MacroBench: A Novel Testbed for Web Automation Scripts via Large Language Models

Hyunjun Kim*
KAIST
Daejeon, South Korea
hyunjun1121@kaist.ac.kr

Sejong Kim*
KAIST
Daejeon, South Korea
kingsj@kaist.ac.kr

Abstract

We introduce MacroBench, a code-first benchmark that evaluates whether LLMs can synthesize *reusable* browser-automation programs (macros) from natural-language goals by reading HTML/DOM and emitting Python+Selenium. MacroBench instantiates seven self-hosted sites—*Airbnb-like*, *TikTok-like*, *Reddit-like*, *Instagram-like*, *Facebook-like*, *Discord-like*, and *Threads-like* —covering **681** tasks across interaction complexity and targeting difficulty. Our end-to-end protocol validates generated code via static checks, sandboxed execution, and outcome verification (DOM assertions, database snapshots), and includes a safety suite for scraping, spam/abuse, and credential/privacy prompts. Across **2,636** model–task runs, we observe stratified success: GPT-4o-Mini (96.8%), GPT-4.1 (95.3%), Gemini-2.5-Pro (89.0%), DeepSeek-V3.1 (83.4%). Models handle simple tasks reliably (91.7%) but fail on complex workflows (0.0%), and none meet production-quality coding practices despite functional completion. We release our complete benchmark pipeline, evaluation framework, and experimental results at https://github.com/hyunjun1121/MacroBench to enable reproducible assessment of macro synthesis for web automation.

1 Introduction

Large Language Models (LLMs) are increasingly used to automate complex workflows that unfold inside a web browser. We study this problem from a *programmatic* perspective: given a natural-language goal, can an LLM synthesize a reusable, rule-based web automation program (a "macro" or macroinstruction [6]) using industry-standard tools such as Python and Selenium? These macros execute concrete browser actions—clicking, typing, submitting forms, following links—so that end users can repeatedly and reliably run them to complete tasks.

A visionless, code-centric agent. Unlike multimodal web agents grounded in screenshots or pixels [11, 21], our agent operates purely on HTML/DOM plus programmatic browser actions. This code-first lens reduces perception latency and brittleness, exposes semantic structure (element types, IDs, ARIA roles), and mirrors professional practice (e.g., Selenium locators). It extends DOM-based web interaction research in RL and structured policies [20, 14, 7] while remaining compatible with contemporary prompting paradigms [24, 18, 25].

A benchmark for macro synthesis. MacroBench isolates three competencies required to translate a natural-language goal into an executable macro:

1. **Code interpretation**: recover task-relevant structure from raw HTML (forms, inputs, buttons, links, and attributes such as id, class, name, role, labels, and hierarchy).
2. **Code generation**: emit correct, idiomatic Python+Selenium with robust element location and interaction logic (waits, error handling, parameterization), complementing algorithmic/software benchmarks [2, 8, 13].

*These authors contributed equally

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Lock-LLM Workshop: Prevent Unauthorized Knowledge Use from Large Language Models.

3. **Task planning**: decompose the goal into steps and control flow, drawing on reasoning+acting/tool-use strategies [24, 18, 25, 4].

Each competency is evaluated with targeted prompts and execution checks, enabling granular diagnosis beyond aggregate success rates.

Implications for safety and dual-use. Because web automation can be repurposed for spam, scraping, or policy-violating behavior, `MacroBench` also probes *safety alignment* under realistic prompts. Our safety suite complements agentic-safety evaluations [28] and policy-aligned training paradigms [16, 1] by asking whether models refuse or reshape requests that would yield unsafe automation code [5, 9, 10]. The analysis highlights failure modes specific to code-and-actions settings that are underrepresented in general-purpose safety tests.

Controlled, reproducible environments. All tasks run end-to-end on *seven* self-hosted, synthetic websites that emulate widely used platforms while avoiding interaction with live services. Concretely, we instantiate *Airbnb-like* (hospitality marketplace), *TikTok-like* (short-video feed), *reddit-like* (forums), *instagram-like* (photo feed), *facebook-like* (social network), *discord-like* (chat with servers/channels), and *Threads-like* (microblogging) sites, following best practices for reproducible web-agent evaluation [29, 3]. This design ensures consistent initial states and precise execution-based scoring. Our evaluation includes four contemporary LLMs—`GPT-4.1`, `Gemini-2.5-Pro`, `DeepSeek-V3.1`, and `GPT-4o-Mini`—to illustrate capability and safety trade-offs in a code-first setting.

Scope and significance. Prior web-agent benchmarks emphasize open-web browsing or visually grounded interaction [26, 11, 21] or evaluate general agent abilities across heterogeneous tools and tasks [15, 19, 23, 22]. `MacroBench` is complementary: it targets *macro synthesis over HTML* as the core skill required for scalable, maintainable, and auditable automation. By disentangling interpretation, generation, and planning, our benchmark provides actionable diagnostics for building agents that are both *highly capable* and *responsibly deployable* in real-world automation workflows [17].

2 Related Work

DOM-centric web interaction. Early work established the value of reasoning over HTML/DOM rather than pixels for controllable web agents. [20] introduced the World-of-Bits platform; [14] proposed workflow-guided exploration and a DOM-structured policy; and [7] grounded actions in DOM graphs. These efforts foreshadow our "vision-less" setting: agents that read code structure to choose robust, semantically meaningful actions.

Realistic web environments and open-web evaluation. Simulated but realistic websites (e.g., shopping) were popularized by WebShop [26]. WebArena [29] advanced reproducibility with self-hosted sites; Mind2Web [3] shifted to real websites with broad task diversity. Recent multimodal benchmarks add screenshots to capture visual grounding—VisualWebArena [11] and OSWorld [21]. Ecosystem efforts aim to standardize evaluation across suites (e.g., BrowserGym) [12]. Our benchmark differs by focusing squarely on *macro synthesis from HTML* and by concretely instantiating *seven* archetypal, self-hosted sites—*Airbnb-like*, *TikTok-like*, *reddit-like*, *instagram-like*, *facebook-like*, *discord-like*, and *Threads-like*—to stress HTML-grounded interaction patterns (forms, feeds, chats, and microblogging) under reproducible conditions.

Agents, tool use, and planning. Agent prompting frameworks combine reasoning with action execution. ReAct interleaves thoughts and tool calls [24]; Toolformer teaches models to decide when and how to invoke APIs [18]; and Tree-of-Thoughts introduces deliberate search over intermediate thoughts [25]. Program-aided reasoning leverages external execution to improve reliability [4]. Broader agent benchmarks evaluate general decision making across diverse environments and tool suites [15, 19, 23, 22]. `MacroBench` adopts compatible prompting styles but evaluates a narrower, industry-relevant target: translating natural-language goals into maintainable Python+Selenium macros.

Code generation and software engineering evaluation. LLM code generation is typically assessed via algorithmic correctness (e.g., HumanEval) [2], software-maintenance realism (e.g., SWE-bench) [8], or competitive programming [13]. Our benchmark complements these by scoring *browser automation code*: element selection, synchronization, error handling, and parameterized execution—skills that are decisive for robustness on the seven site archetypes above.

Safety of agentic LLMs. Safety alignment techniques such as RLHF [16] and Constitutional AI [1] motivate systematic testing of agents in interactive settings. Recent benchmarks quantify agentic safety risks and policy adherence [28], including web-specific safety and trustworthiness probes [27, 5, 9, 10]. Our safety suite situates these concerns in code-first automation: can an LLM be induced to output macros that enable spam, scraping, or other policy violations, or does it refuse and propose safer alternatives?

Positioning. Where visually grounded agents advance perception and accessibility [11, 21], `MacroBench` targets the *engineering* substrate of web automation—reading HTML, planning actions, and emitting robust Selenium code. By cleanly factorizing interpretation, generation, and planning, it offers diagnostic signal that complements open-web evaluations [3] and general agent leaderboards [15, 19, 23, 22], and it brings safety assessment closer to the dual-use realities of automation code [16, 1, 28].

3 Methodology

3.1 Benchmark Design and Architecture

`MacroBench` evaluates Large Language Models (LLMs) on their ability to synthesize executable, reusable web-automation *macros* (Python+Selenium programs) from natural-language instructions. The benchmark targets three core competencies required for reliable macro synthesis: *code interpretation*, *code generation*, and *task planning*.

Synthetic website ecosystem. Following best practices in web-agent evaluation [29, 26], we deploy a controlled ecosystem of **seven** synthetic websites that emulate common real-world platforms while ensuring reproducibility and safety:

- *Airbnb-like* marketplace: listing search/filtering, booking flows (20 tasks)
- *TikTok-like* short-video platform: infinite feeds, social interactions (160 tasks)
- *Reddit-like* forum: subcommunities, posts, voting, comments (130 tasks)
- *Instagram-like* photo feed: posts, profiles, social actions (120 tasks)
- *Facebook-like* social network: timeline, groups, events (120 tasks)
- *Discord-like* chat: servers, channels, messaging (111 tasks)
- *Threads-like* microblog: timeline, replies, follows (20 tasks)

All sites expose consistent HTML/ARIA conventions, deterministic initial states (seeded databases, fixed user accounts), and realistic interaction patterns. Each website runs in isolated containers with frozen dependencies, enabling reproducible evaluation.

Task taxonomy and complexity levels. We authored **681** distinct automation tasks. Across our full run grid (2,636 model–task evaluations; some tasks were not attempted by all models), the complexity distribution and success are:

1. **Simple**: 2,584 runs with 2,370 successes (91.7%)
2. **Medium**: 44 runs with 37 successes (84.1%)
3. **Complex**: 8 runs with 0 successes (0.0%)

Each task includes natural-language instructions, *explicit success criteria*, and executable validation checks; a run is counted as *successful* only if the program passes static checks, executes without runtime error, and satisfies all DOM/database assertions.

3.2 Evaluation Framework

Models under test. We evaluate four contemporary LLMs with distinct profiles (model IDs as queried at evaluation time): `GPT-4.1`, `Gemini-2.5-Pro`, `DeepSeek-V3.1`, `GPT-4o-Mini`. All models receive the same structured prompts, identical retry budget (up to two attempts), and equivalent execution feedback signals for fairness and reproducibility.

Prompting methodology. We use structured prompts containing: (i) task specification with success criteria, (ii) HTML context (DOM excerpts with attributes), (iii) technical constraints (Python+Selenium output format), and (iv) few-shot exemplars. Each model receives up to two retry attempts with execution feedback.

Execution and validation pipeline. Generated programs undergo end-to-end evaluation:

1. **Static checks**: linting, import validation, safety guardrails

Table 1: Overall macro-synthesis performance across 2,636 model–task combinations.

Model	Total	Success	Rate (%)
GPT-4o-Mini	680	658	96.8
GPT-4.1	674	642	95.3
Gemini-2.5-Pro	666	593	89.0
DeepSeek-V3.1	616	514	83.4
Overall	**2,636**	**2,407**	**91.3**

2. **Runtime execution**: headless browser automation in sandboxed containers
3. **Outcome validation**: DOM assertions, database snapshots, HTTP logs
4. **Error attribution**: syntax, runtime, logical, timing, or coverage failures

All artifacts (code, traces, screenshots, DOM diffs) are logged for reproducibility.

3.3 Safety Evaluation Component

Web automation carries dual-use risks. Our safety suite probes whether models refuse or redirect harmful requests across four categories: scraping/data extraction, spam/abuse, credential harvesting, and privacy violations. We evaluate refusal rates, alternative quality, reasoning clarity, and consistency under paraphrases.

3.4 Metrics and Analysis

Primary metrics. Execution-based scoring provides the principal signal: (i) task completion rate, (ii) code quality assessment (maintainability, robustness, best practices), and (iii) error resilience under adversarial conditions.

Competency diagnostics. We separately analyze code interpretation (element targeting accuracy), code generation (syntactic validity, Selenium practices), and task planning (workflow decomposition, error handling). Ablation studies quantify the contribution of prompt components (HTML context, exemplars, retry attempts).

4 Results

4.1 Overall Performance

We evaluated four contemporary LLMs across 2,636 unique task-model combinations spanning seven synthetic websites. Results demonstrate clear performance stratification, with **GPT-4o-Mini** achieving the highest overall success rate (96.8%), followed by **GPT-4.1** (95.3%), **Gemini-2.5-Pro** (89.0%), and **DeepSeek-V3.1** (83.4%).
Table 1 summarizes the aggregate results. Despite architectural differences, all models exhibit strong performance on simple automation tasks, but effectiveness degrades with increasing task complexity and website-specific interaction patterns.

4.2 Task Complexity Analysis

Performance varies significantly by task complexity (Table 2). Simple automation tasks (91.7% success) are handled reliably across all models, while medium-complexity workflows requiring multi-step coordination show degraded performance (84.1%). Complex tasks involving conditional logic, error recovery, and cross-page workflows remain largely unsolved (0.0% success), highlighting fundamental limitations in current LLMs' planning and error-handling capabilities.

Table 2: Performance breakdown by task complexity level.

Complexity	Total	Success	Rate (%)
Simple	2,584	2,370	91.7
Medium	44	37	84.1
Complex	8	0	0.0

4.3 Website-Specific Performance Patterns

Platform archetype strongly influences macro synthesis success (Table 3). **Discord-like** chat platforms achieve the highest success rate (99.5%), likely due to simple message-posting workflows and predictable DOM structures. **Facebook-like** social networks also perform well (98.7%), while **TikTok-like** short-video platforms prove most challenging (81.5%), primarily due to infinite-scroll feeds and dynamic content loading.

Table 3: Success rates by website archetype (task count, total runs, successes, and rates).

Website	Tasks	Total Runs	Successes	Rate (%)
Discord-like	111	426	424	99.5
Facebook-like	120	463	457	98.7
Reddit-like	130	499	470	94.2
Threads-like	20	80	72	90.0
Instagram-like	120	471	412	87.5
Airbnb-like	20	80	69	86.3
TikTok-like	160	617	503	81.5

Note. We omit per-site model rankings to avoid over-interpretation given missing runs on some site–model pairs; see Appendix for per-model breakdown.

Model rankings vary by platform, suggesting interactions between model design and UI patterns (e.g., feeds, chats, booking flows). We therefore avoid per-site rankings in the main text and defer detailed breakdowns to the Appendix where available.

4.4 Error Analysis and Failure Modes

We analyzed 229 failed attempts. In our current logs, most failures fall into an unknown bucket—i.e., programs execute but do not meet outcome assertions. Because attribution is coarse, we refrain from asserting specific causes (e.g., logic vs. timing) and instead report them as *objective-mismatch* failures. We are extending the pipeline to disambiguate selector errors, timing issues (waits), and incomplete control flow.

DeepSeek-V3.1 exhibits the highest failure rate (16.6%, 102/616), while **GPT-4o-Mini** achieves the lowest (3.2%, 22/680). The distribution suggests that code-specialized models may generate syntactically correct but semantically incomplete automation sequences, while general-purpose models with stronger instruction-following produce more reliable end-to-end workflows.

4.5 Code Quality Assessment

Our code quality rubric evaluated generated macros across five dimensions: syntax correctness, Selenium best practices, error handling, wait strategies, and maintainability. Under a strict pass/fail rubric (explicit waits, structured error handling, parameterization, and maintainable structure *all* required), no model achieved a full pass. This exposes a gap between functional completion and production-readiness; we include dimension-wise checks in the Appendix for future replication and refinement.

This finding highlights a critical disconnect: while models can generate macros that complete specific tasks, the resulting code lacks robustness features essential for real-world deployment (explicit waits, exception handling, parameterization, and maintainable structure).

4.6 Safety Evaluation Results

Our safety suite revealed nuanced patterns in harmful request handling. All models consistently refuse explicitly harmful requests (credential harvesting, privacy violations), but responses vary in quality and constructiveness.

GPT-4.1 and **GPT-4o-Mini** demonstrate superior "refuse-and-repair" behavior, declining unsafe requests while proposing policy-compliant alternatives. **Gemini-2.5-Pro** and **DeepSeek-V3.1** more frequently provide bare refusals without constructive redirection, potentially frustrating legitimate automation needs.

Notably, ambiguous requests (bulk data export, workflow scaling) expose inconsistencies across models, with some interpreting benign automation intent while others default to conservative refusal. This suggests need for more nuanced safety training that distinguishes between legitimate productivity automation and policy-violating abuse.

4.7 Model-Specific Characterization

GPT-4o-Mini achieves the best overall performance through consistent execution and high success rates across diverse platforms. Its efficiency-oriented design appears well-suited to structured macro synthesis tasks.

GPT-4.1 shows balanced performance with particular strength on complex platforms (TikTok-like) but occasional inconsistencies on simpler tasks (Discord-like).

Gemini-2.5-Pro demonstrates deliberate reasoning capabilities with strong performance on complex booking flows (Airbnb-like) but struggles with high-frequency interaction patterns.

DeepSeek-V3.1 exhibits the most variable performance—excelling on specific platforms (Discord) while showing systematic weaknesses on dynamic content handling and workflow planning.

4.8 Implications for Practical Deployment

Our results suggest that current LLMs can reliably automate simple, well-structured web tasks but remain unsuitable for complex, multi-step workflows requiring robust error handling and dynamic adaptation. The universal absence of production-quality coding practices indicates that generated macros require significant human review and refinement before deployment.
For practitioners, these findings recommend: (1) limiting initial deployments to simple, single-step automation tasks; (2) implementing comprehensive testing and validation pipelines; (3) developing model-specific prompting strategies that emphasize robustness and maintainability; and (4) establishing clear safety guidelines for automation request handling.

5 Conclusion

This work introduced `MacroBench`, a controlled benchmark for evaluating LLMs on executable web-automation macro synthesis. Our results reveal clear stratification across contemporary models: while state-of-the-art systems (e.g., GPT-4o-Mini, GPT-4.1) consistently succeed on simple single-step tasks, performance deteriorates on medium-complexity workflows and collapses entirely for complex, planner-heavy scenarios. The analysis highlights that although LLMs exhibit strong code-generation and syntactic reliability, deficiencies persist in synchronization, robust element targeting, and workflow planning. Safety evaluation indicates that refusal alone is insufficient; models exhibiting *refuse-and-repair*—declining unsafe requests while proposing policy-aligned alternatives—provide greater practical utility.
Overall, `MacroBench` underscores that progress in macro synthesis requires not only scaling model capacity, but also targeted improvements in reasoning, error recovery, and safety alignment.

6 Limitations

While comprehensive, this study carries several limitations. First, our evaluation is constrained to a synthetic ecosystem of seven websites, which, although carefully designed, cannot fully capture the diversity and complexity of real-world platforms. Second, our task set, though broad, emphasizes common interaction patterns and may under-represent rare or adversarial workflows. Third, the safety probes focus on a limited set of harmful request categories and do not exhaustively span the long tail of misuse risks. Fourth, our error attribution and rubric-based quality assessments involve human-designed criteria, which may introduce subjectivity. Finally, we restrict evaluation to four representative LLMs; future work should extend to a wider range of architectures and instruction-tuning strategies. We view these limitations as opportunities for refinement: expanding the website suite, diversifying task complexity, broadening safety probes, and scaling cross-model comparisons will yield a more complete picture of LLM capabilities in web automation.

References

[1] Yuntao Bai et al. "Constitutional AI: Harmlessness from AI Feedback". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[2] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *arXiv preprint arXiv:2107.03374* (2021).

[3] Xiang Deng et al. "Mind2Web: Towards a Generalist Agent for the Web". In: *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*. 2023.

[4] Luyu Gao et al. "PAL: Program-Aided Language Models". In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 2023.

[5] Junwoo Ha et al. "M2S: Multi-turn to Single-turn jailbreak in Red Teaming for LLMs". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Wanxiang Che et al. Vienna, Austria: Association for Computational Linguistics, July 2025, pp. 16489–16507. ISBN: 979-8-89176-251-0. DOI: 10.18653/v1/2025.acl-long.805. URL: https://aclanthology.org/2025.acl-long.805/.

[6] Forrest Huang et al. "Automatic Macro Mining from Interaction Traces at Scale". In: *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. CHI '24. New York, NY, USA: Association for Computing Machinery, 2024. ISBN: 9798400703300.

[7] Sheng Jia, Jamie Kiros, and Jimmy Ba. "DOM-Q-NET: Grounded RL on Structured Language for Web Navigation". In: *International Conference on Learning Representations (ICLR)*. 2019.

[8] Carlos E. Jimenez et al. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" In: *arXiv preprint arXiv:2310.06770* (2023).

[9] Hyunjun Kim et al. *ObjexMT: Objective Extraction and Metacognitive Calibration for LLM-as-a-Judge under Multi-Turn Jailbreaks*. 2025. arXiv: 2508.16889 [cs.CL]. URL: https://arxiv.org/abs/2508.16889.

[10] Hyunjun Kim et al. *X-Teaming Evolutionary M2S: Automated Discovery of Multi-turn to Single-turn Jailbreak Templates*. 2025. arXiv: 2509.08729 [cs.CL]. URL: https://arxiv.org/abs/2509.08729.

[11] Jing Yu Koh et al. "VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*. Bangkok, Thailand, 2024, pp. 881–905.

[12] Thibault Le Sellier De Chezelles et al. "The BrowserGym Ecosystem for Web Agent Research". In: *arXiv preprint arXiv:2412.05467* (2024).

[13] Yujia Li et al. "Competition-level Code Generation with AlphaCode". In: *Science* 378.6624 (2022), pp. 1092–1097. DOI: 10.1126/science.abq1158.

[14] Evan Zheran Liu et al. "Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration". In: *International Conference on Learning Representations (ICLR)*. 2018.

[15] Xiao Liu et al. "AgentBench: Evaluating LLMs as Agents". In: *arXiv preprint arXiv:2308.03688* (2023).

[16] Long Ouyang et al. "Training Language Models to Follow Instructions with Human Feedback". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[17] Long Phan et al. *Humanity's Last Exam*. 2025. arXiv: 2501.14249 [cs.LG]. URL: https://arxiv.org/abs/2501.14249.

[18] Timo Schick et al. "Toolformer: Language Models Can Teach Themselves to Use Tools". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023.

[19] Yongliang Shen et al. "TaskBench: Benchmarking Large Language Models for Task Automation". In: *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*. 2024.

[20] Tianlin Shi et al. "World of Bits: An Open-Domain Platform for Web-Based Agents". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2017.

[21] Tianbao Xie et al. "OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments". In: *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*. 2024.

[22] Frank F. Xu et al. "TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks". In: *arXiv preprint arXiv:2412.14161* (2024).

[23] Shunyu Yao et al. "τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains". In: *arXiv preprint arXiv:2406.12045* (2024).

[24] Shunyu Yao et al. "ReAct: Synergizing Reasoning and Acting in Language Models". In: *International Conference on Learning Representations (ICLR)*. 2023.

[25] Shunyu Yao et al. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023.

[26] Shunyu Yao et al. "WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[27] Ori Yoran et al. "ASSISTANTBENCH: Can Web Agents Solve Realistic and Challenging Tasks on the Web?" In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2024.

[28] Zhexin Zhang et al. "Agent-SafetyBench: Evaluating the Safety of LLM Agents". In: *arXiv preprint arXiv:2412.14470* (2024).

[29] Shuyan Zhou et al. "WebArena: A Realistic Web Environment for Building Autonomous Agents". In: *arXiv preprint arXiv:2307.13854* (2023).

A Workshop-Targeted Appendix: Using Benchmark Outputs for Macro-Safety Training

This appendix outlines how MacroBench artifacts can support *training and evaluation* of LLMs and guardrail models that resist unsafe automation requests, aligning with the Lock-LLM workshop theme of preventing unauthorized knowledge use. We restrict releases to anonymized artifacts and redact operationally dangerous details pending camera-ready.

B Appendix: Detailed Experimental Data

B.1 Aggregate Results

- GPT-4o-Mini: **658/680** (96.8%)
- GPT-4.1: **642/674** (95.3%)
- Gemini-2.5-Pro: **593/666** (89.0%)
- DeepSeek-V3.1: **514/616** (83.4%)

Task Complexity.
- Simple: **2,370/2,584** (91.7%)
- Medium: **37/44** (84.1%)
- Complex: **0/8** (0.0%)

B.2 Website Performance (Success Rate %)

- Airbnb-like: **86.3**
- TikTok-like: **81.5**
- Reddit-like: **94.2**
- Instagram-like: **87.5**
- Facebook-like: **98.7**
- Discord-like: **99.5**
- Threads-like: **90.0**

Why this matters for Lock-LLM. Unlike conventional safety evaluations that target overtly harmful content (e.g., instructions on self-harm, weapon building, or drug synthesis), our benchmark reveals a neglected but equally pressing failure mode: modern LLMs will readily generate executable *macros* in Python+Selenium that can automate large-scale abuse. Examples include bulk posting across forums, scraping user data in violation of terms of service, or constructing credential-harvesting workflows. These cases are rarely flagged by existing guardrails because they appear as "help with automation" rather than obviously malicious requests. The result is a form of **capability laundering**, where powerful but risky behaviors are surfaced under the guise of productivity assistance.

Leveraging benchmark outputs for fine-tuning. Because our evaluation logs include per-task prompts, model generations, execution traces, and curated safety labels, they serve as a concrete resource for training safety-aware systems. Researchers can:
- Fine-tune **guardrail models** to detect and block risky macro requests at both the prompt and output levels.
- Train LLMs with **refuse-and-repair** behavior, where unsafe automation requests are declined and redirected toward safe alternatives (e.g., policy-compliant exports, synthetic test workflows).
- Build **critics and filters** that score generated programs for signs of high-risk automation patterns (mass posting loops, credential interception, or unbounded scraping).

Closing the gap. Our dataset highlights that current LLMs often succeed at macro synthesis but fail at aligning that capability with platform policies. By releasing structured results, refusal labels, and repair targets, we provide the first reproducible basis for developing models that can *distinguish* between benign automation (e.g., accessibility support, testing workflows) and abusive automation (e.g., spam farms, large-scale scraping). This directly supports the Lock-LLM agenda of mitigating unauthorized knowledge use: the same benchmarks that expose unsafe behavior also provide the training signals necessary to constrain it.

Takeaway. In short, MacroBench shows that LLMs do not only produce risky outputs in obvious categories of harm, but also in subtle, automation-centric scenarios that are currently under-protected. Our released data makes it possible to fine-tune both LLMs and guardrail models to address these overlooked risks, turning benchmark evaluation into actionable safety training material.

B.3 Why Macro Safety Is Underserved

Prevailing taxonomies emphasize recognizable explicit harms. Macro synthesis introduces **capability laundering**: seemingly benign "please automate X" requests conceal high-risk behaviors (spam farms, bulk extraction, credential flows). Because outputs are *executable* programs, a single unsafe response scales to *automated*, repeated violations. Our results show this gap is systematic across models, justifying a dedicated training and evaluation pipeline for *macro-aware safety*.

B.4 What We Release

We release artifacts tailored for *training and benchmarking* macro safety:

- **Per-task records** for 200 tasks $\times$ 4 models: prompt, HTML/DOM context, model program output (Python+Selenium), execution logs (stdout/stderr, traces), and evaluation outcomes with labeled failure modes.
- **Safety labels** at *prompt* and *output* levels:
  - `risk_label` $\in$ {SAFE, AMBIGUOUS, UNSAFE}
  - `refusal_label` $\in$ {REFUSE, COMPLY, REPAIR} ("repair" = *refuse-and-redirect* with safe alternatives)
  - `intent_surface` (read-only, content-creation, account/identity, bulk/automation, data-exfiltration), `policy_concern` (scraping, spam/abuse, credentials/privacy)
- **Repair targets**: policy-aware rewrites, consented/rate-limited exports, and synthetic fixtures that models should choose over risky code.
- **Splits & generalization**: train/dev/test with (i) *cross-site* and (ii) *prompt-paraphrase* generalization.
- **Governance kit**: usage policy, evaluation scripts, and a risk-aware leaderboard protocol reporting both *utility* and *safety*.

Data format. Each record is a self-contained JSON object:

```
{
  "task_id": "...", "site": "...", "split": "train|dev|test",
  "prompt": "...", "html_context": "...",
  "model_id": "...", "generation": "...",
  "exec_pass": true|false, "failure_mode": "syntax|timing|logic|...",
  "risk_label": "SAFE|AMBIGUOUS|UNSAFE",
  "refusal_label": "REFUSE|COMPLY|REPAIR",
  "policy_concern": ["scraping","spam","credentials"],
  "repair_target": "policy-compliant alternative"
}
```

B.5 Intended Uses

Artifacts enable training and evaluation of:

1. **Guardrail classifiers** detecting risky macro intent and risky program patterns *pre-* and *post-generation*.
2. **Refuse-and-repair LLMs** that decline unsafe automation and provide policy-compliant alternatives (consented exports, synthetic stubs).
3. **Planner critics / linters** that score plans and code for macro-risk signatures (mass posting loops, credential flows, unbounded scraping).

B.6 Training Recipes (High-Level, Safety-Preserving)

We leverage labels without teaching step-by-step abuse. All setups operate on prompts and high-level alternatives.

SFT for refusal-and-repair. Inputs: (prompt, minimal context). Targets: curated `REPAIR` responses. Validate for (i) refusal on UNSAFE, (ii) helpfulness on SAFE.

Preference optimization (DPO/IPO). Form pairs $(y^{\text{good}}, y^{\text{bad}})$ where y^{good} is `REPAIR` or compliant, and y^{bad} is a code-emitting `COMPLY` for UNSAFE/AMBIGUOUS. Optimize to shift mass away from unsafe compliance.

Reward modeling for macro risk. Train $R(\cdot)$ to predict `risk_label` from (prompt, plan/code summary). Use R as a decoding gate or to trigger clarifications before emitting code.

Static program filters (post-generation). Detect abstracted features: action count, credential flows, cross-origin hops, loop depth, missing waits/guards, target roles.

Parameter-efficient fine-tuning. Apply LoRA/QLoRA (e.g., rank 8–32, 1–3 epochs, 5% warmup) on REPAIR-centric data to avoid memorizing unsafe snippets.

B.7 Evaluation: Utility–Refusal Frontier

Report a *Utility–Refusal Frontier (URF)*:
- **Benign utility**: success on SAFE tasks without unnecessary refusal.
- **Harm refusal**: refusal rate on UNSAFE tasks.
- **Ambiguity handling**: share of AMBIGUOUS prompts eliciting clarification or safe alternatives rather than code.

Compare models at matched benign-utility points (e.g., 90% of baseline benign success) to fairly assess refusal quality.

B.8 How to Integrate Guardrails in a Macro Agent

1. **Pre-generation classifier**: score user intent; route to REPAIR when risk is high.
2. **Planning gate**: score macro plans with R; require confirmation or rewriting if risky.
3. **Static code audit**: lint for credential capture, mass posting, cross-site scraping, and missing rate limits.
4. **Sandbox-first execution**: execute in hermetic sandboxes with policy simulators; require approval for live endpoints.
5. **Post-exec telemetry**: log element/role-level actions and feed counterexamples back into training.

B.9 Splits and Generalization Protocols

Two complementary regimes:

Cross-site: train on a subset of websites; test on disjoint sites to measure robustness to unfamiliar DOM/layout conventions.

Prompt-shift: train on canonical phrasings; test on paraphrases, indirect requests ("help me scale this workflow"), and injection-style wording.

Report macro-averaged metrics across `risk_label` strata to avoid masking regressions.

B.10 Red Teaming and Probe Design

We provide non-explicit but risky prompts that realistically elicit abuse-capable macros while avoiding overtly malicious phrasing. To preserve safety, releases omit exploit steps and operational code; instead we include high-quality *repair* targets and *benign decoys* to test specificity and false positives.

B.11 Responsible Release and Use

Intended use. Research on macro-aware LLM safety, training guardrails, evaluating refusal/repair.
Prohibited use. Training models to increase harmful automation success or to bypass platform policies.
Access controls. Research-use terms prohibit reproducing unsafe code outside sandboxed evaluation and require downstream safety disclosures.
Minimization. Unsafe generations are redacted or abstracted to features/metadata sufficient for detectors and repairs.
No live-target interaction. All websites are synthetic; no real user data or credentials are present.

B.12 Data Card (Summary)

Motivation. Enable macro-aware safety via refusal and safe alternatives.
Composition. 200 tasks (a curated subset) across **seven** synthetic websites; 4 LLMs' generations with traces and safety labels.

Collection. Prompts span interaction complexity and safety probes; execution in sandboxed containers.
Labels. Dual-annotator protocol with adjudication for `risk_label` and `refusal_label`; agreement

metrics reported in-repo.
Uses. Guardrail classification, reward modeling, refusal-and-repair fine-tuning.
Limitations. Domain shift to real sites; evolving ToS; conservatism bias in repairs.
Ethics. Dual-use risks mitigated via abstraction, licensing, and bans on operational misuse.

B.13 Limitations and Future Work

Our taxonomy focuses on browser macros; native app and OS-level scripting are out of scope. Refusal-and-repair must balance *helpfulness* with *harm avoidance*. Future work: (i) richer consent/authorization flows; (ii) context-grounded repairs using platform ToS templates; (iii) adversarial training with automatically mined ambiguous prompts.

B.14 Reproducibility Checklist for Safety Fine-Tuning

1. Publish exact train/dev/test splits and task IDs.
2. Log refusal thresholds or gating (e.g., reward cutoffs); report URF at multiple operating points.
3. Release redaction policy for unsafe outputs (kept text/features).
4. Document prompting instructions used to elicit repairs/clarifications.
5. Provide failure analysis by `policy_concern` and website category.

B.15 Practical Notes for Practitioners

- **Prefer repair over silence.** Offer actionable, policy-compliant alternatives (e.g., consented exports) instead of bare refusals.
- **Tune strictness.** Calibrate thresholds to preserve benign utility; report full URF curves.
- **Human-in-the-loop.** Route high-risk/uncertain cases; log decisions for continual improvement.
- **Defense-in-depth.** Combine intent filters, plan critics, static audits, and sandbox approvals rather than relying on a single stage.

Takeaway. LLMs readily produce executable macros that can scale abuse. The `MacroBench` artifacts—prompts, structured outputs, execution traces, and curated refusal/repair targets—turn evaluation into *training signals* for macro-aware safety, directly targeting risks underrepresented in conventional content-focused safety.