TestPlayer 2.0

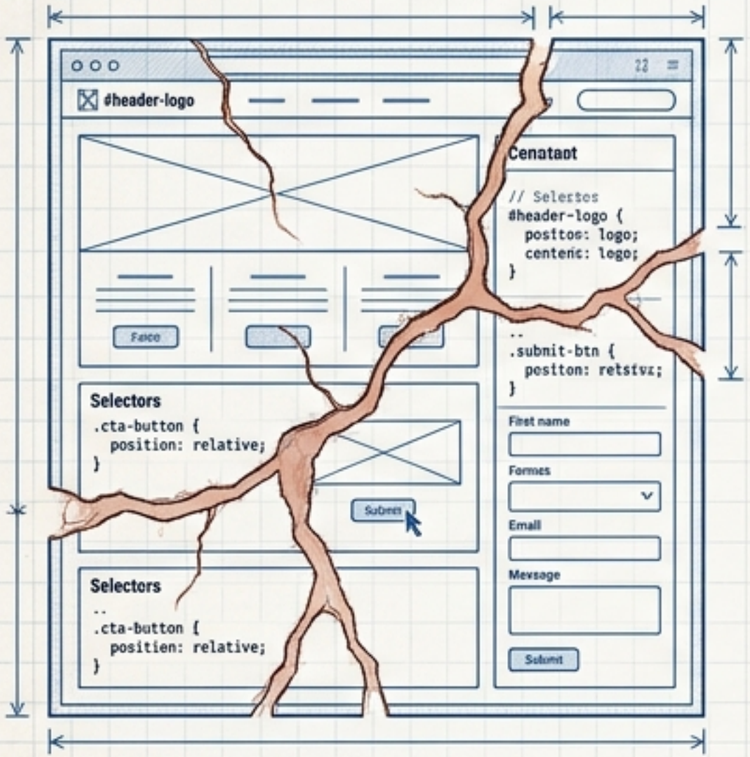
AI-Powered Model-Based Testing Platform

Autonomous UI exploration and dynamic test adaptation powered by Vision LLMs and Markov Chains.

THE QA MAINTENANCE TRAP

Classic End-to-End (E2E) UI tests are fragile, expensive, and fail to scale with modern agile release cycles.

FLAKY TESTS

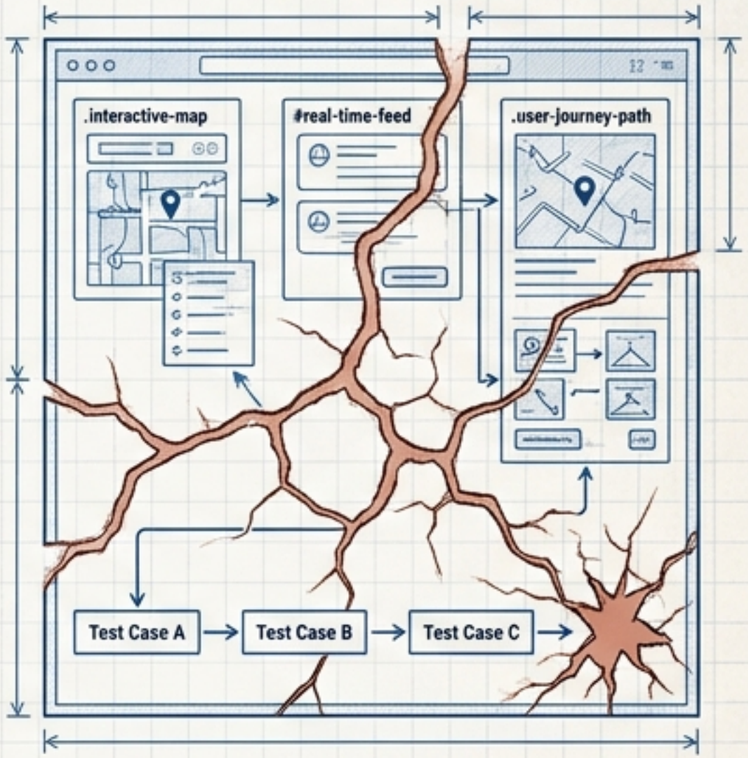


Static selectors break with every minor UI change. A single pixel shift or DOM update can halt a release pipeline.

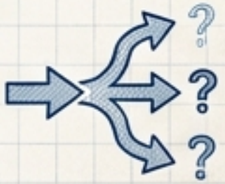
MAINTENANCE HELL

Constant manual script updates drain developer time, shifting focus from building features to repairing tests.

LINEAR LIMITATIONS



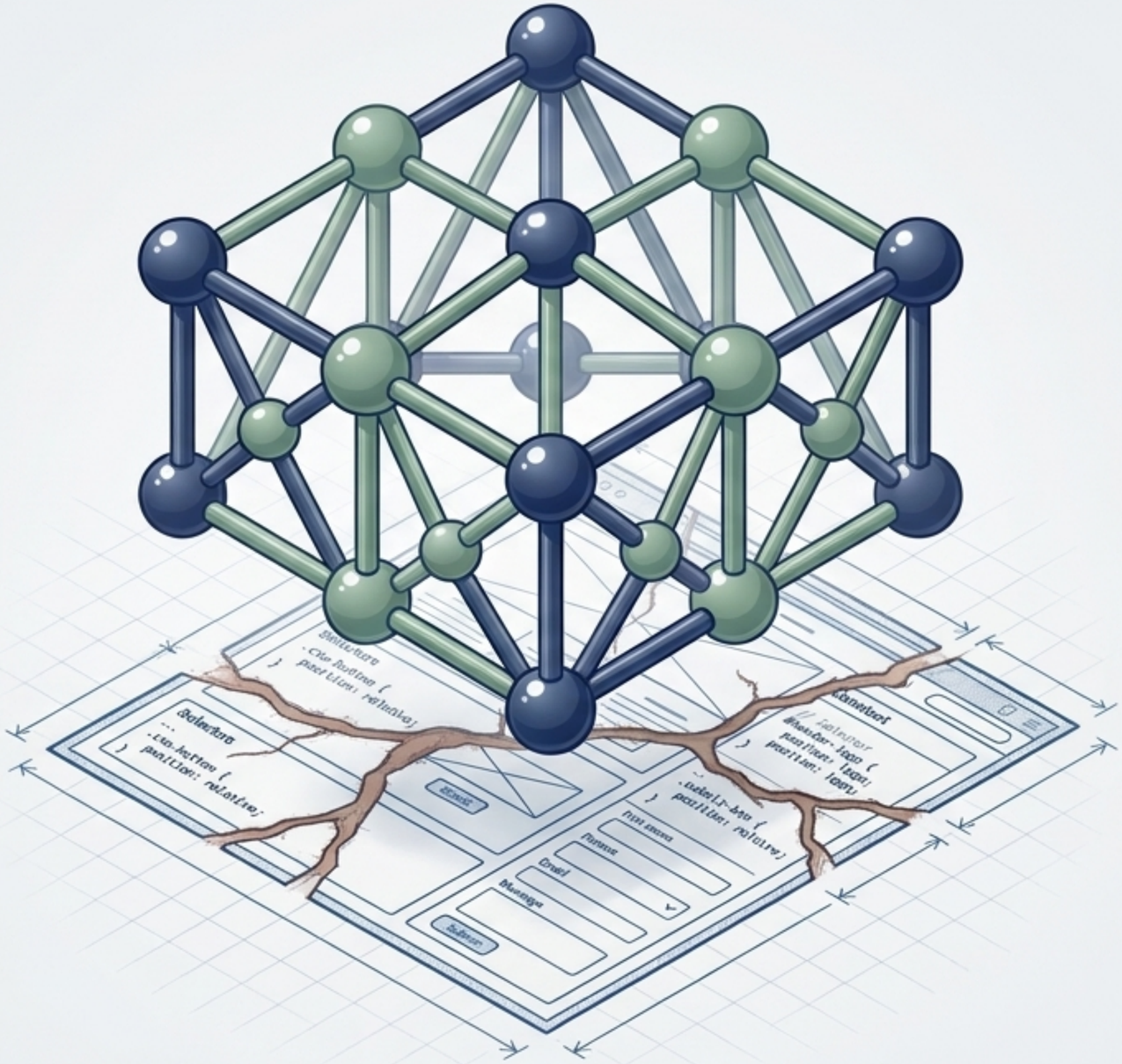
Handwritten, linear test cases cannot keep pace with the complex, stochastically driven user paths of modern web applications.



Autonomous, Self-Healing Testing

Slashing QA costs through generative resilience.

- **Auto-Discovery:** Vision-AI autonomously crawls and analyzes complex Web-UIs to generate complete, mathematically sound test models without manual scripting.
- **Self-Healing:** Context-aware intelligence dynamically repairs broken locators (DOM/CSS) on the fly during test execution.
- **Zero-Maintenance Scalability:** Execute highly dynamic, non-linear test suites that automatically adapt to your agile release cycles.

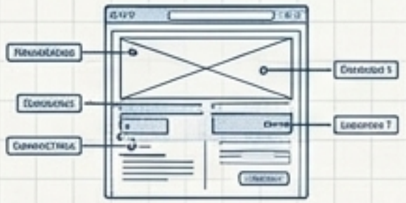


The Intelligent Engine: Where Math Meets Generative AI

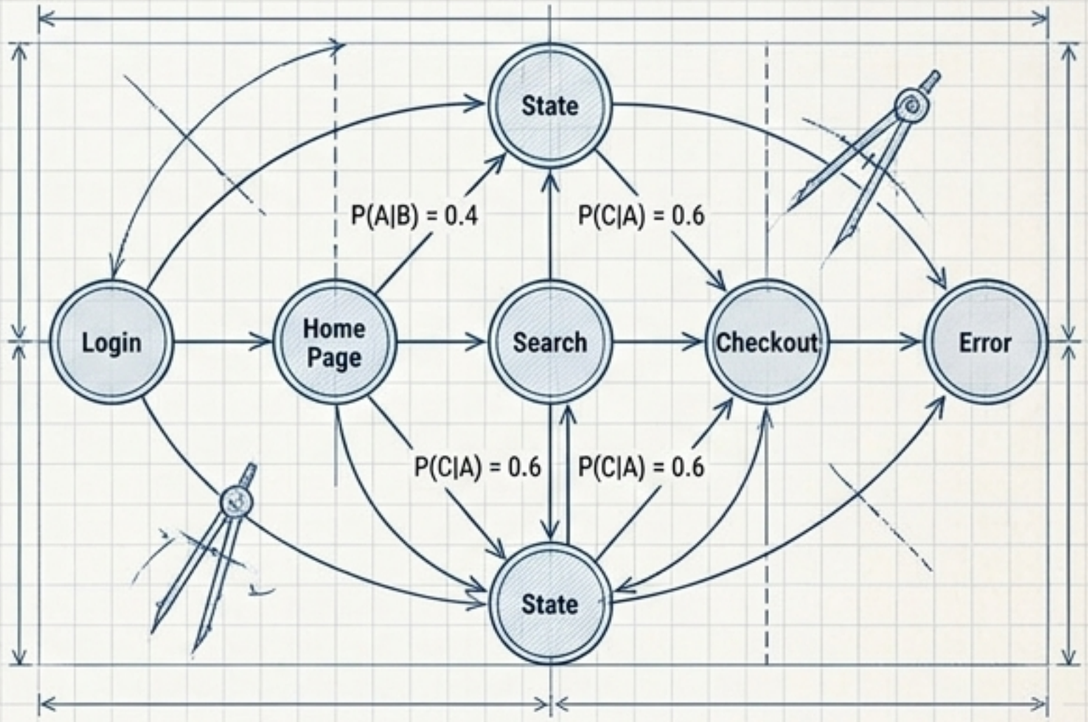
Unlike pure "Blackbox" AI tools that lack provability, TestPlayer 2.0 operates on a strict symbiosis between sensorics and stochastic control.

The Eyes (Sensorics)

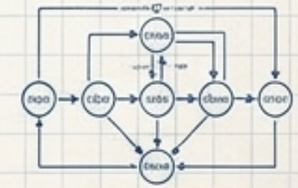
Powered by **Vision-LLMs** and **SOM** (Set-of-Marks) DOM extraction. It "sees", interprets, and heals the UI dynamically.



The Brain (Control)



Powered by **Markov Chain Usage Models** (MCUM). Test coverage, state transitions, and execution flows remain mathematically rigorous, strictly provable, and fully deterministic.



The Auto-Discovery Flywheel

How the engine autonomously builds topological ‘Ground Truth’ state graphs.

Initialization

Navigates to the start URL and deeply analyzes the Document Object Model (DOM).

Action Extraction

A heuristic parser, utilizing Vision LLMs, identifies all interactable elements (buttons, forms, links) while filtering dynamic noise.

Execution & Tracing

Executes actions (e.g., clicks) using heuristic DFS-backtracking and waits for a network-idle status.

Graph Building

Logs the origin state, edge (action), and target state, compiling a robust State Transition Model on the fly.

Enterprise-Grade System Architecture

A robust, asynchronous technology stack built for massive testing workloads and real-time trace evaluations.

1: The Interface (UI & Dashboards)

Tech: Bootstrap 5, D3.js, Chart.js, Graphviz.
Function: Interactive MBT IDE, analytical visualizations, and diagram viewers.

2: The Intelligence (External)

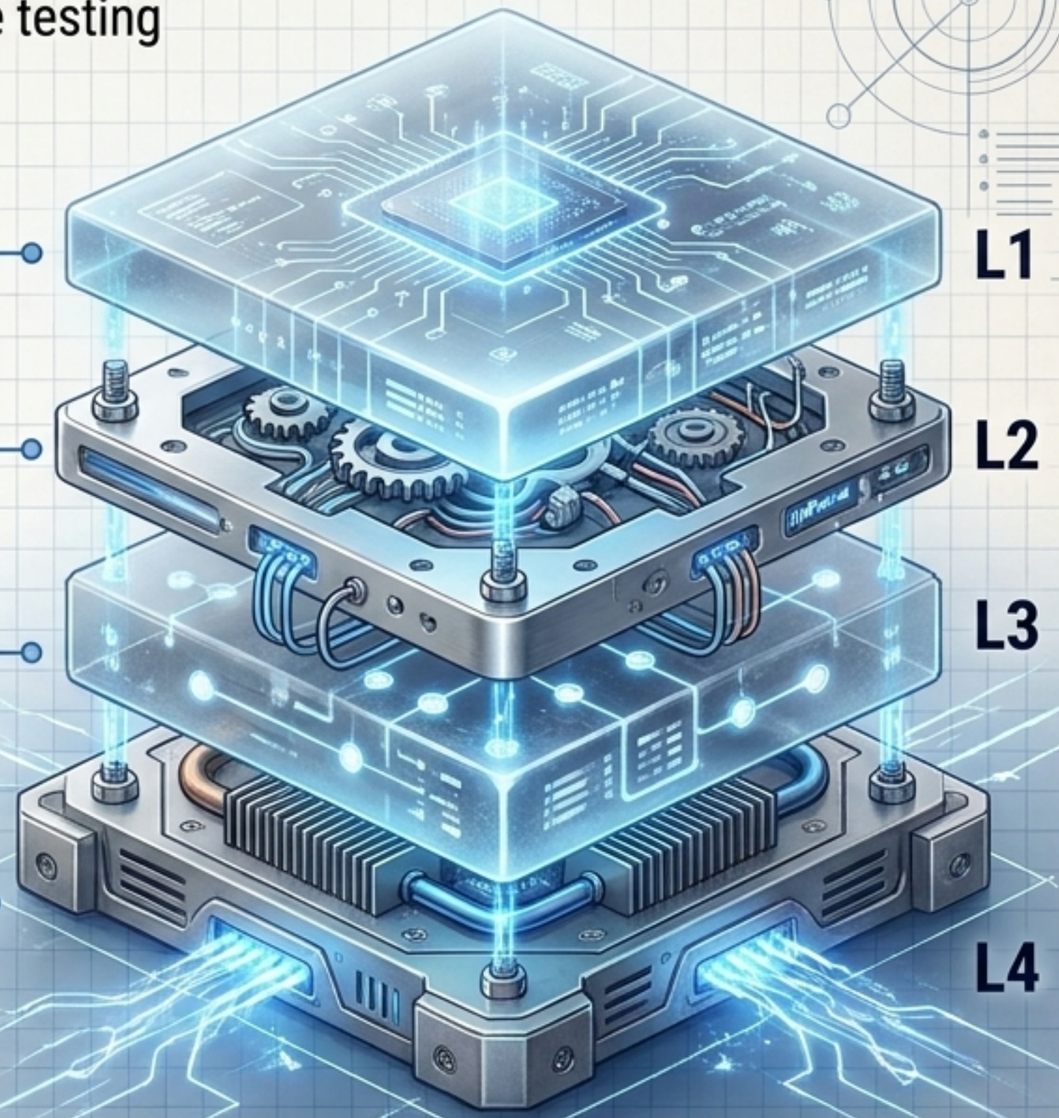
Tech: Vision-LLM APIs.
Function: Semantic mapping, heuristic DOM parsing.

3: The Engine (Execution)

Tech: Playwright, Django-Q2, SUT-Adapter.
Function: Headless UI interaction, async task queueing, robust execution.

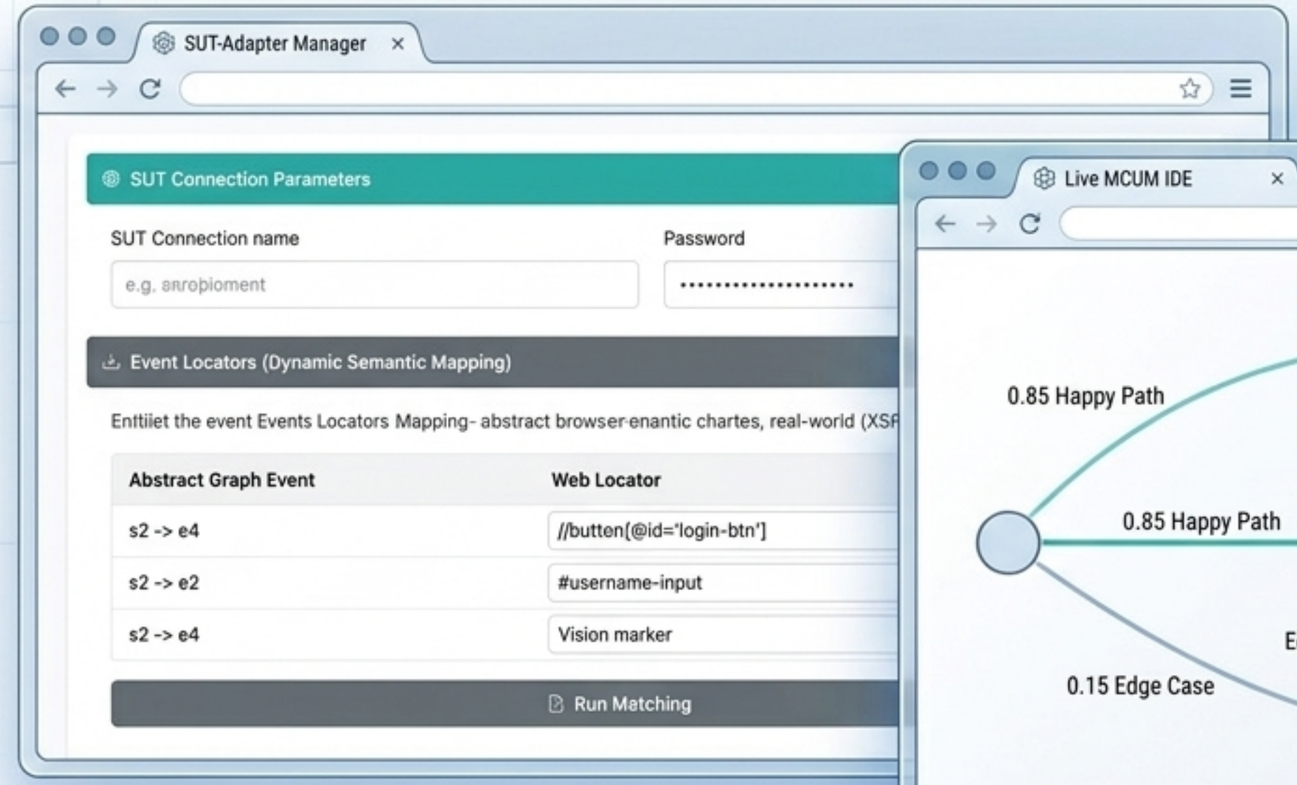
4: Foundation (Data & Core)

Tech: Python 3.12, Django 6.0, DuckDB.
Function: High-performance trace analytics and centralized state management.

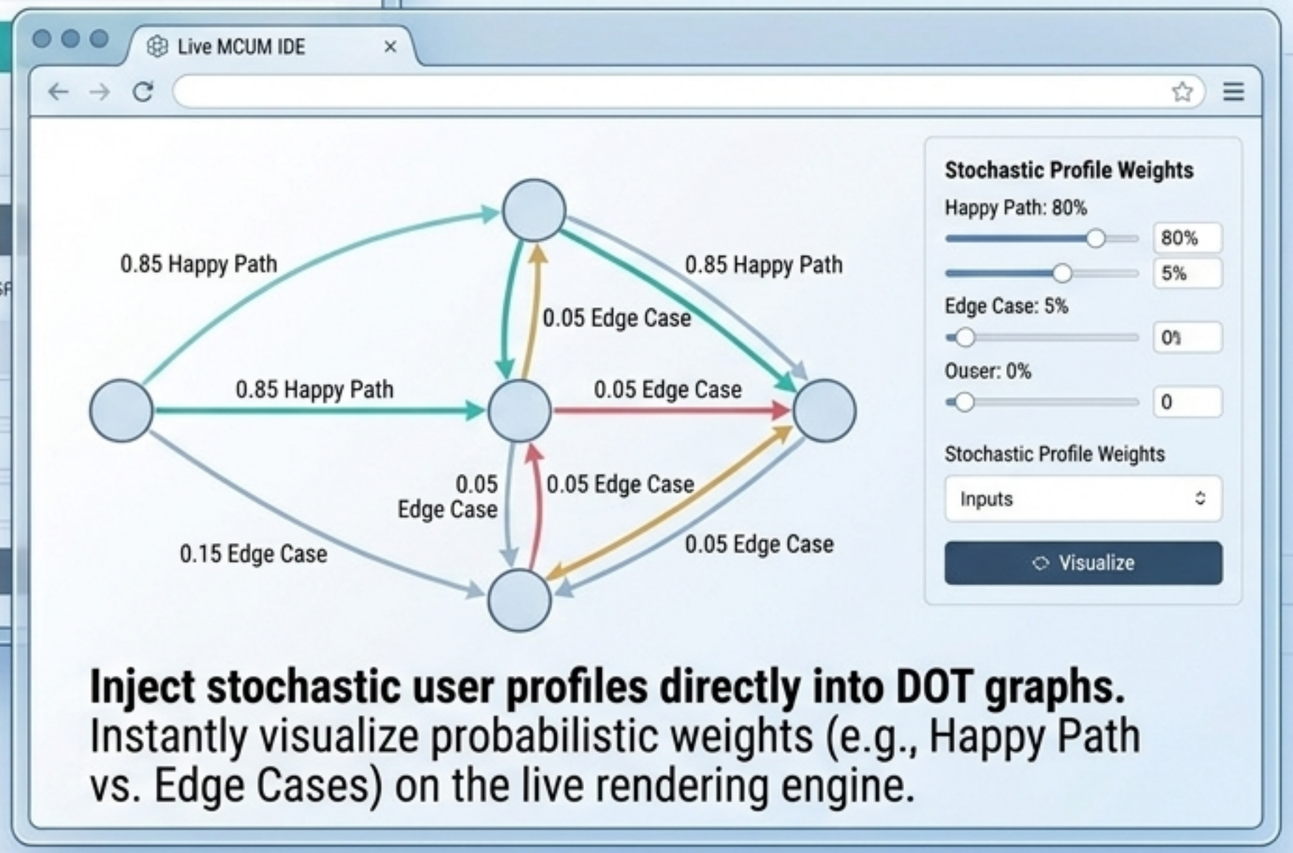


The MBT IDE & SUT-Adapter

Bridging abstract mathematical models with concrete web interactions.



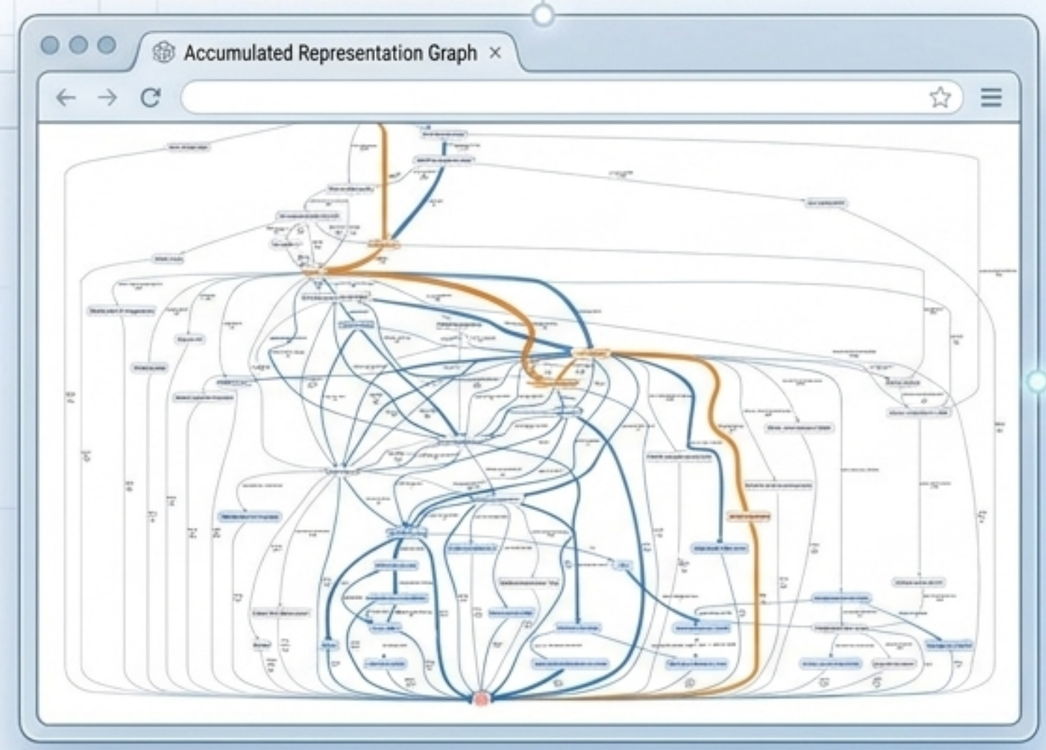
Robust dynamic semantic mapping.
Translates abstract graph events (e.g., s2 -> e4) into precise, real-world web locators (XPath/CSS/Vision markers).



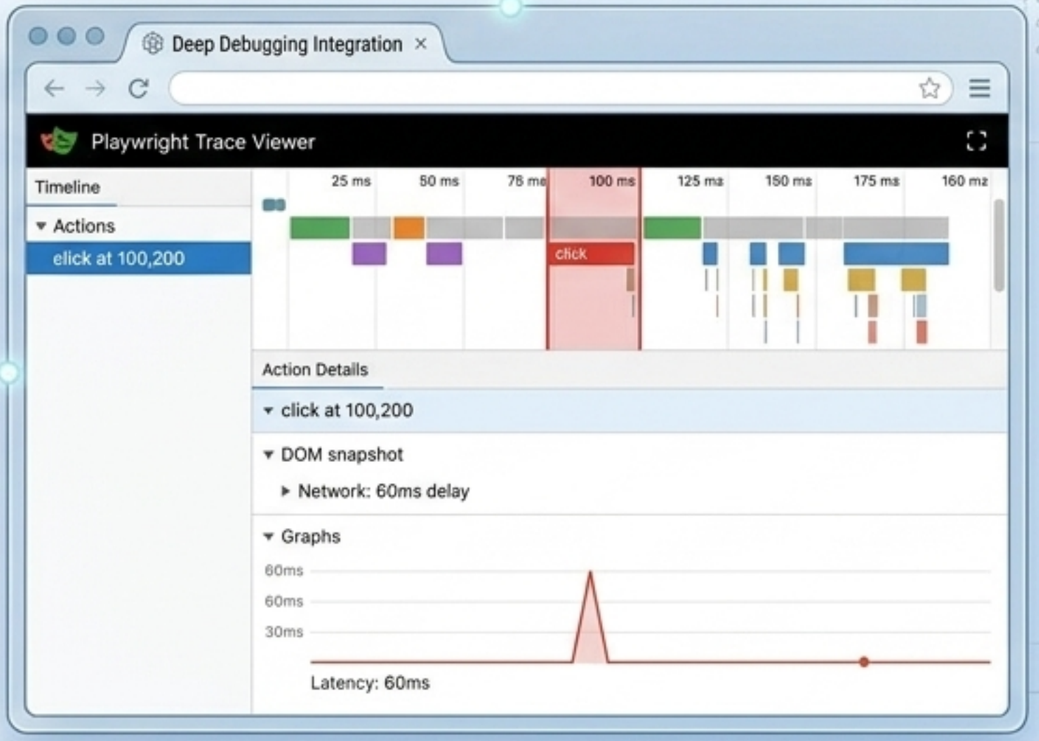
Inject stochastic user profiles directly into DOT graphs.
Instantly visualize probabilistic weights (e.g., Happy Path vs. Edge Cases) on the live rendering engine.

Visual Diagnostics: Seeing the Code Execute

Unprecedented transparency with 100% state coverage visualization.



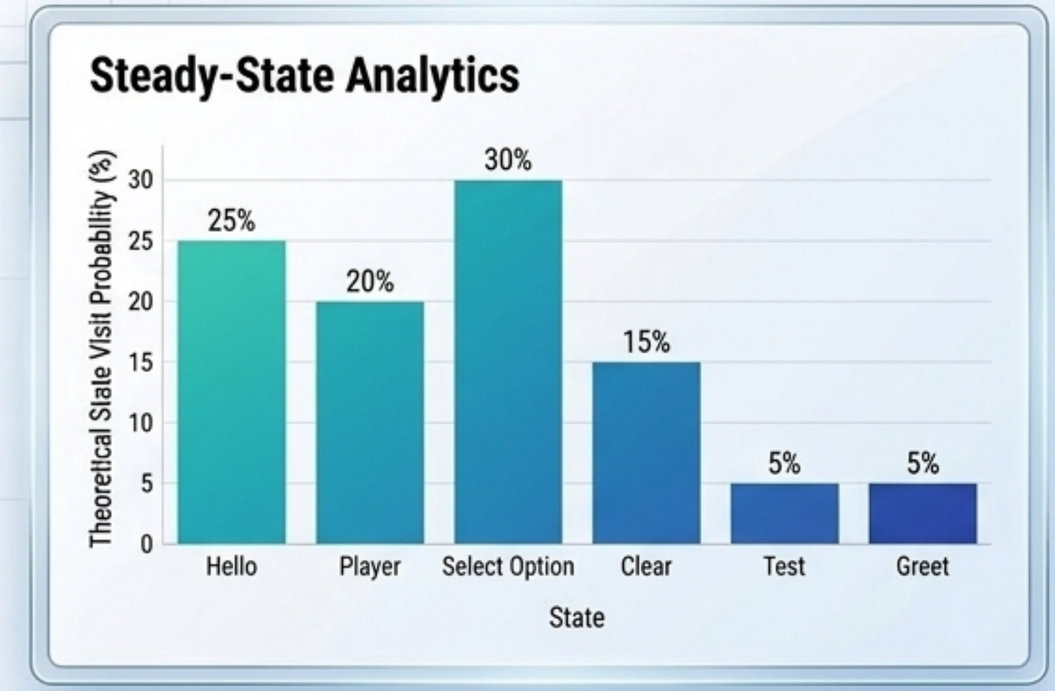
Frame-by-Frame Tracing: Visualize the exact historical path of test suites. Identify heavily traversed routes and isolate untested edge cases with pinpoint accuracy.

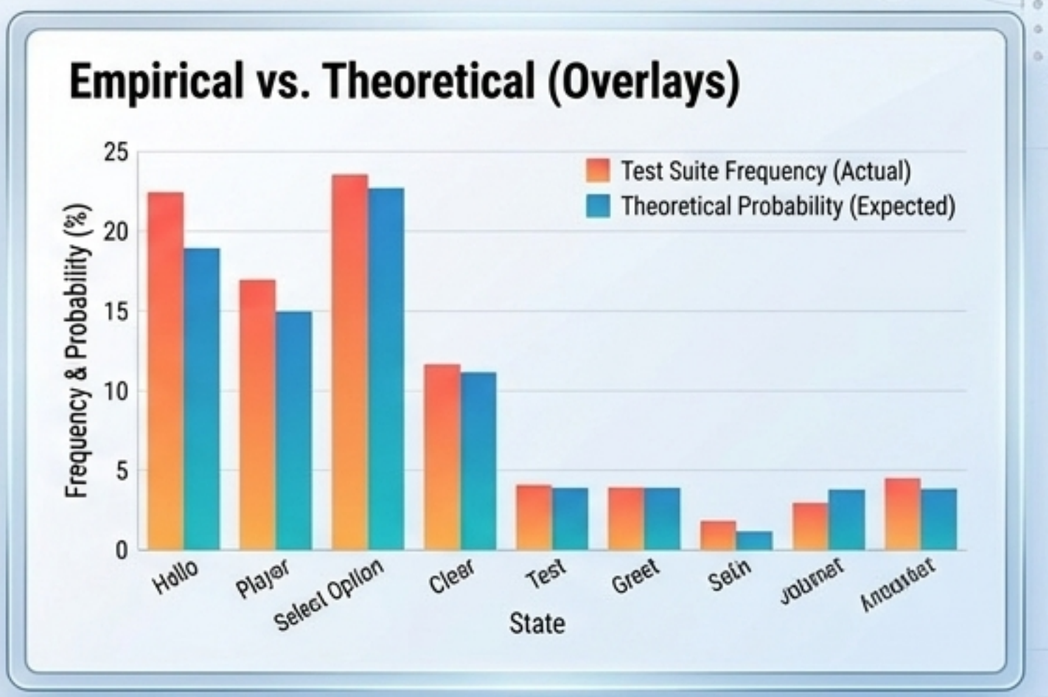


Trace Evaluation: Full integration with Playwright Trace Viewer for exact DOM state inspection, network latency analysis, and micro-bottleneck identification (e.g., 60ms click delays).

Data-Driven QA: Stochastic Analytics

Track performance, calculate probabilities, and validate test quality in real-time via DuckDB.





- **Steady-State Analytics:** Calculate the long-term mathematical probability of state visits to ensure critical user journeys are optimally tested.
- **Empirical vs. Theoretical (Overlays):** Automatically compare generated test suite frequencies against theoretical MCUM probabilities to prove stochastic convergence.
- **Performance Optimization:** Drastically reduce test execution volume by identifying the mathematical point of diminishing returns.

Case Study: Project ‘Playwright’

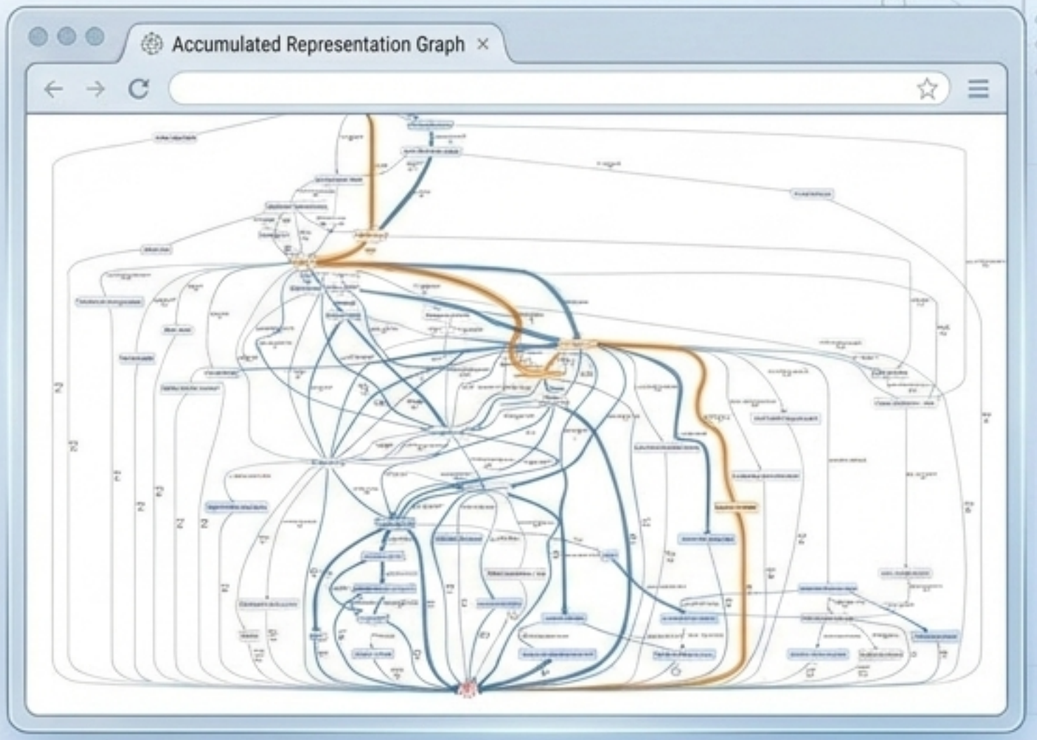
Autonomous discovery deployed in the wild.

- **The Target:** The official Playwright documentation website.
- **The Action:** An autonomous DFS (Depth-First Search) Model Discovery run by the TestPlayer 2.0 agent.

15
Semantic States Discovered

137
Events Automatically Mapped

100%
Node Coverage Achieved



Conclusion: A complete, executable, mathematically sound test suite generated in minutes—without a single line of manual test code written.

The Competitive Landscape

The only platform combining State-Machine Rigor with AI-Sensoric integration.

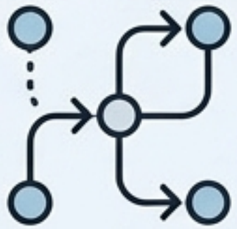
Feature	Scripting Frameworks (Playwright, Cypress)	AI-Testing SaaS (Mabl, Testim)	Traditional MBT Tools	TestPlayer 2.0
Low Setup & Maintenance Effort	✗	✓	✗	✓
Autonomous Self-Healing	✗	✓	✗	✓
Stochastic State Modeling (MCUM)	✗	✗	✓	✓
Vision-LLM Sensoric Integration	✗	✗	✗	✓

Target Market & Commercial Value

Built to unclog CI/CD pipelines and scale enterprise delivery.

Target Audience

- Enterprise QA Departments
- Agile DevOps Teams
- Specialized Software Agencies



The Bottleneck (Pain)

QA engineering hours are disproportionately spent on maintaining fragile legacy scripts rather than expanding test coverage and product quality.

The Value Proposition (ROI)

- Drastic reduction in manual engineering hours.
- Elimination of the ‘flaky test’ bottleneck stalling rapid CI/CD deployment.
- Highly scalable, data-driven QA reporting.

Business Model: Dual Licensing Strategy

A proven monetization engine for modern developer tools.

Open Source (AGPLv3)

- **Focus:** Frictionless, bottom-up developer adoption.
- **Benefit:** Rapid community growth, widespread testing in diverse environments, and organic product evangelism.

Commercial Enterprise License

- **Focus:** Top-down enterprise monetization.
- **Benefit:** Premium closed-source features, priority support SLAs, compliance guarantees, and commercial integration needs.

Roadmap & Technical Milestones

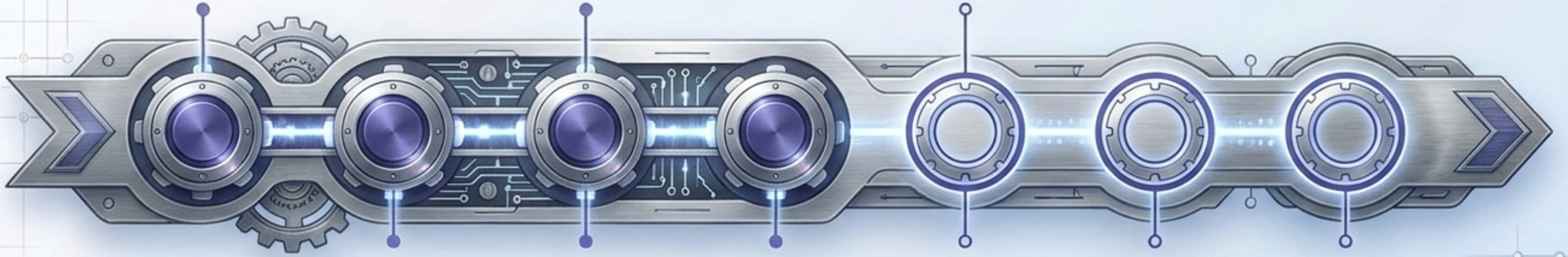
A track record of complex engineering execution.

Achieved:

- Proprietary Heuristic Matching & DOM Hashing.
- DuckDB Analytics & D3.js visualization integration.

Next Steps:

- Enhanced Vision-LLM fine-tuning for edge-case UIs.



- DuckDB Analytics & D3.js visualization integration.
- Autonomous DFS Model Discovery algorithm completed.
- SUT-Adapter dynamic lifecycle & Self-Healing active.
- Enhanced Vision-LLM fine-tuning for edge-case UIs.
- Expanded native CI/CD pipeline integrations (GitHub Actions, Jenkins).
- Launch of the Final Enterprise Commercial Release.

The Future of Provable, AI-Driven QA.

TestPlayer 2.0 transforms software testing from a manual, fragile chore into an autonomous, mathematically provable science.

Lead Software Engineer: Dr. Winfried Dulz

Join the Beta | Explore Enterprise Partnerships
testplayer.io | partners@testplayer.io