Finetuning LLMs for Automatic Form Interaction on Web-Browser in Selenium Testing Framework

Nguyen-Khang Le[1], Hiep Nguyen[1], Minh Ngoc Nguyen[1], Son T. Luu[1,3], Trung Vo[1],
Quan Minh Bui[2], Shoshin Nomura[2], Le-Minh Nguyen[1]
[1]*Japan Advanced Institute of Science and Technology, Ishikawa, Japan*
[2]*Amifiable Inc., Tokyo, Japan*
[3]*VNU University of Information Technology, Ho Chi Minh City, Vietnam*

***Abstract*—Automated web application testing is a critical component of modern software development, with frameworks like Selenium widely adopted for validating functionality through browser automation. Among the essential aspects of such testing is the ability to interact with and validate web forms, a task that requires syntactically correct, executable scripts with high coverage of input fields. Despite its importance, this task remains underexplored in the context of large language models (LLMs), and no public benchmark or dataset exists to evaluate LLMs on form interaction generation systematically. This paper introduces a novel method for training LLMs to generate high-quality test cases in Selenium, specifically targeting form interaction testing. We curate both synthetic and human-annotated datasets for training and evaluation, covering diverse real-world forms and testing scenarios. We define clear metrics for syntax correctness, script executability, and input field coverage. Our empirical study demonstrates that our approach significantly outperforms strong baselines, including GPT-4o and other popular LLMs, across all evaluation metrics. Our work lays the groundwork for future research on LLM-based web testing and provides resources to support ongoing progress in this area.**

***Index Terms*—Web application testing, Large language models, Form interaction automation**

I. Introduction

Web applications are at the core of modern digital infrastructure, enabling critical services across domains such as finance, healthcare, education, and e-commerce. Ensuring their correctness and robustness is essential, as even small defects in user-facing components—especially web forms—can lead to serious usability issues or functional failures. Among the wide range of web functionalities, form interaction is particularly important and ubiquitous, facilitating tasks such as user registration, login, checkout, and data submission.

While frameworks like Selenium have become standard tools for automating web testing, they typically rely on manually authored test scripts. Writing these scripts is labor-intensive and error-prone, especially when dealing with modern, dynamic web forms that include conditional inputs, client-side validation, and real-time feedback. Despite the critical role of form testing, there exists no public dataset or benchmark that focuses on this specific task, limiting the development and evaluation of automated solutions. Moreover, existing large language models (LLMs), although capable of generating code, often struggle with producing syntactically correct, executable, and semantically accurate test cases for real-world web forms.

The emergence of large language models (LLMs), such as ChatGPT and GPT-4 [1], has enabled new capabilities in automating complex tasks across domains. In web-based environments, LLM-powered agents have been developed to perform operations such as navigation and interaction by leveraging instruction-following and tool-use capabilities [2]–[4]. Several studies have investigated LLMs in web automation settings, introducing text-based browsing interfaces and strategies for managing verbose HTML structures through simplification and restructuring [5]–[8]. WebVoyager [9] further advances this line of work by incorporating multimodal LLMs into end-to-end automated web testing pipelines.

Despite these advances, most existing approaches primarily target high-level navigation or general-purpose web exploration. In contrast, fine-grained form interaction—such as accurately filling out input fields and generating executable scripts for testing frameworks like Selenium—remains underexplored. Existing systems often lack robustness in generating test scripts with comprehensive input coverage and execution reliability. Addressing this gap is essential for advancing LLM-driven web testing toward practical integration into software development pipelines.

In this paper, we introduce a novel approach for training large language models (LLMs) to generate high-quality form interaction test cases in Selenium. Our method emphasizes three key criteria: syntax correctness, executability, and input field coverage. To support this goal, we construct and release the first form interaction testing dataset, which includes both human-written and LLM-generated examples covering diverse real-world web forms. The contributions of this paper are as follows.

- We propose a training approach for LLMs to generate executable Selenium test scripts that ensure syntax correctness and achieve high input field coverage.
- We develop a dataset for form interaction testing, combining human-annotated and synthetic web data.
- Extensive experiments demonstrate that our approach significantly outperforms strong baselines, including GPT-4o, across multiple evaluation metrics.

* Corresponding author: lnkhang@jaist.ac.jp.

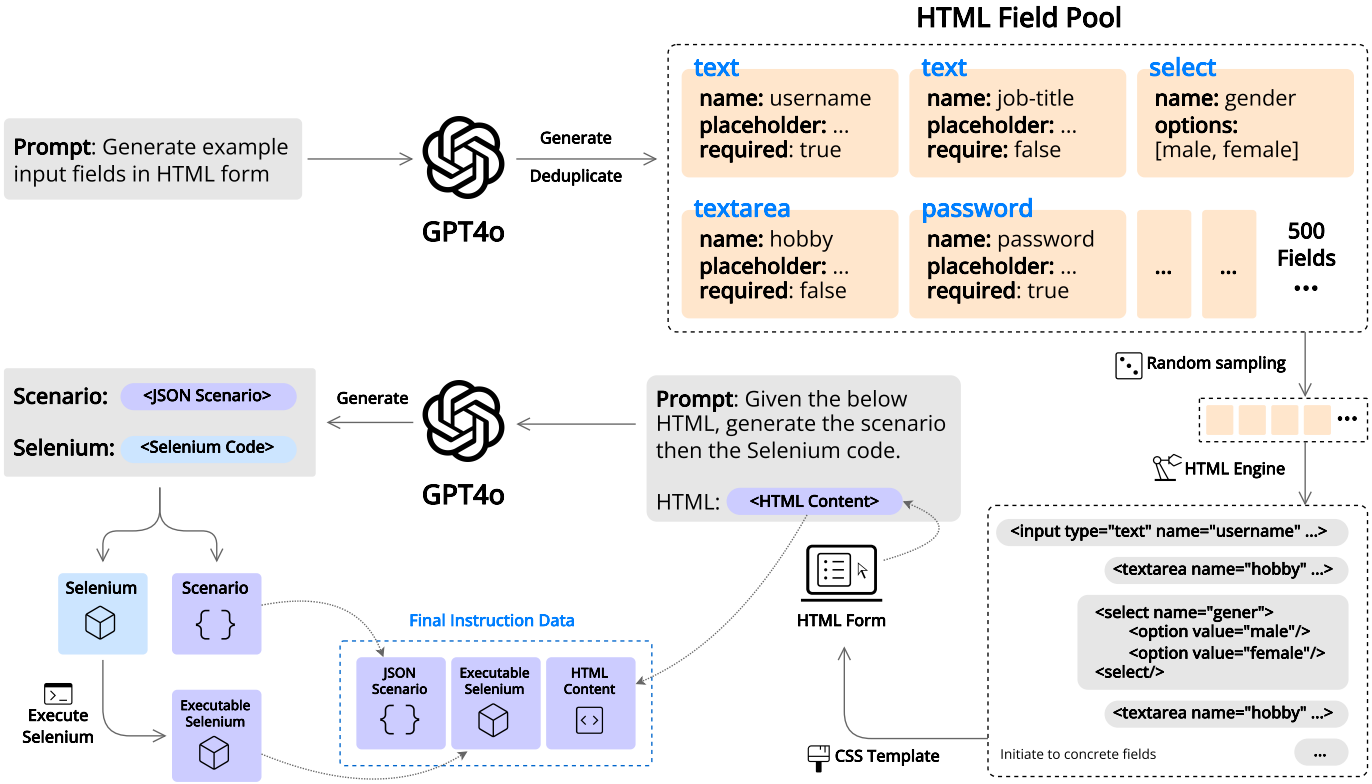


Fig. 1: Overview of the data creation process.

II. Related Work

Web application testing is a fundamental yet resource-intensive task in modern software engineering. Traditional research has focused on both static and dynamic analysis techniques to ensure the functionality and security of web systems. Early efforts, such as [10] and [11], laid the groundwork by exploring systematic approaches for improving test coverage and detecting faults through formal analysis and test case generation.

With the growing complexity of web applications, automation frameworks like Selenium have become essential tools for simulating user interactions and validating UI behavior. Despite their effectiveness, these tools typically rely on hand-crafted test scripts, which are time-consuming to author and maintain. Studies such as [12] have surveyed the inherent challenges in automating web application testing, emphasizing the need for scalable and low-maintenance solutions. More recent frameworks attempt to alleviate manual effort by recording user actions and translating them into test cases automatically [13]. While such techniques reduce scripting overhead, they still depend on access to human demonstrations and lack adaptability to unseen page structures or form behaviors.

In parallel, researchers have begun exploring the use of large language models (LLMs) to automatically generate test scripts. For instance, [14] proposed using generative models that adapt to changes in the application under test, enabling more resilient and dynamic testing workflows. WebVoyager [9] further advances this direction by deploying multimodal LLM agents to explore and interact with live websites end-to-end. However, existing LLM-based methods often struggle with key aspects of form interaction, including syntax correctness of the generated scripts, execution reliability, and comprehensive input field coverage—especially in forms with conditional logic or validation constraints.

Our work addresses these challenges by introducing a dataset-driven approach to training LLMs for form interaction testing using Selenium. Unlike prior methods that treat form interaction as an auxiliary task, we treat it as a central objective, developing human-annotated and LLM-generated datasets specifically designed for evaluating and training models on script generation quality. By measuring and optimizing for syntax correctness, script executability, and input field coverage, our method achieves robust test case generation across diverse web forms. This approach complements and extends existing research by focusing on the intersection of LLMs and practical web testing needs—particularly in the domain of automated, reliable form filling.

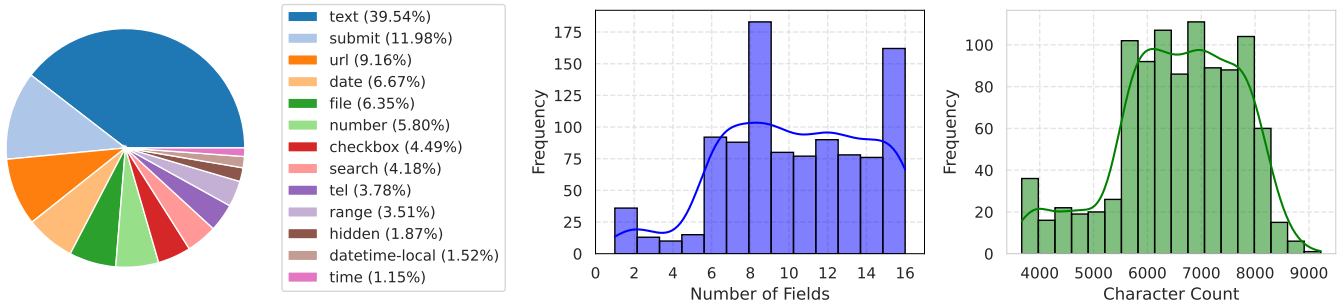
III. Methodology

A. Training Data Creation

To generate high-quality instruction data for training a large language model (LLM) to produce test scenarios and corresponding Selenium code, we design a multi-stage pipeline centered around GPT-4o, as illustrated in Figure 1.

TABLE I: Prompts used in GPT4o to generate the scenario and Selenium code.

Generate a test case scenario for filling in a given HTML form, providing step-by-step instructions, dummy data, the expected outcome. The test case scenario must be JSON-formatted. The goal is to ensure that each form element is filled appropriately, actions are conducted in the correct order. # Steps 1. **Form Analysis**: Break down the HTML form into its major fields (e.g., text inputs, radio buttons, dropdowns) and requirements. 2. **Generate JSON Test Case**: - Identify each field by its name or identifier. - Identify if the field is required and only include required fields in the test case. - Include the html snippet for each form field. - Include appropriate dummy data for each form field and make sure the date is valid and follows the field's requirements. For entering date, use format 'mm/dd/yyyy'. - Specify the instructions for filling out each field in sequential order. - Mention the expected outcome (e.g., successful submission, error messages indicating missing required fields). - Exclude the submit button and use the 'form.submit()' method for submitting forms instead of click button. # Output Format **Test Case JSON**: - Formatted in a structured JSON that describes the form fields, the data to be filled, step-by-step instructions, and the expected outcome. ### JSON Structure: [Json structure]



(a) Distribution of field types. (b) Distribution of number of fields. (c) Distribution of number of characters.

Fig. 2: Statistics of the synthetic HTML in training data.

The process begins by prompting GPT-4o to generate diverse input fields commonly found in HTML forms, such as `<input>`, `<textarea>`, `<select>`, and `<password>`. Each field is annotated with attributes such as `name`, `placeholder`, `required`, and, in the case of selection fields, a list of `options`. Redundant entries are removed to create a clean and diverse HTML Field Pool containing 500 unique form field configurations.

A random sampling strategy is then applied to select a subset of fields from this pool. These fields are assembled into a concrete HTML form using an HTML engine and styled with a predefined CSS template to enhance realism. Figure 2 shows the statistics of the generated HTML. This generated HTML form serves as the input to a second GPT-4o prompt.

B. Test Scenario and Selenium Generation

We prompt GPT-4o to simultaneously generate both the test scenario and the corresponding Selenium code from a given HTML form. Table I shows the prompts used in GTP-4o. GPT-4o then generates two outputs: (1) a natural language scenario represented in JSON format and (2) the corresponding Selenium code that automates interaction with the form. The scenario is structured in JSON format and includes a detailed description of the user action, the relevant input fields, their identifiers (e.g., by `id`, by `name`), the interaction method, the instruction, and the expected outcome. Based on this structured scenario, GPT-4o proceeds to generate the Selenium script. The generated code is then executed using a web engine; scripts that fail to execute are discarded, while only executable ones are retained for further use.

The final instruction data consists of three components: the HTML content, the scenario in JSON, and the executable Selenium code. This structured and realistic dataset enables the LLM to learn to generate both test scenarios and automation scripts grounded in actual web content. The key advantage of this process lies in its ability to synthesize high-quality, executable, and semantically consistent data by leveraging LLMs and controlled sampling mechanisms.

IV. Experiments

We evaluate our approach on the task of generating Selenium scripts for form interaction testing. Our experiments assess the ability of large language models (LLMs) to produce scripts that are syntactically correct, executable, and achieve high coverage of input fields. Below, we describe the models, datasets, and evaluation metrics used in our study.

A. Models

GPT Family: We compare our approach with the GPT family models, including *GPT4o* and *GPT-OSS* (20B and 120B versions), which serve as a strong baseline for code generation.

State-of-the-art LLMs: We use the families of *Qwen2.5* [15], *Qwen3* [16], and *Llama3.1* [17], which are the recent state-of-the-art open-source language models with strong performance on code generation tasks. We employ their instruction-tuned variants and fine-tune them on our synthetic data of form interaction. We compare the performance of the original model with the model finetuned using our approach.

Code generation LLMs: We use the model *Qwen2.5 Coder Instruct* [18]. We fine-tune them on our synthetic data of form interaction. We compare the performance of the original model with the model finetuned using our approach.

B. Datasets

We evaluate the models on two test sets:

- **Synthetic Forms:** A curated collection of 500 HTML forms programmatically generated to cover a wide range of input types (e.g., `text`, `email`, `password`, `checkbox`, `radio`, `select`), layout patterns, and interaction complexity (e.g., required vs. optional fields, conditional logic).
- **Real-World Forms:** 133 web forms collected from publicly available websites, including login, registration, and contact forms. These forms reflect real-world HTML structure diversity and domain-specific constraints.

C. Evaluation Metrics

To measure the quality of generated scripts, we use the following three metrics:

- **Syntax Correctness:** Whether the generated script is syntactically valid. We validate Selenium scripts using language-specific syntax checkers.
- **Executability:** Whether the script runs without errors in a headless browser environment (e.g., Chromium). This checks runtime correctness and functional validity.
- **Input Coverage:** The proportion of input fields in the form that are correctly identified and filled by the script. This metric captures the completeness of the test case in terms of form interaction.

V. Results

A. Main Results

Table II presents the performance comparison on real-world form interaction testing. Across different model families, our approach consistently improves executability and input coverage compared to the original baselines, while maintaining high syntax correctness. The filtering strategy proves effective in eliminating noisy or unexecutable samples, leading to more reliable script generation that runs successfully and covers a broader range of input fields. Importantly, the improvements are observed across diverse backbones, including Qwen, Llama, and code generation models, demonstrating that our method generalizes well beyond a single model family. These results confirm the effectiveness of our approach in enhancing LLM-based form interaction testing in realistic web environments.

Table III reports the performance on synthetic form interaction testing. Across all models, our approach consistently improves executability and input field coverage compared to the original settings, while maintaining perfect syntax correctness. This indicates that our approach leads to more reliable script generation that not only runs successfully but also achieves broader coverage of input fields. Notably, the improvements hold across different model backbones, demonstrating the robustness and generality of our approach in enhancing LLM-based form interaction testing.

B. Analysis of Error rate in different field types

Figure 3 shows the percentage of form field errors across different HTML field types for the GPT-4o model, and baseline LLMs supervised fine-tuning with our approach (Filtered SFT), and fine-tuning without filtering, where all Selenium code is used (Full SFT). In Qwen2.5-14B-Instruct (Figure 3a), all three models exhibit relatively high error rates in complex field types such as `text`, `dropdown`, and `radio`, while achieving low error rates on simpler or infrequently used fields like `tel`, `textarea`, and `hidden`. The Filtered SFT variant displays a more even distribution of errors, reducing failure in common fields such as `submit` and `radio` compared to the Full SFT baseline. This suggests that data filtering may help reduce overfitting to noisy or rare patterns. GPT-4o shows overall lower error percentages and maintains consistency across field types, serving as a reference point for generalization performance.

In the Qwen2.5-7B-Instruct (Figure 3b) and Qwen3-14B-Instruct (Figure 3c), similar trends emerge, but with sharper contrasts between field types. Notably, Full SFT and Filtered SFT both struggle significantly with the `other` and `text` categories, indicating that smaller models may be more sensitive to ambiguous field semantics. The Filtered SFT variant again shows improved robustness in fields such as `submit` and `radio`, although its error rate on `dropdown` slightly increases compared to the Full SFT model. Interestingly, GPT-4o's error pattern remains stable across both scales, reinforcing the observation that model size and fine-tuning strategy can introduce distinct biases in field-specific performance. Comparing the two model scales, Qwen2.5-14B exhibits more stable performance across field types, while Qwen2.5-7B shows more pronounced spikes in error for specific categories.

C. Analysis of Causes of Errors

Across both model scales and all training regimes, the dominant failure mode underlying the per–field-type error profiles in Figure 3 is *inconsistent identification of the intended target field* within the active interaction context (typically, the currently focused form). Concretely, the model often predicts a correct *semantic intent* (e.g., "fill email"), yet binds that intent to an *incorrect DOM element*. This manifests as: (i) selecting a visually adjacent but non-interactable wrapper (e.g., a `<div>` around an `<input>`); (ii) selecting a hidden or disabled control; (iii) picking the wrong member of a group (`radio`/`checkbox`); (iv) confusing native `<select>` with

TABLE II: Performance comparison on real-world form interaction testing. OUR APPROACH is evaluated under two settings: with filtered training data that excludes unexecutable Selenium code; and without filtering (w/o filter), where all Selenium code is used for training.

Model	Run	Syntax Correctness (%)	Executability (%)	Input Coverage (%)
		GPT Family (Baseline)		
GPT-4o	Original	97.67	50.38	39.28
GPT-OSS (120B)	Original	100	45.74	25.07
GPT-OSS (20B)	Original	91.47	46.51	23.74
		Qwen2.5 Family of models		
Qwen2.5-14B (Instruct)	Original	99.22	49.61	39.35
	OUR APPROACH (w/o filter)	**100.00**	58.91	39.79
	OUR APPROACH	99.22	**59.69**	**40.53**
Qwen2.5-7B (Instruct)	Original	99.22	48.84	4.96
	OUR APPROACH (w/o filter)	59.38	23.26	16.26
	OUR APPROACH	100.00	48.84	23.68
		Qwen3 Family of models		
Qwen3-14B (Instruct)	Original	97.67	48.06	**37.83**
	OUR APPROACH (w/o filter)	98.44	60.47	33.15
	OUR APPROACH	**100.00**	**60.47**	35.96
Qwen3-8B (Instruct)	Original	100.00	17.24	19.12
	OUR APPROACH (w/o filter)	98.41	28.57	27.38
	OUR APPROACH	**100.00**	25.20	21.44
		Llama Family of models		
Llama3.1-8B (Instruct)	Original	94.57	34.11	0.45
	OUR APPROACH (w/o filter)	**99.22**	39.53	34.24
	OUR APPROACH	98.45	**45.74**	**36.80**
		Code Generation Models		
Qwen2.5-7B-Coder (Instruct)	Original	96.80	56.80	10.89
	OUR APPROACH (w/o filter)	**100.00**	57.85	**20.12**
	OUR APPROACH	98.31	**66.10**	15.40

TABLE III: Performance comparison on Synthetic Form interaction testing, measured on syntax correctness (Syntax), executability (Execute), and input coverage (Cover).

Model	Setting	Syntax	Execute	Cover
GPT-4o	API	99.60	74.68	76.40
Qwen2.5-14B-Inst	Original	96.40	77.40	77.97
	OUR APPROACH	100.00	**82.00**	**86.38**
Qwen3-14B	Original	100.00	65.20	85.30
	OUR APPROACH	100.00	**81.60**	**87.18**

a custom dropdown built from `<div>/<li>`; or (v) leaving the form scope and hitting a similarly named field elsewhere on the page. Typical Mis-Bind patterns include:

- **Text inputs** (text, email, number): Choosing a placeholder -only field, a hidden clone used for masking, or a sibling used for validation.
- **Dropdowns**: Treating a custom combobox as a native `<select>` (or vice versa).
- **Radio/Checkbox groups**: Clicking the first matching input by `@type` or `@name`, ignoring the label/value.
- **Submit**: Clicking a non-submit button (e.g., a reset or a tab switch) or a disabled button before client-side validation completes.
- **Hidden/Offscreen**: Selecting the elements with attributes `display:none`, `visibility:hidden`, zero-size, or off-canvas.
- **Context leakage**: Leaving the current form and interacting with a homonymous field elsewhere (e.g., header search).

D. Examples of Faulty Selenium Snippets and Implications

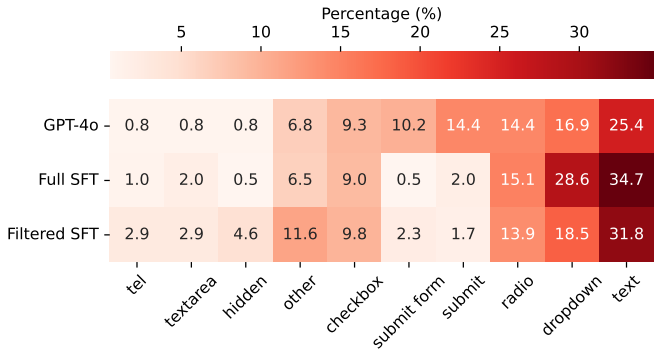
To better illustrate how incorrect field identification translates into faulty automation scripts, we analyze common failure patterns observed in generated Selenium code. The following examples reflect real mis-bindings found in the evaluation set, highlighting the subtle but critical mismatches between semantic intent and DOM binding.

- **Example 1: Misbinding to Wrapper instead of Input**

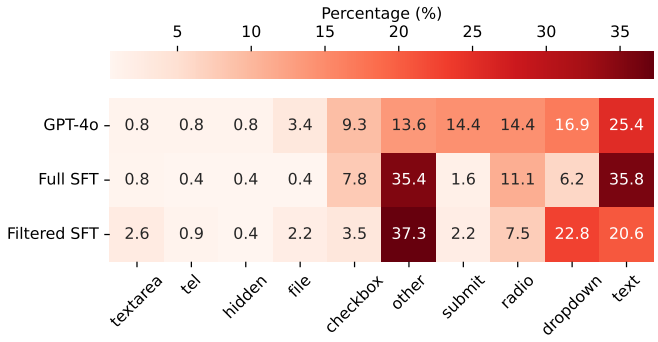
```
driver.find_element(By.XPATH,
  "//div[@id='mail']").send_keys("a@a")
```

  This line targets a `<div>` that visually wraps the input field (e.g., for styling or layout). Since `<div>` is not an interactable input element, this action raises a `ElementNotInteractableException`.
- **Example 2: Selecting Disabled or Hidden Field**

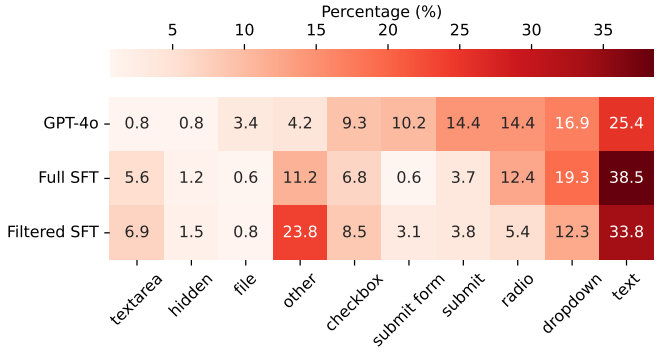
```
driver.find_element(By.CSS_SELECTOR,
  "input[name='tel']").send_keys("123")
```



(a) **Qwen2.5-14B-Instruct**'s Percentage of error in each field type.



(b) **Qwen2.5-7B-Instruct**'s Percentage of error in each field type.



(c) **Qwen3-14B-Instruct**'s Percentage of error in each field type.

Fig. 3: Percentage of errors across field types for various LLMs. "Filtered SFT" refers to models fine-tuned with our filtering step, where unexecutable Selenium code is excluded, while "Full SFT" denotes fine-tuning without filtering, using all Selenium code for training.

The matched element exists in the DOM but is either `disabled` or hidden (e.g., `display:none`). No visible feedback or state change occurs on the page.

- **Example 3: Context Leakage**

```
driver.find_element(By.NAME,
  "search").send_keys("query")
```

If multiple fields share the same name (e.g., one in the form, one in the navbar), the global selector may match the wrong instance.

VI. Conclusions

This work presents a comprehensive framework for form-centric web automation using large language models, addressing a critical yet underexplored area in automated web testing. By introducing both a novel data generation pipeline and robust evaluation metrics, we enable systematic training and assessment of LLMs on form interaction tasks. Our curated datasets and empirical results demonstrate that LLMs can be effectively adapted to generate syntactically correct, executable, and high-coverage Selenium scripts. The significant performance improvements over strong baselines highlight the potential of LLMs in this domain. We hope our contributions—benchmarks, datasets, and methodology—serve as a foundation for further research into LLM-based software testing and broader applications in web automation.

References

[1] OpenAI, "Gpt-4 technical report," *arXiv preprint*, 2023.

[2] AutoGPT, "Autogpt," 2022.

[3] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, and e. a. Bill Qian, "Toolllm: Facilitating large language models to master 16000+ real-world apis," *arXiv preprint*, 2023.

[4] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *arXiv preprint*, 2023.

[5] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, and e. a. William Saunders, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint*, 2021.

[6] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, Y. Bisk, D. Fried, and e. a. Uri Alon, "Webarena: A realistic web environment for building autonomous agents," *arXiv preprint*, 2023.

[7] I. Gur, H. Furuta, A. Huang, M. Safdari, Y. Matsuo, D. Eck, and A. Faust, "A real-world webagent with planning, long context understanding, and program synthesis," *arXiv preprint*, 2023.

[8] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2web: Towards a generalist agent for the web," *arXiv preprint*, 2023.

[9] H. He, W. Yao, K. Ma, W. Yu, Y. Dai, H. Zhang, Z. Lan, and D. Yu, "WebVoyager: Building an end-to-end web agent with large multimodal models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 6864–6890.

[10] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *Proceedings of the International Conference on Software Engineering*, vol. 25, no. 3, 2001, pp. 25–34.

[11] A. Andrews, J. Offutt, and R. Alexander, "Test generation for web applications," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 187–202, 2005.

[12] A. Marchetto, P. Tonella, and F. Ricca, "A survey on web application testing," *ACM Computing Surveys*, vol. 44, no. 3, pp. 1–36, 2012.

[13] J. Smith and R. Taylor, "Automated frameworks for dynamic web testing," *Software Testing Journal*, vol. 37, no. 1, pp. 45–67, 2022.

[14] K. Lee and S. Johnson, "Leveraging generative ai for automated test case creation," in *Proceedings of ICSE*, 2022, pp. 198–207.

[15] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[16] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[17] A. G. et al., "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[18] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, K. Dang, Y. Fan, Y. Zhang, A. Yang, R. Men, F. Huang, B. Zheng, Y. Miao, S. Quan, Y. Feng, X. Ren, X. Ren, J. Zhou, and J. Lin, "Qwen2.5-coder technical report," 2024. [Online]. Available: https://arxiv.org/abs/2409.12186