manual creation and continuous maintenance of **End-to-End (E2E) tests** represent a severe **time and financial bottleneck** in rapid agile software development. This often leads to a compounding **technical debt**, where test suites become legacy code that teams actively avoid maintaining.

Traditional **record-and-replay tools** lack semantic depth, generate **fragile selectors**, and completely fail to capture complex business logic.

Conversely, attempts to generate test code via **static, single-prompt LLM** interactions inevitably fail due to **code hallucinations** and an inability to adapt to transient UI states. This issue is further compounded because the primary inputs – **agile user stories** found in tools like Jira – are **highly unstructured and inconsistent** in the real world. To bridge this gap between messy business specifications and executable Playwright code, a structured, **adaptive intermediary framework** is strictly required.

Core Benefits

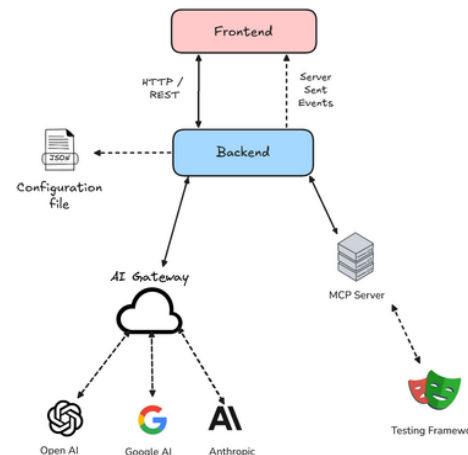
TestPilot AI fundamentally shifts the testing paradigm from error-prone script recording to an autonomous, intelligent verification process. By utilizing standard user stories as direct inputs, the framework empowers **non-technical QA engineers** and product owners to easily synthesize production-ready TypeScript code that strictly adheres to **Playwright best practices**.

The system does not generate code blindly; it performs **automated cross-verification** by executing real-time browser actions to validate acceptance criteria directly against the live application state. This approach eliminates the fragile, hardcoded selectors typical of conventional tools, drastically reducing technical debt and catching critical defects early in the release cycle.

System Architecture

The framework is distributed as a **local-first, Git-integrated** package. It stores generated test suites directly within the project development repository, ensuring the codebase remains the single source of truth without relying on external databases.

The core engine drives an autonomous **ReAct orchestration loop** (Thought → Action → Observation). Instead of creating scripts statically, the agent actively navigates and inspects live DOM states to discover verified element locators before synthesizing any code.



To guarantee safety and modularity, the system leverages the open **Model Context Protocol (MCP)**. This architecture isolates the browser automation runtime into an independent process, abstracting distinct LLM providers via a unified **AI Gateway** layer. When the agent encounters execution errors or syntax mismatches, a dedicated **self-healing compilation pipeline** automatically injects the TypeScript linter feedback back into the context. This feedback loop allows the system to autonomously refactor and correct its own testing scripts in real-time before finalizing the output.

Evaluation & Cross-Verification

- **Rigorous Multi-Model Benchmarking:** Evaluated 9 state-of-the-art LLMs, specifically tracking and comparing leading frontiers including OpenAI **GPT-5**, Anthropic **Claude Sonnet 4.5**, and Google **Gemini 2.5 Pro**.
- **Controlled Defect Environment:** Developed a custom reference application embedded with 11 hidden production defects (such as DOM state desynchronization, authorization leaks, and missing UI elements) to evaluate the agent's capabilities.
- **High Defect Detection Rate:** Advanced reasoning models successfully detected the application bugs, automatically documenting the discovered discrepancies and flagging the failing scenarios.
- **Test Self-Healing:** Thanks to the self-healing loop and feedback from error reports, the system successfully corrected 19 initially failing tests.
- **Reliable Cross-Verification:** Proved the framework's capability to validate generated test assertions directly against both user-defined acceptance criteria and live application states.

Conclusion & Deployment

The implementation of TestPilot AI successfully demonstrates that integrating large language models with structured communication layers like the Model Context Protocol provides a viable architectural approach for automated software testing. It transitions automated QA **from static** code generation **into an interactive, tool-driven agent workflow**.

Comprehensive multi-model benchmarking verified the framework's core capabilities in test generation, cross-verification, and defect detection. Distributed as a local-first package, this functional **proof of concept** successfully validates the architecture and lays a solid foundation for future production extensions, such as automated test maintenance and seamless enterprise **CI/CD pipelines**.