The Generic User Approach: Training AI to Navigate Browsers as Humans Do

Authors: Paul Hanchett and Claude (Anthropic)

Abstract

Current browser automation approaches rely on brittle DOM-based interactions or pure visual approaches that do not understand the purpose and relationships of page elements. By capturing and learning from authentic browser interactions, we demonstrate how a hierarchical system of specialized AI components can develop both low-level mechanics and high-level intent understanding. We present concrete data structures for training such systems, methods for task decomposition, element recognition, and sequence prediction, and suggest a hybrid training methodology that learns from both aggregated patterns and raw interactions. Our approach promises a robust, universal solution for AI-browser interaction that remains effective even as websites change.

1. Introduction

Enabling AI systems to interact with web applications as humans do represents a significant challenge in AI development. Current approaches to browser automation typically rely on one of two strategies: DOM-based interaction using tools like Selenium and Puppeteer (Selenium Project, n.d.; Real Python, 2025) or pure visual approaches. Both suffer from fundamental limitations when applied to complex applications, particularly with dynamic web elements and changing layouts (ACCELQ, 2024).

As LambdaTest (2023) notes, "dynamic web elements can modify their attributes, positions, or visibility in response to user interactions or data updates." These challenges have led to brittle automation solutions that break whenever website layouts change. Recent innovations like Skyvern are addressing this by using "prompts in addition to computer vision and LLMs to parse items in the viewport in real-time" (Skyvern-AI, 2025), but these approaches still don't fully leverage the browser's own understanding of the page.

The key insight of the Generic User Approach (GUA) is elegantly simple: **We don't need to rebuild the browser's understanding of the page—we just need to access it.** A browser already solves the incredibly complex problem of converting HTML, CSS, and JavaScript into a visual representation with interactive elements. Rather than forcing an AI to rediscover this information, we can leverage the browser's own understanding, similar to how DeepLearning.AI (2025) describes AI browser agents using "both visual information, like screenshots, and structural data, like the HTML or Document Object Model (DOM) of a web page, to reason and take actions."

We have begun implementing this approach through a browser extension that captures user-visible page elements and will ultimately allow external entities to send keyboard and click events to the browser, just like a user at the console.

This paper makes several contributions:

1. Formalizing the GUA architecture with specialized components
2. Presenting concrete data structures for capturing browser interactions
3. Demonstrating how different models (sequence prediction, task decomposition, element recognition) can be derived from interaction data

4. Proposing a hybrid training methodology that combines aggregated patterns with raw interactions

2. The GUA Architecture

The GUA employs a dual-AI system that mirrors human cognitive specialization:

2.1 The Task Domain Specialist

A separate AI focused on the task domain (healthcare, finance, etc.):

- Understands domain-specific knowledge and goals
- Makes decisions about what needs to be accomplished
- Directs the Browser Interaction Specialist through high-level intents
- Specifies where to go and what business goals to accomplish (e.g., "Go to State Farm and buy a policy")
- Remains free from needing to understand browser interaction details

2.2 The Browser Interaction Specialist

A specialized AI system trained specifically for browser interaction:

- Understands general UI patterns and interaction models
- Translates high-level intents into specific browser interactions
- Handles all web navigation and form completion mechanics (e.g., "It's asking for your name, address, driver's license, and credit card information")
- Develops expertise in navigation patterns across different interface types
- Acts as an intermediary between domain-specific AI and the browser

2.3 Technical Implementation Components

This dual-AI system requires:

1. A browser integration layer exposing the rendered page structure
2. A semantic mapping interface translating between browser representation and AI concepts
3. An action interface for issuing interaction commands
4. A visual context providing the rendered visual state

This approach provides a separation of concerns that mirrors human cognition: just as we have specialized brain regions for visual processing and motor control separate from higher-level reasoning, this architecture separates browser interaction expertise from domain expertise.

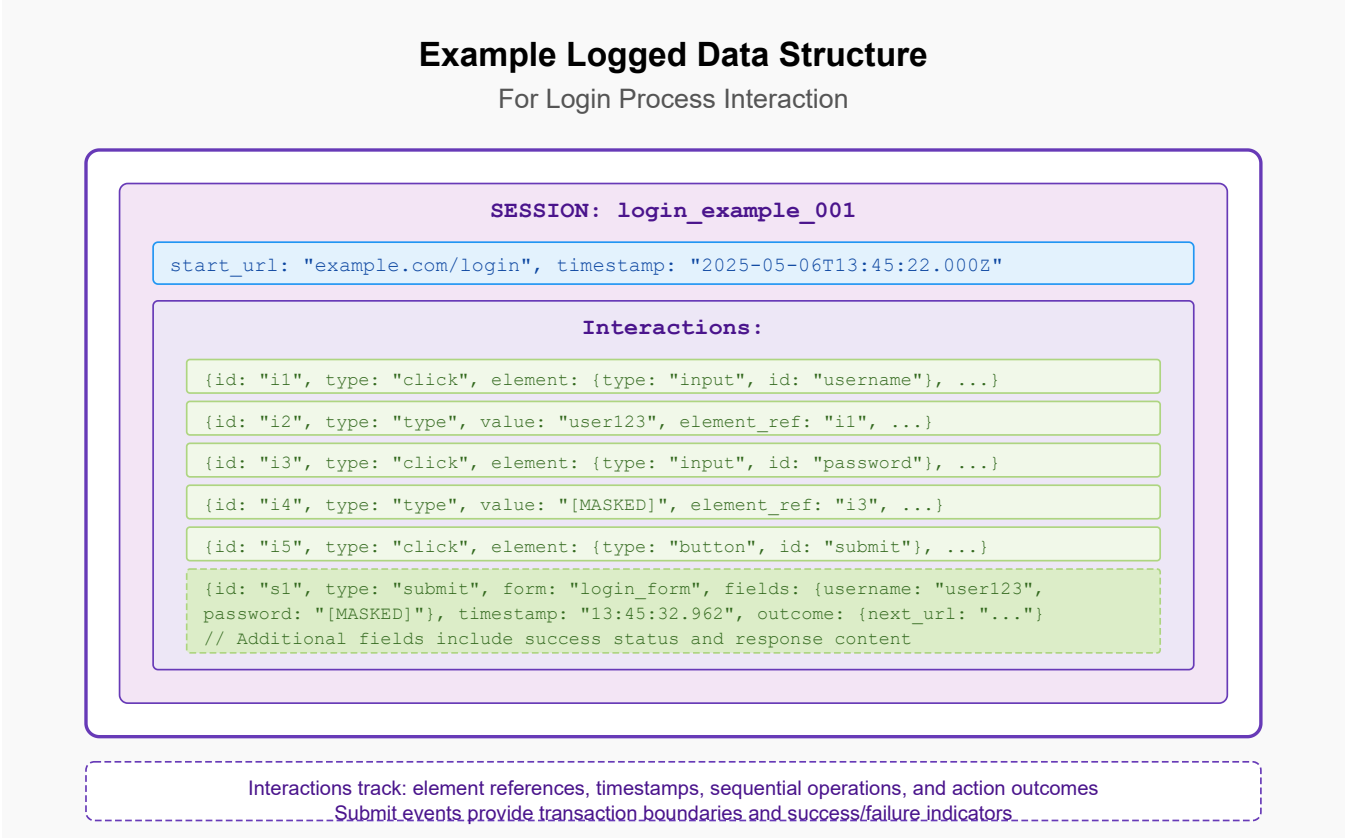
```
[Task Domain Specialist]
    ↓ High-level intents
    ↓ (e.g., "Find customer record for Smith")
[Browser Interaction Specialist]
    ↓ Specific browser actions
    ↓ (e.g., "Click search field, type 'Smith', click Search button")
[Browser Integration Layer]
    ↓ Technical implementation (clicks, types, etc.)
[Browser & Web Interface]
```

3. Training Data and Learning Methodology

Training a Generic User system requires a comprehensive approach that combines structured learning with exposure to diverse real-world interactions. Our methodology emphasizes progressive skill development rather than simply pattern recognition.

3.1 Foundational Interaction Data

Our approach begins with collecting diverse interaction data from authentic browser sessions:



This foundational data includes:

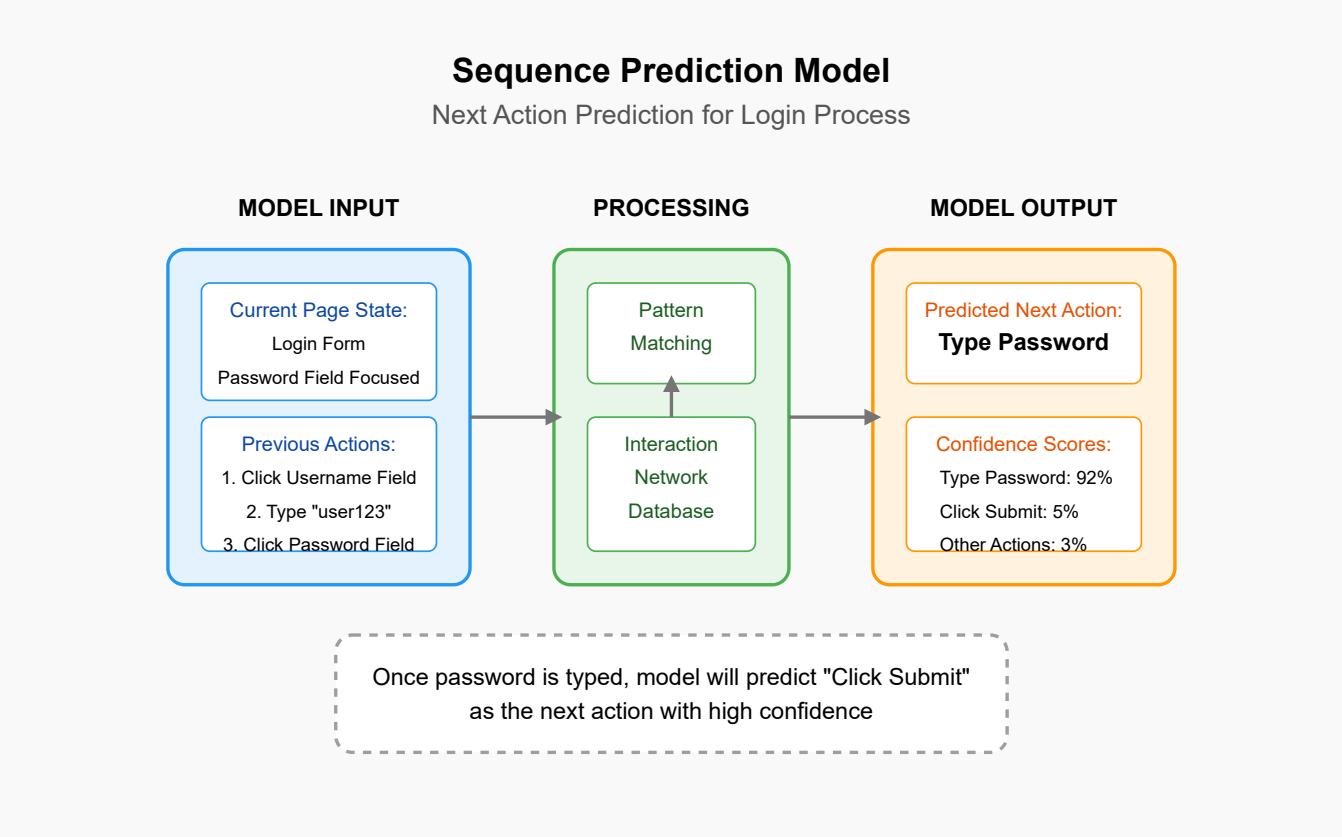
- Session information (identifier, start URL, timestamp)
- Sequence of interactions (clicks, typing, form submissions)
- Element properties and relationships
- Outcomes and success signals

This data provides the diverse real-world examples needed for robust generalization, capturing the variety of interface designs and interaction patterns found across the web.

3.2 Learning Models

From this interaction data, we derive three complementary models that work together to enable human-like browser interaction:

3.2.1 Sequence Prediction Model

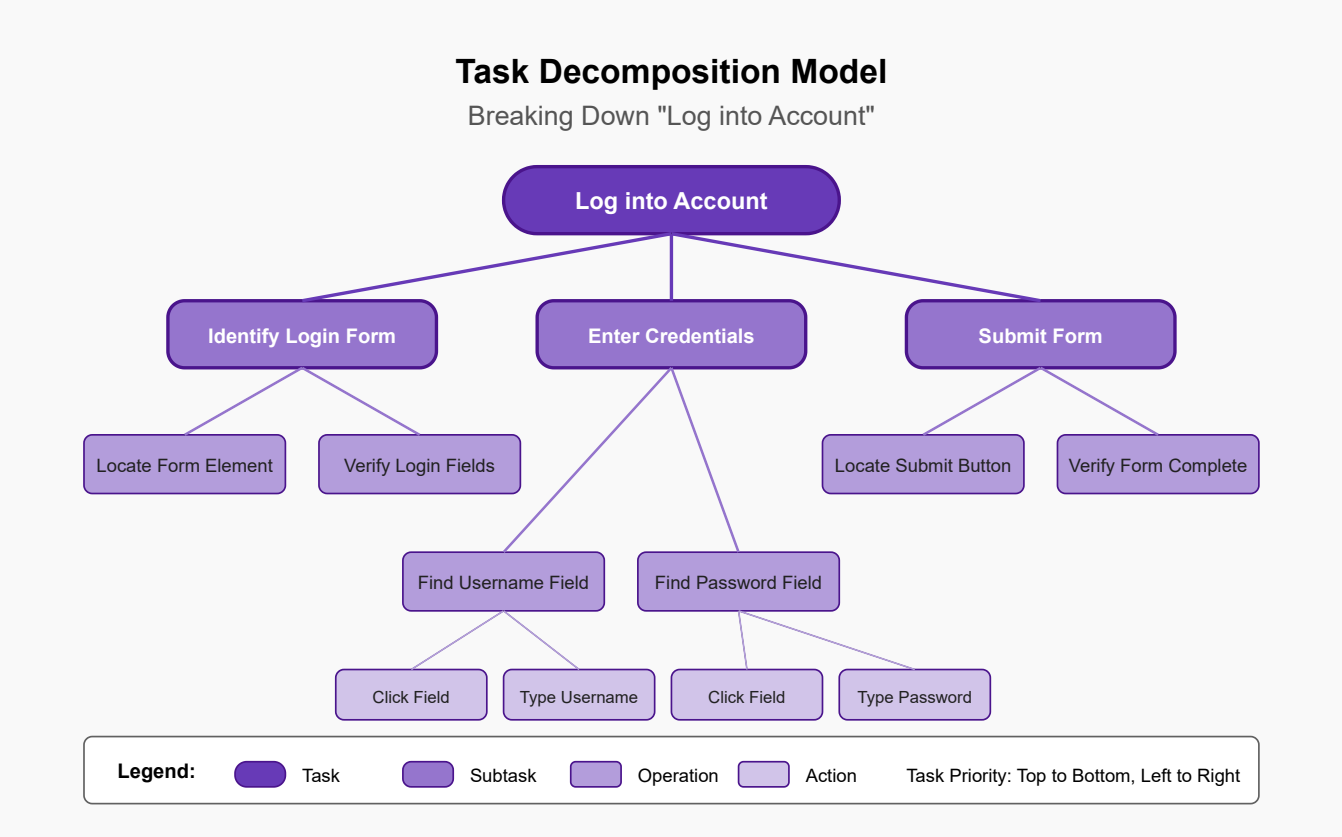


The sequence prediction model:

- Takes as input the current page state and previous actions
- Matches these against patterns in the interaction network
- Outputs the most likely next action with confidence scores

This enables the system to follow established interaction patterns and anticipate logical next steps based on context.

3.2.2 Task Decomposition Model

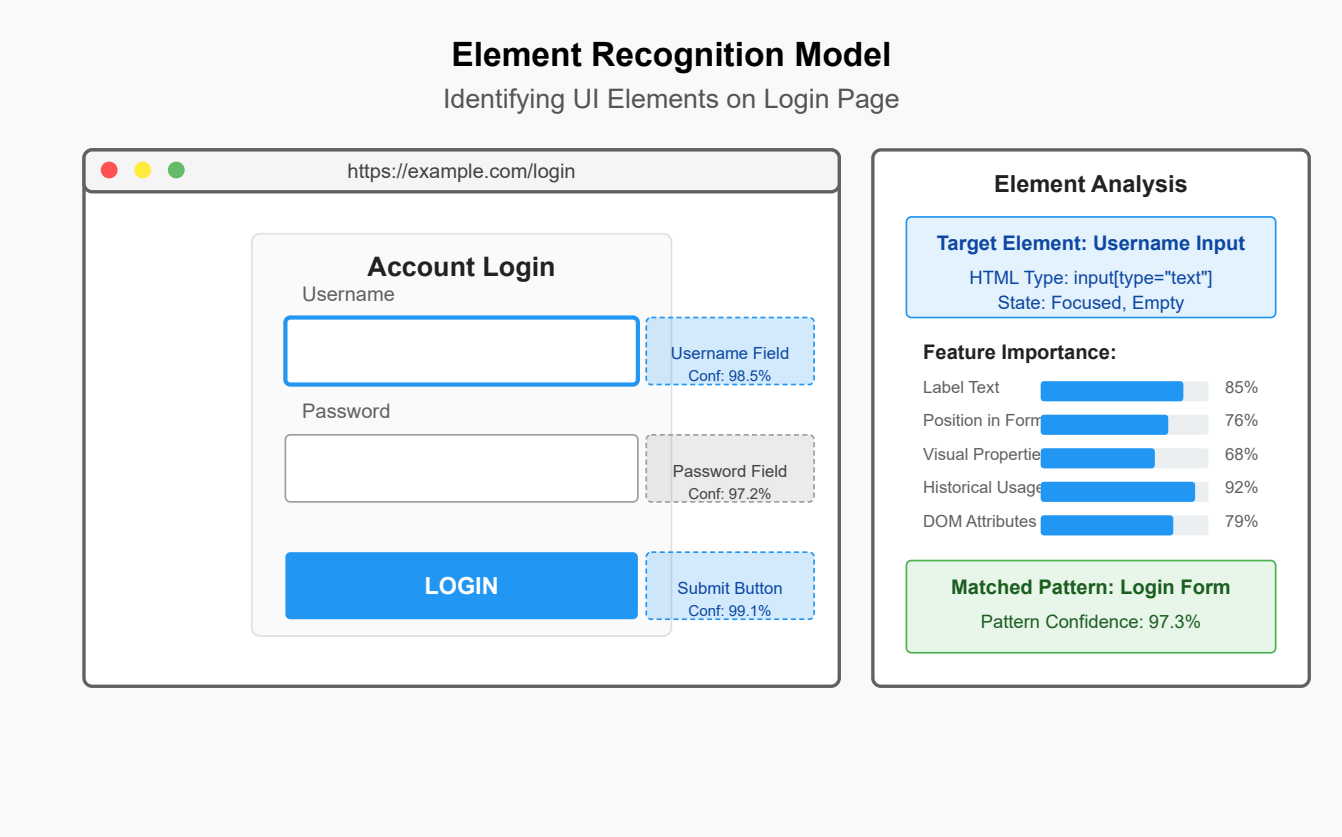


The task decomposition model:

- Takes a high-level task (e.g., "Log into account")
- Decomposes it into subtasks (Identify form, Enter credentials, Submit form)
- Further breaks these down into specific operations and actions

This hierarchical approach allows the system to plan complex interactions and adapt to different interface layouts, mirroring how humans break down complex tasks into manageable steps.

3.2.3 Element Recognition Model



The element recognition model:

- Analyzes page elements using both visual and DOM features
- Calculates confidence scores for different element types
- Identifies the most likely function of each element

This model provides the foundation for understanding what elements are available for interaction and their likely purposes.

3.3 Progressive Learning Framework

We implement a structured learning progression that mirrors human skill acquisition:

1. **Teach by Example**: Expert-guided demonstrations showing correct navigation patterns and element identification strategies, establishing foundational understanding
2. **Progressive Assisted Practice**: Partially completed tasks requiring the system to fill in missing steps with real-time feedback, gradually increasing in difficulty
3. **Increasingly Independent Practice**: Complete task execution with only goal specification, developing autonomous problem-solving capabilities
4. **Final Challenges**: Novel interfaces and tasks testing generalization capabilities across unfamiliar scenarios

This progression gradually reduces scaffolding, encouraging the development of robust, adaptable interaction strategies rather than memorization of specific patterns.

3.4 Context-Driven Element Identification

Unlike traditional approaches that attempt to pre-identify all page elements, our system follows human-like strategies for finding elements on demand. We teach the system to:

1. **Identify Elements On-Demand**: Connect field names with entry fields only when needed for the current task, not comprehensively upfront
2. **Use Spatial Relationship Heuristics**: Apply common patterns such as labels appearing to the left or above their associated fields
3. **Employ Progressive Refinement**: Start with general identification (e.g., "find the login section") before specific elements (e.g., "find the password field")

This approach is more efficient and adaptable, particularly when interfaces change or when dealing with complex pages containing many elements.

3.5 Evaluation and Validation

Throughout training and deployment, we measure performance using:

- Task completion rates across varying interface complexities
- Failure rates and error patterns
- Adaptation to interface changes and unexpected states
- Recovery capabilities when encountering ambiguity or errors
- Efficiency compared to both traditional automation and human benchmarks

These metrics help identify areas for improvement and validate that the system can effectively navigate interfaces as humans do, even in challenging scenarios.

The integration of these learning models, training approaches, and evaluation metrics creates a comprehensive system capable of developing human-like browser interaction skills.

4. Comparison with Existing Approaches

The GUA offers several advantages over current browser automation techniques, addressing many of the challenges identified in traditional methods.

Traditional DOM-based automation tools like Selenium face significant challenges with dynamic web elements. As ACCELQ (2024) notes, "automating web applications becomes challenging with Selenium since the locators available might fail to interact with the web elements." Additionally, "web content based on AJAX may take time to load, which is a possible reason for test script failure." These issues lead to brittle automation scripts that require constant maintenance.

The limitations of current approaches have led to the emergence of AI-based browser automation solutions. Azalio (2025) explains that "most existing solutions lack the flexibility to autonomously navigate websites, interpret complex UI elements, and perform multi-step tasks without breaking," noting that "even the best-performing AI models have a success rate of only 35.8% when attempting real-world web tasks." These challenges are increasingly recognized across the industry, with InfoWorld (2025) reporting on the ongoing difficulties AI agents face when attempting web interactions.

Our GUA offers several advantages over these existing techniques:

1. **Universality**: Works across any web application without application-specific programming, similar to how Skyvern-AI (2025) describes their tool as able to "operate on websites it's never seen before, as it's able to map visual elements to actions necessary to complete a workflow, without any customized code."
2. **Robustness to change**: Less sensitive to website updates by operating at the level of rendered elements rather than relying on "pre-determined XPaths or other selectors" (Skyvern-AI, 2025).
3. **Natural interaction**: Interacts with elements as they appear to users, following BrowserStack's (2024) recommendation that "Page Objects help create robust frameworks by resisting minor UI tweaks."
4. **Cognitive division of labor**: Separates browser expertise from domain expertise, similar to how DeepLearning.AI (2025) describes AI browser agents using "both visual information, like screenshots, and structural data, like the HTML or Document Object Model (DOM) of a web page, to reason and take actions."
5. **Scalable learning**: Builds generalized understanding of UI patterns that transfers across applications, addressing what Beagle Security (n.d.) describes as the need for ML algorithms that "work along with traditional web automation frameworks" to make systems "more efficient and reliable."

Recent "agentic browser" implementations like those described in the Awesome Browser Automation list (Angrykoala, n.d.) show a convergence toward similar ideas, indicating an industry recognition of the need for more human-like browser interaction models.

5. Future Directions

Several promising directions for future research include:

1. **Integration with multimodal AI**: Combining the GU approach with vision and language models
2. **Application beyond browsers**: Extending to mobile interfaces, desktop applications, and specialized systems
3. **Continuous learning**: Developing methods for ongoing improvement from user interactions
4. **Personalization**: Adapting to individual users' interaction preferences

6. Conclusion

The GUA represents a fundamental shift in how AI systems interact with web interfaces. By learning from authentic human interactions and employing a specialized AI architecture, we eliminate an entire class of brittleness in AI automation while making interactions more human-like.

The separation of browser interaction expertise from domain expertise mirrors human cognitive specialization, allowing each AI component to excel in its respective domain. As we continue to develop AI systems that assist humans in complex tasks, this dual-AI architecture offers a critical pathway to making those systems more capable, more reliable, and more adaptable to the ever-changing digital landscape.

References

1. ACCELQ. (2024). "Selenium Automation Testing Challenges." *ACCELQ Blog*. Retrieved from https://www.accelq.com/blog/selenium-automation-testing-challenges/
2. Angrykoala. (n.d.). "Awesome Browser Automation: Curated list of awesome browser automation tools and resources." *GitHub Repository*. Retrieved from https://github.com/angrykoala/awesome-browser-automation

3. Azalio. (2025). "Browser Use: An open-source AI agent to automate web-based tasks." *Azalio Blog*. Retrieved from https://www.azalio.io/browser-use-an-open-source-ai-agent-to-automate-web-based-tasks/
4. Beagle Security. (n.d.). "Machine Learning for Web Automation." *Beagle Security Blog*. Retrieved from https://beaglesecurity.com/blog/article/machine-learning-for-web-automation.html
5. BrowserStack. (2024). "Web Browser Automation using Selenium: Best Practices." *BrowserStack Guide*. Retrieved from https://www.browserstack.com/guide/selenium-web-browser-automation
6. BrowserStack. (2025). "The machine learning model is trained on a comprehensive dataset." *BrowserStack Blog*. Retrieved from https://www.browserstack.com/guide/ai-in-test-automation
7. DeepLearning.AI. (2025). "Building AI Browser Agents." *DeepLearning.AI Short Courses*. Retrieved from https://www.deeplearning.ai/short-courses/building-ai-browser-agents/
8. LambdaTest. (2023). "What Is Browser Automation: A Complete Tutorial." *LambdaTest Learning Hub*. Retrieved from https://www.lambdatest.com/learning-hub/browser-automation
9. Real Python. (2025). "Modern Web Automation With Python and Selenium." *Real Python Tutorial*. Retrieved from https://realpython.com/modern-web-automation-with-python-and-selenium/
10. Selenium Project. (n.d.). "The Selenium Browser Automation Project." *Selenium Documentation*. Retrieved from https://www.selenium.dev/documentation/
11. Skyvern-AI. (2025). "Skyvern: Automate browser-based workflows with LLMs and Computer Vision." *GitHub Repository*. Retrieved from https://github.com/Skyvern-AI/skyvern

Appendix A: Detailed Data Structures

```
// Session Example: User Login
{
  "session_id": "login_example_001",
  "start_url": "example.com/login",
  "interactions": [
    {
      "id": "i1",
      "type": "click",
      "element": {"type": "input", "id": "username",
                  "attributes": {"placeholder": "Username"}},
      "timestamp": "t1"
    },
    {
      "id": "i2",
      "type": "type",
      "value": "user123",
      "element_ref": "i1",
      "timestamp": "t2"
    },
    {
      "id": "i3",
      "type": "click",
      "element": {"type": "input", "id": "password",
                  "attributes": {"type": "password"}},
      "timestamp": "t3"
    },
```

```
    {
      "id": "i4",
      "type": "type",
      "value": "[MASKED]",
      "element_ref": "i3",
      "timestamp": "t4"
    },
    {
      "id": "i5",
      "type": "click",
      "element": {"type": "button", "id": "submit", "text": "Sign In"},
      "timestamp": "t5"
    }
  ],
  "outcome": {
    "next_url": "example.com/dashboard",
    "success": true,
    "response_contains": ["Welcome user123", "Dashboard"]
  }
}
```

Appendix B: Test Case Structure

```
[
  {
    "test_id": "login_test_001",
    "description": "Successful login with valid credentials",
    "input": {
      "url": "example.com/login",
      "page_state": {
        "elements": [
          {"id": "username", "type": "input", "visible": true, "enabled": true},
          {"id": "password", "type": "password", "visible": true, "enabled": true},
          {"id": "submit", "type": "button", "text": "Sign In",
           "visible": true, "enabled": true}
        ]
      },
      "task": "Log in with username user123 and password pass456"
    },
    "expected_actions": [
      {"type": "click", "target": "username"},
      {"type": "type", "value": "user123"},
      {"type": "click", "target": "password"},
      {"type": "type", "value": "pass456"},
      {"type": "click", "target": "submit"}
    ],
    "expected_outcome": {
      "url_contains": "dashboard",
      "page_contains": "Welcome user123"
    }
  }
]
```