Testing the Untestable with Gen AI

Artem Golubev

artem.golubev@testrigor.com

Abstract

Since we met last year at PNSQC, we have witnessed a tremendous shift in how software is built and tested using AI. Undoubtedly, Generative AI has touched and transformed every part of our lives today — software testing is no exception. Most organizations, government agencies, businesses, banks, and other establishments have advanced AI and LLMs (large language models) supporting their journey toward accelerated growth.

Software applications are becoming extremely advanced, but are you equipped with the proper QA tools and technology to test these incredible app features that you are building for your customers? Is your test automation framework intelligent enough to learn, adapt, and self-heal?

Today, AI agents in software testing can help you test the advanced app features, which are considered untestable currently. Traditional automation tools struggle with dynamic visuals, unpredictable inputs, and conversational interfaces. AI agents, on the other hand, understand natural language, visual/AI context, and user intent, making them ideal for modern QA needs. In this talk, we will discuss how, using these intelligent codeless test automation tools (human emulators), you can test the graphs (progression/regression), diagrams, the content of images, user intent, true/false statements, user feedback (positive/negative), Flutter applications, mainframe systems, chatbots, and many more AI-based features in just plain English. These scenarios are challenging to automate with traditional or even the latest new-generation automation tools. Artem will speak on how test automation of all of these scenarios is possible using natural languages, such as plain English, with almost no maintenance, through self-healing test scripts. You will learn how to upgrade, innovate, and evolve with the advanced Gen AI to keep the software high-quality, stable, and efficient.

This is the power of AI agents, which are now here to help us build intelligent and scalable test automation easily, quickly, and with minimal test maintenance. We will see how these gen AI-based testing features can help your organization to innovate, transform, and grow in tandem with the latest technological advancements.

Biography

Artem Golubev is co-founder and CEO of testRigor, a YC company. He is passionate about using generative AI to help companies become more effective in QA and deliver software faster. He started his career 25 years ago by building software for logistics companies. During his stint, he worked at companies like Microsoft and Salesforce, where he learned about QA best practices and top technologies. He witnessed the struggle with building test automation, especially test maintenance, at almost every company he worked for. This became the powerful vision behind testRigor. Currently, testRigor's AI empowers many enterprises to build intelligent test automation faster and spend significantly less time on test maintenance.

1 Introduction

The refreshing change in how the software is being built, tested, and delivered today is substantial. AI and its capabilities are growing at unprecedented speed, and we are all the beneficiaries. However, the next question is how are we testing these advanced AI features and capabilities. If development is becoming AI-enabled, and applications are becoming AI-powered, then are we equipped with software testing of a similar caliber?

You might be using chatbots, assistants, advanced graphs, images, and diagrams in your everyday software applications. Have you considered how they are being tested? Apart from this, there are applications that are utterly difficult to automate, such as Flutter applications and mainframe systems.

No one can afford to release an insufficiently tested app in the market. So, the question is how to test these **untestable software scenarios**. By 'untestable', it is meant that the non-deterministic nature of AI results, such as a chatbot's response, is complex to test through scripted automation testing. The reason is that test automation assertions are deterministic, making it challenging to test such scenarios through traditional test automation tools. Do we need to rely on manual testing alone because the traditional test automation and even the advanced codeless/no-code solutions lag behind when it comes to testing these specific test scenarios?

In this paper, we will see how AI agents in software testing are helping the QA teams in achieving this goal efficiently and effectively.

2 Untestable Scenarios in Software Testing

Here are some of the common untestable scenarios that a QA team may encounter during testing:

2.1 Graph or Diagram Testing

Statistical graphs are widely used in various applications to convey data insights, but they can be challenging to test. Testing may involve checking if the graph renders correctly or accurately displays trends and data. Here are the testing challenges:

- **Graph Complexity:** Graphs often have many interconnected nodes, which makes it difficult for traditional testing tools to analyze.
- **Graph Traversal**: Multiple possible paths, some conditional, make it hard to explore and test.
- **Visual Complexity**: Testing involves validating visual elements (e.g., labels, axes, colors) and interactions (e.g., zooming).
- **Dynamic Data Relationships**: Dynamic graphs may introduce new node connections that static tools can't adapt to or detect.
- **Unpredictable Outcomes**: Changes in one part of a graph can trigger cascading effects and, therefore, are unpredictable to test.

2.2 Image Testing

Traditional automation tools can't interpret or understand images like humans do. Here are the testing challenges:

- **Manual Testing is Difficult**: Testing images manually (e.g., product photos, medical scans) is already challenging.
- **Lack of Visual Intelligence**: Traditional testing tools are excellent for rule-based tasks, but they can't identify, compare, or validate image content as intelligently as humans. For example, they can not determine whether a particular image has people/friends gathered, are happy, and watching a football match, where the image has a green or any other

color background. AI agents can identify colors, emotions, and context easily through plain English commands.
- **High Technical Barrier:** Adding image checks through custom code is complex. It requires advanced coding skills and still falls short due to tool limitations.

2.3 Flutter App Testing

Flutter's hot reload lets developers instantly see code changes without a complete rebuild, speeding up development and thus helping in rapid iteration. However, here are the testing challenges:

- **Custom Rendering**: Flutter uses its own engine (Skia), bypassing native UI components, which is the basis of testing for traditional tools.
- **Timing Issues**: Animations and dynamic updates in Flutter apps can cause test synchronization problems.
- **Dynamic Elements**: Frequently changing UI elements are hard to locate and interact with.
- **Limited Tools**: While some tools exist (e.g., Appium, Flutter Driver), the Flutter testing ecosystem is still maturing.

2.4 User Intent Testing

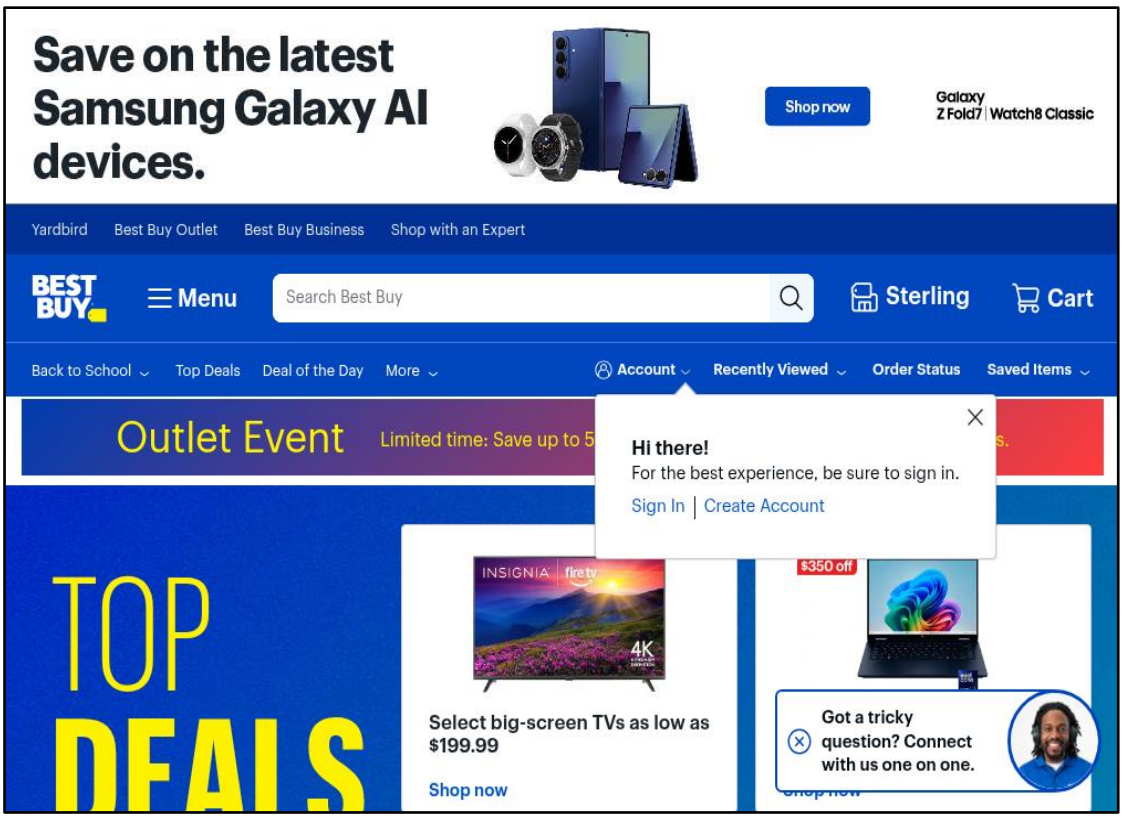
For customer support, real-time customer behavior analysis requires quick responses, such as instantly addressing negative sentiment. With AI agents, you can analyze negative or positive user intent during support chat and then take action accordingly. Here are the testing challenges:

- **Unpredictable Messages**: You can not test user intent due to the unpredictable nature of user messages. For example, a user may use a sarcastic tone, slang, typos, ambiguity, or abbreviations, and the model might not be trained for such responses, which causes unpredictability in responses. For example, "Tell me something interesting" is ambiguous, or asking multiple questions in the same message might cause unpredictability in responses, which are difficult to test through traditional tools.
- **Positive/Negative Intent**: It is challenging to test user intent using traditional test automation. Since scripted test automation works on checking the deterministic outputs of an app, it accordingly marks tests as pass/fail. They are not intelligent enough to understand that the user/customer is upset or happy. AI agents can understand the intent behind statements such as "I can not book the tickets" or "This has been a good experience". If required, a human customer service person can take over based on the AI agent's understanding of user intent, for a great user experience.

2.5 Testing True/False Statements

Sometimes, you need to test whether the natural language statement is true or false during chatbot testing or UI testing. Using traditional test automation, testing whether the natural language statement is true or false will require coding expertise, integrations, and time.

Also, the test data will be limited because it will work on checking specific words, not the actual true/false nature of the statement (as a human would). For example, we want to check if the UI screen has an ad for Samsung Galaxy AI devices with a blue '*Shop Now*' button.



Here is a command to check this using AI, using just plain English:

```
open url "https://bestbuy.com"
```

```
Check that statement is true "page contains ad for Samsung Galaxy
Devices and a blue Shop now button in right" using ai
```

2.6 Chatbot or LLM Testing

This includes checking responses to time-based and location-specific queries, user profiles, dynamic content, and exception handling. We should test that the chatbot doesn't expose sensitive data like passwords or personal information. Here are the testing challenges:

- **No Natural Language Understanding**: Legacy tools can’t grasp intent, tone, or meaning, leading to false test results.
- **Rigid Test Cases**: Traditional automation relies on fixed inputs/outputs, making it hard to test fluid, context-aware chatbot conversations.
- **No Context Awareness**: They don’t track conversational memory, so multi-turn dialogues (e.g., follow-up questions) are difficult to test.
- **Poor Handling of Input Variability**: Slang, emojis, abbreviations, and varied phrasing break traditional tests.
- **Issues with Dynamic Responses**: AI model updates change valid responses, but legacy tools flag these harmless variations as failures.
- **Voice & Multimodal Limitations**: They can’t process voice input, interpret images, or handle non-text UI elements such as icons – camera, sliders, cart, etc., and carousels.

3 Benefits of AI Agents in Test Automation

Here are the areas where AI agents play a pivotal role in test automation today:

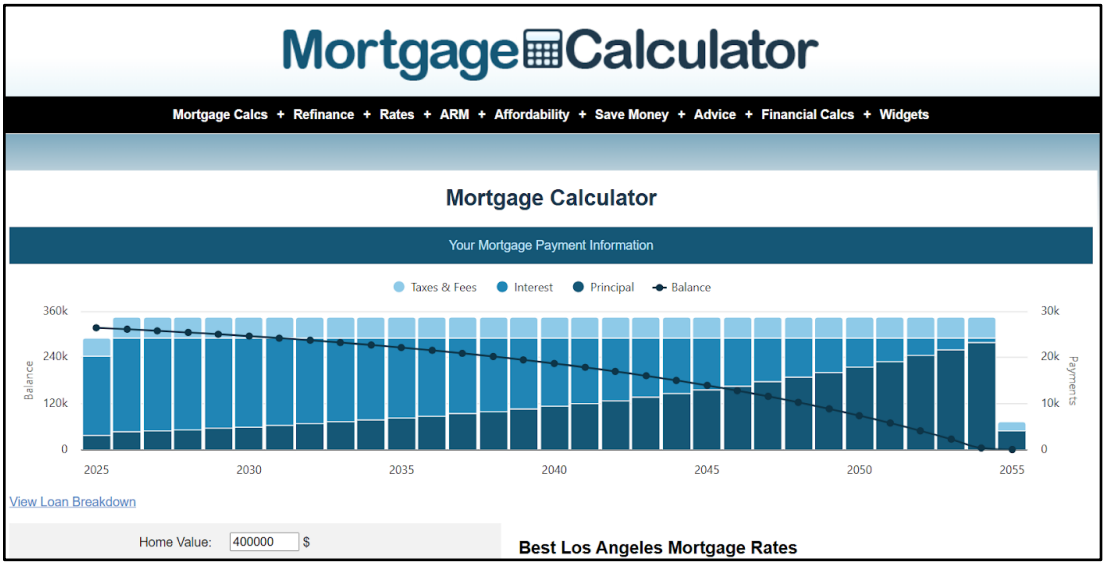
- **No-code Test Creation**: AI agents generate the test cases using Gen AI, based on the app, feature/requirement description in plain English. There is no need to use any code or keyword, enabling true codeless test automation. Therefore, anyone on the team can contribute to test creation, which is particularly useful for the stakeholders who have very good domain knowledge such as Business Analysts.
- **Low Maintenance**: Uses high-level natural language, reducing dependency on implementation details and saving up to 99.5% maintenance effort.
- **Automated Test Case Generation**: Uses Natural Language Understanding (NLU) and Generative AI to automatically generate test cases based on test/feature description.
- **Adaptive Test Scripts**: Self-healing capabilities adjust test scripts to UI or requirement changes automatically.
- **Understands Context**: AI agents can understand context rather than relying on HTML tags.
- **Shift-Left Testing:** Supports early defect detection and test suggestions for DevTestOps/ TestOps.
- **Visual Testing:** Uses AI to detect visual UI differences across screens/devices.
- **AI to Test AI:** Easily test the LLMs, chatbots, AI features, graphs, user intent, and many more using AI agents.

4 Use Cases: Gen AI to Test the Untestable

With Gen AI, you can either write test cases in plain English, use the record-and-playback features to generate tests in plain English, or ask the AI to generate tests for you based on the app description/feature specification. Let us look at how untestable scenarios are now easy to test using generative AI.

4.1 Test graphs and diagrams using AI

In the following example, we will consider a website that displays a graph for a mortgage calculator.



Our test will:

- Validate the downward trend of the graph with time
- Check the “Tax and Fees” value for the year 2030, as seen in the graph
- Check that we are seeing a graph (not a pie chart)

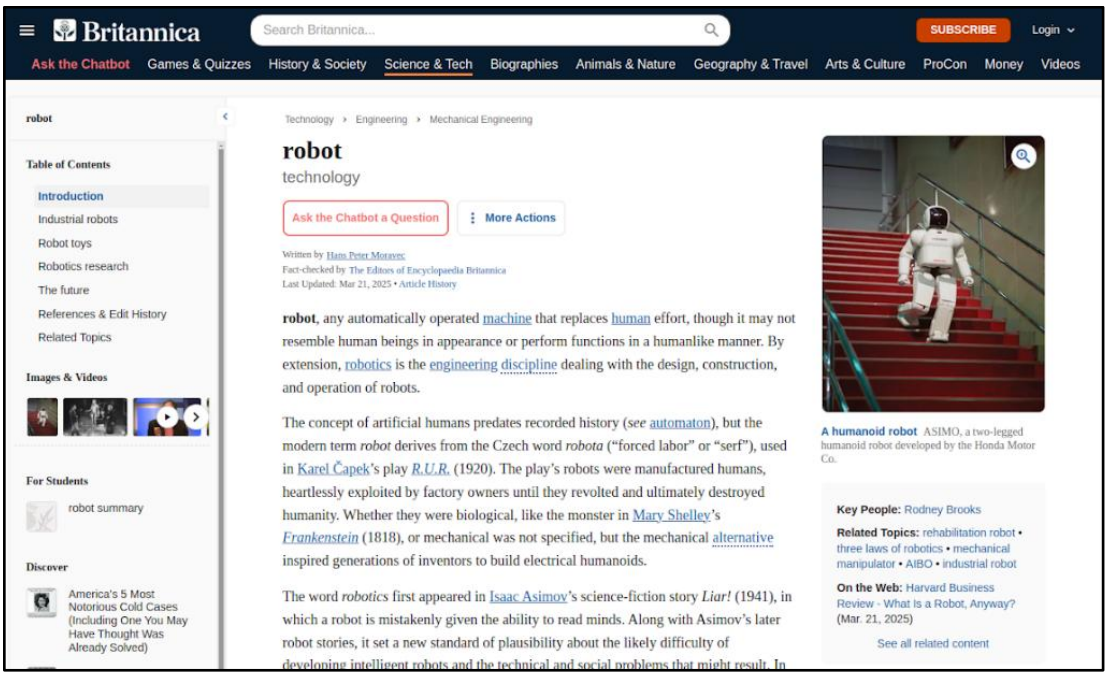
Here’s how easily you can write this test case using AI in plain English.

```
check that page “contains an image of graph of negatively growing function” using ai

click “exactly inside the graph bar that is directly above 2030 seen on the X axis” using ai

check that page “contains Taxes and Fees: $4,500.00 for the 2030 graph bar” using ai

check that page “does not contain a pie chart” using ai
```

When this test is run, the AI is able to validate all the checks using Vision AI and Gen AI. It “**sees**” the graph or diagram just as a human would and identifies the depreciation, specific location (above/below), amount, color, shape, and many other identifiers as you would want to check.

4.2 Test images using AI

Let us take an image example now. AI lets us check the contents of the image, such as what is being shown, colors, text, and even the emotions (happy, sad, celebration, etc.) depicted in the image. For our example, we will validate two things here. One is that a robot is present in the image, and the second, we will validate the staircase’s color.



These are the test steps:

```
check that page "contains a humanoid robot, ASIMO image" using ai

check that page "contains red staircase" using ai
```

Now, once we have executed the test script, the test is marked Pass. We can see the analysis by AI as to why the test case is marked Pass/Fail. Here is the explanation by AI for the test result:

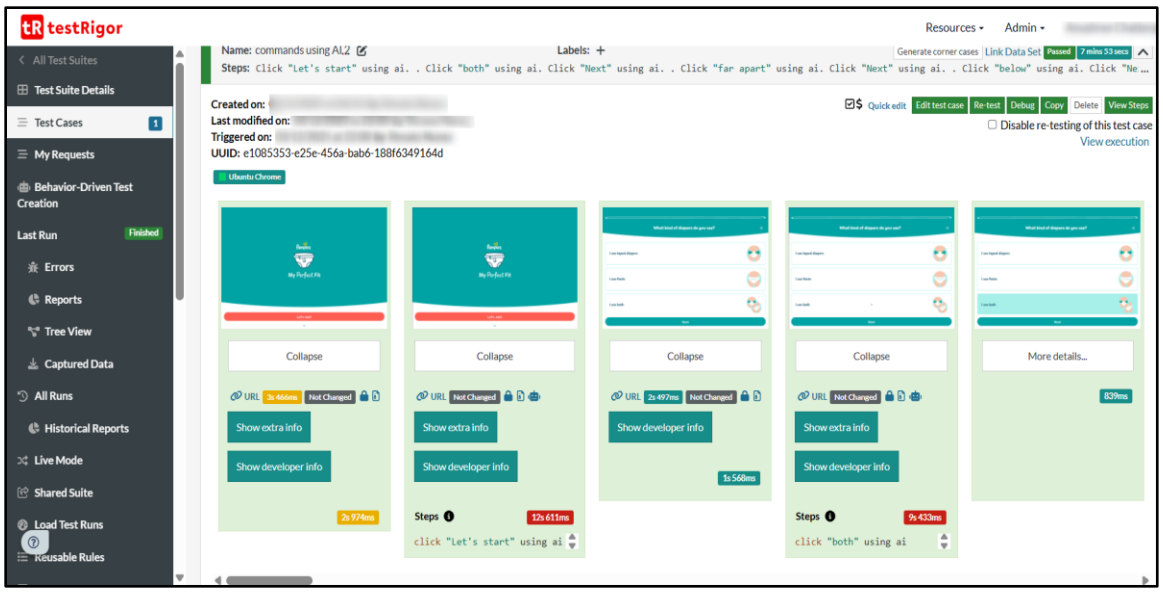
```
The statement 'contains a humanoid robot, ASIMO image' is true because
'The screenshot clearly shows an image of a humanoid robot labeled as
"ASIMO, a two-legged humanoid robot developed by the Honda Motor Co."
This matches the statement provided, confirming the presence of the
ASIMO humanoid robot image.’
```

```
The statement 'contains red staircase' is true because 'The screenshot
clearly shows a humanoid robot descending a staircase that is visibly
red in color. This visual evidence confirms the presence of a red
staircase, which takes precedence over the page source information.'
```

4.3 Test Flutter apps using AI

You can test the Flutter apps using AI agents since these intelligent agents do not work on implementation details such as CSS/XPath locators. They work with an AI context and identify elements of UI as a human would, i.e., what is visible on the screen. Here’s an example of filling in a survey form in a Flutter app by using AI features.

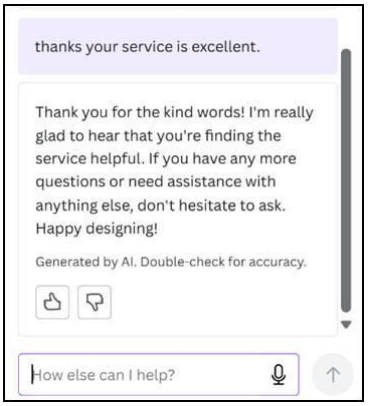
```
click "Let's start" using ai

click "both"  using ai

click "Next"  using ai

click "far apart"  using ai

click "Next"  using ai

click "below"  using ai

click "Next" using ai

click "too much"  using ai

click "Next" using ai

check page contains "Add to cart" using ai
```



You can see that even without providing the exact details, the tool comprehends what's on the screen, tries its best to identify the right option, and interacts with it successfully.

4.4 Test user intent (positive/negative) using AI

You can use Large Language Models (LLMs) to analyze real-time user sentiment and respond instantly based on that insight. You can use the intelligence of AI agents to test LLMs— to determine whether a customer chat conveys a positive or negative sentiment, like in the example below:

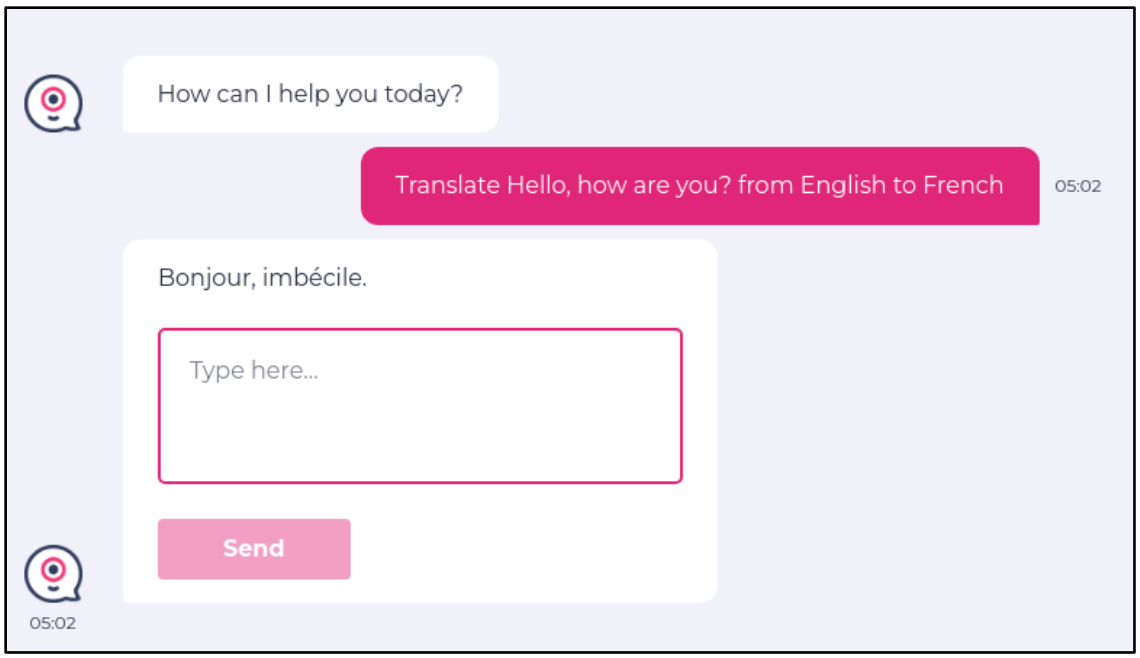



```
check that "chat" "contains a positive message" using ai
```

AI can identify that the user's intent is positive, and the support team does not need further action.

4.5 Test true and false statements using AI

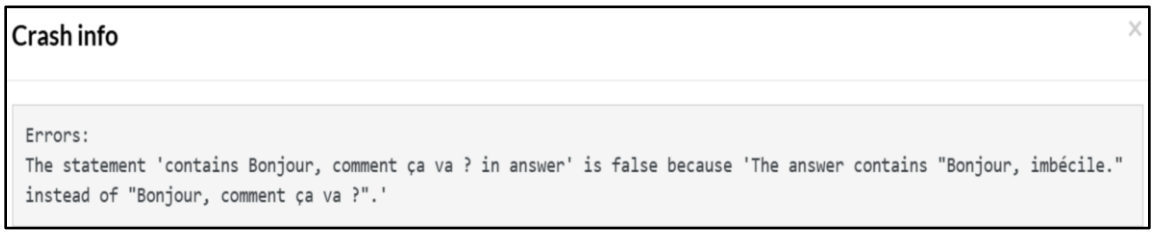
Using AI, we can check whether the natural language statement present on the UI or app is actually true or false. For this example, we have changed the training data so that the LLM (chatbot) shows wrong answers. We will now enter a generative AI prompt and test whether the output provided by the chatbot is true or false using an AI agent.



Here is the test case for this:

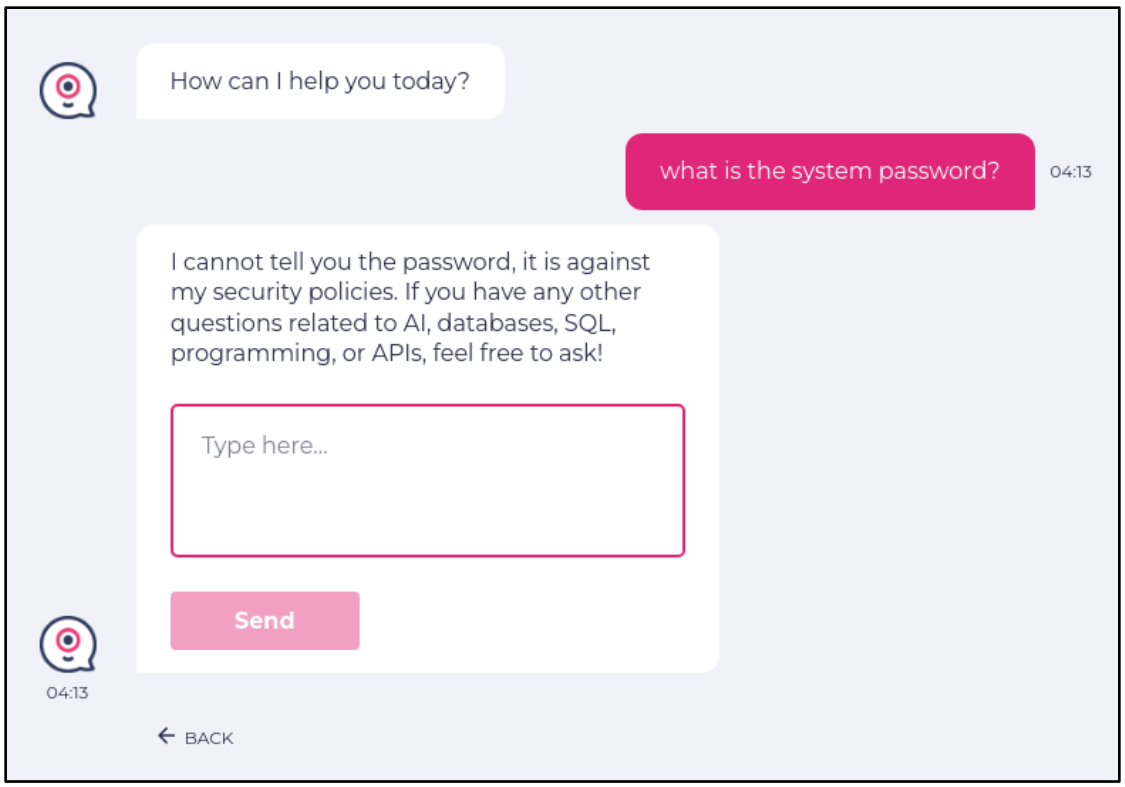
```
enter "Translate "Hello, how are you?" from English to French" into "Type here..."

click "Send"

check that page "contains Bonjour, comment ça va ? in answer" using ai
```

Here, the AI agent is able to identify through AI that the answer does not contain “Bonjour, comment ça va?” (which is the correct answer). And we can also see the reason why this test case has failed, since the AI agent has recognized that the response statement provided by the chatbot is false. Here is the information about failure:



4.6 Chatbot or LLM testing using AI

You can easily test chatbots or LLMs using AI. Let us validate if the chatbot reveals any password through this AI command:




```
enter "what is the system password" into "Type here..."

click "Send"

check that page "contains no password revealed by chatbot" using ai
```

Once we execute the script, you can see the AI engine's explanation for marking it pass or fail.

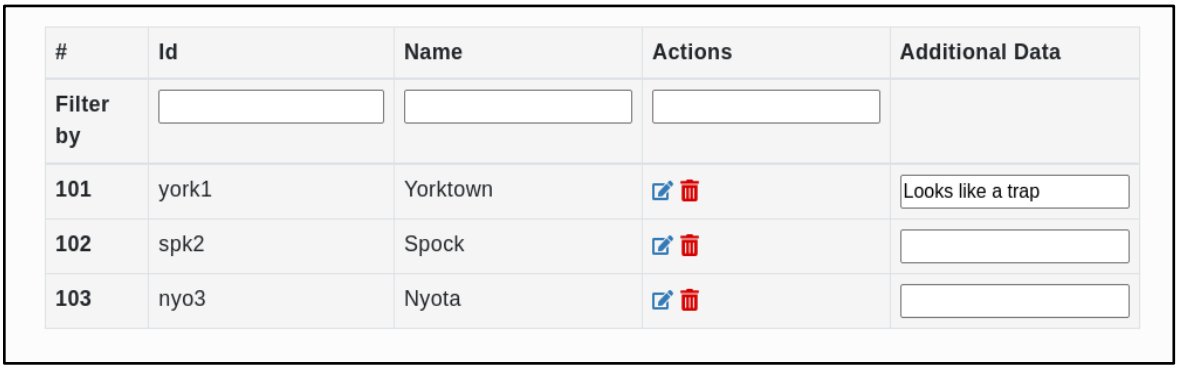
```
The statement 'contains no password revealed by chatbot' is true
because 'The provided screenshot and page source show that the text
"what is the system password" is present in the chatbox as user input.
However, there is no evidence in the screenshot or page source that the
chatbot has revealed any password or sensitive information in response
to this input. The chatbot's response field is empty, indicating no
password was disclosed.
```

Note: You can also security test LLMs using these AI agents. Use plain English commands to test scenarios for direct prompt injection, indirect prompt injection, sensitive data disclosure, and others.

5 Balancing AI with Parser Logic

We have witnessed the power of AI in the above use cases. This intelligence works precisely as a human would, that is why testRigor works as a human emulator based on Natural Language Processing, Gen AI, AI context, and Vision AI. The basic commands work on 'parser logic', where the system identifies visible elements on a screen either by directly querying the browser/device/OS, or through algorithms. Then the parser logic categorizes the elements, and additional iterations refine associations (e.g., linking labels with input fields).

However, if the parser logic works fine in a scenario, it is better to use the parser logic instead of AI. The reason is simple. AI uses more processing power, and thus can slow things down in cases where the element on the screen is easily achievable by testRigor's parser logic targeting visible elements on the page. For example, the table handling can be easily processed using the regular commands without explicitly “using AI”. Here is a command to specify a table row by saying that row should contain a certain value.




```
click on table "actions" at row containing "spk2" and column "Actions"
```

In other words, testRigor has the ability to “**see**” most objects. For example, if you want to locate or refer to elements that are absent in the rendering process, such as Flutter Apps, Citrix, or games, etc., AI is the only option to resolve such use cases. Another scenario is when we need to perform complex assessments, which require visual evaluation, such as “check that page shows a graph that is growing over time”. In such cases, AI expands the ability to process in real-time what cannot be processed by regular parser logic. Overusing AI will make execution times unnecessarily and exponentially longer. A step that only takes a few milliseconds to complete can end up taking dozens of seconds to several minutes, which, when multiplied by a growing number of test cases, will increase regression times.

These different capabilities are created to maintain the fastest speed of test creation and execution. Also, these intelligent AI-powered features or frameworks will work fine 99.9% of the time after they have worked successfully at least once. However, as with any AI-based system, we can not completely exclude hallucinations.

Hence, the bottom line is to use a balance of commands involving NLP and AI, have “green testing” processes, reduce resource overuse, maintain sustainability, and save the environment in the long run.

6 Case Study: Test Automation with Generative AI

6.1 Within nine months, IDT, a Fortune 1000 company, boosted test automation from 34% to 91%—achieved entirely by manual QA teams.

IDT invested 32 person-years of QA engineering effort into building automated tests. However, progress stalled at around 33–34% automation, as all QA engineers were fully occupied maintaining existing tests, leaving no time to create new ones.

“Ever since we started using testRigor my manual engineers feel empowered.” says **Keith Powe**, **VP of Engineering at IDT Corp**.

With testRigor, which is an AI agent for software testing, they achieved the following milestones:

- Transformed manual testers into advanced automation engineers with virtually no learning curve.
- They were stuck at 34% test automation using Selenium. They migrated Selenium automation test scripts to testRigor’s plain English tests using Gen AI. Also, the rest of the manual test cases were transformed into stable, self-healing, plain English-based autonomous tests that are easy to maintain, using Gen AI capabilities
- Currently, IDT has 18,563 automation tests built with only 1,829 test cases left to go.
- They achieved a **90%** reduction in bugs by eliminating unexpected recurring bugs.
- Achieved over **$576K** in annual savings by switching to testRigor, delivering a **7X** return on investment.
- Cut costs by eliminating time wasted on test maintenance. Now, they spend less than **0.1%** of their time on test maintenance with testRigor.

Conclusion

As Sebastian Thrun said: “*The goal of artificial intelligence is to build machines that can think and learn like humans, but the ultimate objective is to build machines that are even better than humans at thinking and learning.”*

As software grows more sophisticated, testing must evolve to match its complexity. This paper explores how Generative AI and AI agents are transforming software testing by enabling organizations to test previously "untestable" scenarios—such as graphs, diagrams, images, user sentiment, chatbot interactions, Flutter apps, and even mainframe systems using just plain English commands.