Toward Autonomous UI Exploration: The UIExplorer Benchmark

Andrei Cristian Nica [*] [1] **Akshaya Vishnu Kudlu Shanbhogue** [*] [1] **Harshil Shah** [1] **Aleix Cambray** [1]
Tudor Berariu [1] **Lucas Maystre** [1] **David Barber** [1] [2]

Abstract

Autonomous agents must know how to explore user interfaces (UIs) for reliable task solving, yet systematic evaluation of this crucial phase is lacking. We introduce UIEXPLORE-BENCH, the first benchmark explicitly dedicated to UI exploration. The benchmark evaluates agents with either Structured mode (granting access to layout information like DOM trees) or Screen mode (relying on GUI-only observations such as screenshots and human-like mouse/keyboard interactions) across three levels in a standardized GitLab sandbox environment. We formalize exploration as the process of maximizing the set of actionable UI components discovered and propose a metric, human-normalized UI-FUNCTIONALITIES OBSERVED (HUFO), to quantify the effectiveness of exploration. Our results show that UIEXPLORE-ALGO achieves the leading mean HUFO scores, reaching up to 77.2% of human performance in Structured mode and 59.0% in Screen mode at 2,000 steps, particularly excelling at the Sparse level. The results highlight the relevance of our benchmark, as current agents show a substantial performance gap compared to 1 hour of human expert exploration, indicating ample room for future advancements. We publicly release the benchmark environment, an exploration dataset, and an evaluation suite to catalyze research into efficient UI exploration strategies and their downstream applications, such as experience-driven task completion and automated training data generation.

1. Introduction

The pursuit of truly autonomous agents capable of interacting with the digital world hinges on their ability to understand and navigate user interfaces. Imagine a robotic process automation bot landing on a freshly upgraded payroll dashboard, with new features and user interface (UI) improvements. In such a scenario, and countless others, the agent cannot rely solely on pre-programmed instructions or static UI knowledge. Frequent UI changes and the sheer diversity of applications require modern agents to actively explore and discover available functionalities. This exploration is not merely a preliminary step; it is a fundamental learning process (Schmidhuber, 2008; 1991; Thrun, 1992).

We posit that efficient exploration is paramount to achieving reliable task execution and genuine autonomy in dynamic digital environments. At a high level, the performance of an UI agent is determined by two factors. First, its awareness of the UI space, such as its understanding of what functionalities exist and where interactions lead. Second, the agent's ability to solve tasks successfully, *conditioned on its knowledge of the UI space*. This distinction is important: for example, an agent might have access to the knowledge required to complete a task but might fail to precisely locate the right UI element. Measuring what situations an agent has been exposed to or not, independently of its ability to solve tasks, is valuable, as it helps to disentangle these two performance factors. Robust UI exploration can provide a mechanism to explicitly address and improve both factors. It can address the first by maximizing knowledge of the UI space instead of success in specific tasks. It can address the second by encouraging agents to minimize the discrepancy between the predicted consquences of an action and the actual observed outcome.

At a more granular level, the design of effective exploration policies provides the following benefits. (i) Enhancing unattended data generation by efficiently discovering the functional space of the UI. This practice is already used to gather training experiences (Su et al., 2025; Fan et al., 2025; Qi et al., 2024; Bai et al., 2024), and more efficient methods are highly valuable. (ii) Uncovering the "unknowns" of the agent's underlying predictive model and its misalignments (Fan et al., 2025). This could be especially beneficial for visual computer-use agents, which often fall short in grounding UI functionality (Agashe et al., 2025) and often misinterpret the purpose or effects of the interface elements (Zhang et al., 2025b). A dedicated benchmark can encourage research into rewarding agents for such information-seeking

---

[*]Equal contribution [1]UiPath [2]UCL, London, UK. Correspondence to: Andrei Cristian Nica <andrei.nica@uipath.com>.

Workshop on Computer-use agents @ ICML 2025, Vancouver, Canada. Copyright 2025 by the author(s).

behavior. (iii) Enabling, through structured exploration, the creation of persistent databases of application-specific memory. This could offer a scalable solution to improve generalist agents with deep contextual understanding and opens an interesting research direction. Driven by these three benefits, our work addresses the need for principled methods to measure efficient UI coverage and guide the design of effective policies.

Despite the recognized importance of exploration, there is a significant gap in systematic evaluation. Current benchmarks for UI agents predominantly entangle exploration with task-specific rewards (Zhou et al., 2023b; Chezelles et al., 2024; Drouin et al., 2024; Boisvert et al., 2025). Consequently, empirical comparisons make it difficult to disentangle the contributions of an agent's exploratory prowess from its ability to exploit learned knowledge for a specific goal. Most existing research either conflates these two aspects or focuses on exploration merely as a mechanism to gather experiences to improve exploitation, often acknowledging that more sophisticated exploration strategies would be beneficial (Su et al., 2025).

To bridge this gap, we introduce UIEXPLORE-BENCH, the first benchmark designed for systematic assessment of UI functionality exploration. UIEXPLORE-BENCH offers a standardized GitLab sandbox environment (adapted from WebArena (Zhou et al., 2023b)) with three difficulty levels, for agents operating in Structured or Screen mode. We formalize exploration as the process of maximizing the discovery of unique functionalities offered by a system, captured by our proposed metric: UI-FUNCTIONALITIES OBSERVED. Designed to surface key exploration challenges, UIEXPLORE-BENCH — together with its publicly released exploration dataset - aims to promote and standardize research on agents that explore more efficiently. In addition, the fixed dataset enables fairer comparison of exploitation strategies and clearer attribution of performance gains to either improved exploration or better use of prior knowledge.

Our contributions are fourfold:

- We publicly release **UIEXPLORE-BENCH**, a benchmark environment with three difficulty levels based on a GitLab sandbox, supporting both Structured and Screen modes.
- We introduce two novel UI-centric metrics that capture functionality discovery (UI-FUNCTIONALITIES OBSERVED) and exploration efficiency (UI-FUNCTIONALITIES TESTED), as well as a lightweight evaluation suite.
- We develop UIEXPLORE-ALGO, a Go-Explore-inspired (Ecoffet et al., 2019) novelty-driven agent designed for structured UI exploration and memory construction, and compare its performance to baseline agents: random, BFS/DFS and GUI-Bee‡ .
- We publicly release an open exploration dataset gathered with UIEXPLORE-BENCH, enabling downstream research on experience-driven task completion, metric learning, and more.

2. Related Work

Exploration in Reinforcement Learning. Effective exploration in reinforcement learning (RL), especially with sparse rewards, remains a significant research challenge. Common paradigms include count-based methods (Bellemare et al., 2016), curiosity-driven intrinsic rewards utilizing prediction errors (Pathak et al., 2017; Burda et al., 2018), and information-theoretic strategies such as VIME (Houthooft et al., 2016). Episodic memory-based approaches (Badia et al., 2020) and hierarchical structures (Steccanella et al., 2020; Nica et al., 2022) have further enabled structured exploration in complex environments. Our proposed algorithm, UIEXPLORE-ALGO, builds upon this literature by integrating a structured and hierarchical search strategy inspired by Go-Explore (Ecoffet et al., 2019; 2021; Lu et al., 2024) that uses novelty and preference-based prioritization to efficiently explore user interfaces.

Computer-Use Agent Benchmarks. A growing set of benchmarks has been proposed to evaluate agents operating digital interfaces, including web-based environments like WebArena (Zhou et al., 2023b), BrowserGym (Chezelles et al., 2024), MiniWoB (Shi et al., 2017) and OpenWebVoyager (He et al., 2024), as well as productivity-focused setups such as WorkArena (Drouin et al., 2024) and WorkArena++ (Boisvert et al., 2025). These benchmarks primarily target the completion of goal-conditioned tasks and often conflate exploration with exploitation, making it difficult to isolate and study exploration behaviors in depth. Vision-centric environments such as OSWorld (Xie et al., 2024) further emphasize the challenges of agents interacting through human-like modalities, e.g., mouse, keyboard, and screen input, without access to privileged structural information. However, they still lack dedicated tools and metrics to systematically evaluate exploration as a standalone capability. In contrast, UIEXPLORE-BENCH introduces a standardized benchmark explicitly designed to assess UI functionality exploration, providing both Structured (DOM-based) and Screen (GUI-based) modes, and metrics focused on functional discovery and exploration efficiency. This allows for a clearer analysis of exploratory competence - an essential precursor to robust and general-purpose computer-use agents.

Computer-Use Agents. Recent state-of-the-art computer-use agents leverage diverse modalities and architectures. General purpose frameworks such as OpenAI's Computer-Using Agent (CUA (OpenAI, 2025), Simular Agent S2 (Agashe et al., 2025), and Claude 3.5 Sonnet Compute-use (Anthropic, 2024) demonstrate strong visual grounding and reasoning capabilities across web, desktop, and mobile tasks. Enterprise-focused agent frameworks like Microsoft AutoGen (Wu et al., 2023) and IBM CUGA (Marreed et al., 2025) leverage machine learning agents to automate the resolution of tasks and workflows.

LLM-Powered Agents and UI Exploration. Several recent works explicitly or implicitly leverage exploration within their approaches. MobileGPT (Lee et al., 2024) and AppAgent (Li et al., 2024) explore app interfaces to build structured knowledge bases and reusable task modules. Symbiotic Cooperation for Web Agents (Zhang et al., 2025a) integrates exploration with small-model distillation loops, and Explorer (Pahuja et al., 2025) employs large-scale LLM-driven trajectory synthesis. WebWalker (Wu et al., 2025) introduces a multi-agent system where an explorer navigates multi-hop web interfaces while a critic evaluates information sufficiency, aiming to improve deep web exploration. LATS (Zhou et al., 2023a), Tree Search for Language Model Agents (Koh et al., 2024), and ExACT (Yu et al., 2024) utilize advanced tree-search methods to enhance targeted traversal and exploration efficiency. Learn-by-interact (Su et al., 2025) further generates exploration data through self-instructed tasks. Another example is GUI-Bee (Fan et al., 2025), which explicitly uses exploration to uncover actionable UI elements and learn from interaction data; it combines a Q-learning framework with a structured graph over the action space to guide exploration in a sample-efficient manner. Unlike these methods that mainly use exploration instrumentally, UIEXPLORE-BENCH prioritizes systematic measurement and improvement of exploration itself, establishing its distinct value for progress in computer use agent research.

By explicitly isolating exploration and introducing rigorous metrics, UIEXPLORE-BENCH not only advances exploration evaluation but provides foundational insights useful for improving downstream tasks such as retrieval-augmented generation (RAG)-conditioned planning and model introspection (Yu et al., 2024; Koh et al., 2024; Zhou et al., 2023a).

3. UIEXPLORE-BENCH Benchmark Suite

UIEXPLORE-BENCH is designed to systematically evaluate the UI exploration capabilities of autonomous agents within a standardized and complex sandbox environment derived from the `WebArena` GitLab setup. GitLab was chosen due to its rich interactivity, multi-layered navigation structure, and clear metric computation facilitated by privileged DOM data. It also offers extensive opportunities for agents to create and manipulate content (such as projects, issues, and files), making it ideal for evaluating goal-directed exploration and interactive discovery. This environment serves as a controlled yet challenging testbed to measure agents' abilities to explore and discover interactive functionalities. Although initially designed as a web-based benchmark, we propose extending it to the vision-only, general-purpose computer interaction setting, facilitating evaluation beyond web applications and towards broader GUI-based interaction scenarios.

3.1. Environment Design

We structure the benchmark into **three distinct levels**, corresponding to three different initial states. Each level introduces unique challenges and constraints to the exploration problem.

Abundant Level. The agent starts in a logged-in state with extensive repositories and visible interactive elements (as per the original WebArena setup). This environment tests agents' abilities to effectively navigate and distinguish actionable components amidst abundant but potentially distracting UI elements.

Moderate Level. The agent remains logged in, but faces a reduced, repository-free UI skeleton, minimizing duplicative interactions. Exploration at this level emphasizes efficient coverage of essential UI functionality without the complexity of content-heavy pages.

Sparse Level. Simulates a fresh GitLab instance, without users and repositories, requiring the agent to initiate exploration from a logged-out state. Although minimal, there are still a few functionalities to explore even without registering a user. This scenario assesses the agent's capacity to bootstrap exploration from minimal initial state information. It also assesses the agent's ability for open-world exploration.

To better align with established efforts in the computer-use community, we formalize **two distinct interaction modes**.

Structured Mode: The agent interacts with environments through the Document Object Model (DOM), perceiving the hierarchical structure of elements, and performing high-level actions (clicking, form filling, hovering) identified via DOM attributes. This mode leverages the semantic web structure, exemplified by benchmarks such as WebArena (Zhou et al., 2023b).

Screen Mode: The agent relies solely on visual inputs, perceiving the UI as rendered images without having access to the underlying DOM structures. Actions are low-level, including mouse movements, pixel-coordinate clicks, and keyboard presses, emphasizing visual understanding and mimicking human interactions. This mode aligns with benchmarks such as OSWorld (Xie et al., 2024).

Additional details on observation and action spaces for these interaction modes are provided in section 3.4. Each level includes guardrails to ensure a self-contained exploration experience, preventing agents from accessing external sites or performing disruptive actions like accidental logouts. External links or pop-ups are sandboxed via a boundary HTML page (detailed in Appendix D).

3.2. Exploration Protocol

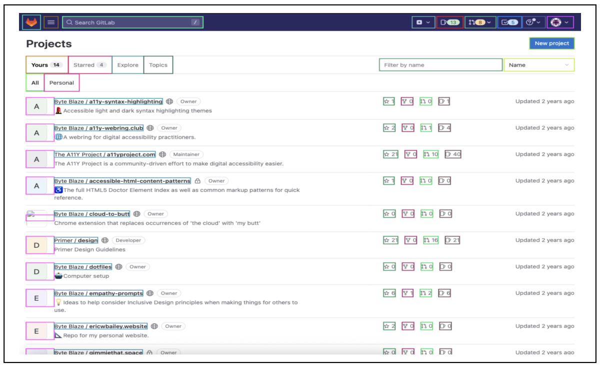
Exploration episodes begin with an exploration agent starting at the GitLab landing page, with session storage cleared to ensure a consistent starting state. The agent is evaluated over a single trajectory that spans 2,000 environment actions (atomic actions), reflecting a realistic and extended interaction scenario where environment changes persist throughout the exploration. This single-trajectory protocol ensures that agents are evaluated on sustained exploratory behaviors rather than repeated short-term interactions.

Structured and Screen modes support navigation actions such as `goto`, `go_back`, and `go_forward`. Explicit support for these actions aligns the benchmark with real-world applications that typically provide deep linking, history navigation, or saved states, thus enhancing practicality. Future extensions of Screen mode may include environments restricting such navigational aids, offering alternative exploration challenges. Details of the exploration protocol, including reset semantics, are provided in Appendix D.

3.3. Metrics

An exploration agent's primary goal is to identify the functionalities exposed by an application in order to exploit it later. Our primary metric is the number of UI-functionalities observed at time T, denoted as UFO@$T \doteq |\cup_{t=1}^{T} F_t|$, where F_t is the set of distinct functionalities revealed at step t. These elements are grouped not only by semantic intent but also by their implementation in the UI structure; for instance, visually distinct "Home" buttons may count as separate functionalities, while all repository links map to a shared class such as $view\_repo$. Examples of functionalities can be found in Fig. 1. The complete formalism and definition of functionality is in Appendix C. In Structured mode, all functionalities accessible in the DOM are considered in the computation of UFO. However, in Screen mode, only the functionalities accessible from the screen (viewport) are considered.

We also normalize UFO with respect to the average number of UFO discovered by human exploration agents in 1 hour. We denote this human-normalized version as HUFO. In order to normalize the score, we collected exploration trajectories from human participants (Appendix B) in the Abundant level. The normalization constants for the Structured mode and the Screen mode can be found in Table 2. Our secondary metric, UI-functionalities tested (UFT@T), counts the distinct functionalities that an exploration agent acted upon, as a percent of the total number of steps. UFT acts as a measure of exploration efficiency and is in the range $[0, 1]$.



A	Byte Blaze / a11y-syntax-highlighting	Owner	Accessible light and dark syntax highlighting themes	1	0	0	1	Updated 2 years ago
A	Byte Blaze / a11y-webring.club	Owner	A webring for digital accessibility practitioners.	2	0	1	4	Updated 2 years ago
A	The A11Y Project / a11yproject.com	Maintainer	The A11Y Project is a community-driven effort to make digital accessibility easier.	21	0	10	40	Updated 2 years ago
A	Byte Blaze / accessible-html-content-patterns	Owner	The full HTML5 Doctor Element Index as well as common markup patterns for quick reference.	1	0	0	0	Updated 2 years ago
	Byte Blaze / cloud-to-butt	Owner	Chrome extension that replaces occurrences of 'the cloud' with 'my butt'	0	0	0	0	Updated 2 years ago
D	Primer / design	Developer	Primer Design Guidelines	21	0	16	21	Updated 2 years ago
D	Byte Blaze / dotfiles	Owner	Computer setup	0	0	0	0	Updated 2 years ago
E	Byte Blaze / empathy-prompts	Owner	Ideas to help consider Inclusive Design principles when making things for others to use.	6	1	2	6	Updated 2 years ago
E	Byte Blaze / ericwbailey.website	Owner	Repo for my personal website.	2	0	0	0	Updated 2 years ago
	Byte Blaze / gimmiethat.space	Owner		0	0	0	0	Updated 2 years ago

Figure 1. Functionality groupings in homepage. Boxes of the same color are grouped under the same functionality

We chose not to adopt per-observation count metrics, such as the Depth-fixed DOM Diversity Counts (D3C) introduced in GUI-Bee (Fan et al., 2025), as they treat all observations equally. Our decision was based on the following considerations. (i) Different observations could differ significantly in the number of functionalities they contain. Treating all observations as equally informative fails to capture this variability and undervalues richer states that offer access to multiple functionalities. (ii) D3C includes structured elements regardless of its visibility on the screen. Due to this, D3C is only applicable for Structured mode and is not applicable in Screen mode. (iii) Minor changes in the DOM structure can significantly affect the score (e.g. user's "theme", see Appendix E).

3.4. Observation & Action Spaces

We distinguish between two agent modalities with distinct observation and action spaces: Structured and Screen mode.

Structured Mode. Agents have access to a structured representation of the interface, such as the DOM or AXTree. Each element is identified via a unique identifier (e.g., `bid`), along with semantic metadata like tag, role, label, and visibility. This enables direct access to high-level functionality. The action space consists of 11 semantic actions including `click(bid)`, `fill(bid, value)`, `goto(url)`, and `keyboard_press(key)`. These are discrete, parameterized actions grounded in the DOM structure.

Screen Mode. Agents receive only raw pixel observations from the UI, rendered as 1280×720 RGB images. Structural information is not available. The action space includes 18 low-level operations, such as `mouse_move(x, y)`, `mouse_click(x, y)`, `keyboard_type(text)`, and `scroll(dx, dy)`. These actions operate in pixel coordinates and approximate human-like GUI interactions.

In both modes, the environment also provides a natural language instruction that defines the available actions and their parameters, facilitating integration with language models.

4. Agents

To provide comprehensive reference points for various methodological approaches, we evaluated a range of agents. These include random baselines, heuristic search methods, established techniques from existing literature, and approaches that take advantage of oracle information. The latter, particularly for Screen mode, helps establish an understanding of potential performance ceilings when, for instance, perfect bounding-box information for interactive elements is assumed. This variety allows for a clearer delineation of challenges and capabilities specific to UI exploration.

A key consideration in our evaluation is the distinction between agents operating in the Structured mode versus Screen mode, each with its own observation and action space constraints (see Section 3.4).

4.1. Baselines

We include several baseline agents to contextualize the performance of our proposed method and highlight specific exploration challenges. In the case of Screen mode, many agents such as GUI-Bee rely on the UI context grounding phase as part of agent execution. This phase is responsible for the extraction of potential actions from screenshots. In order to isolate it's effects on exploration and help ensure a fair comparison of their search strategies rather than their visual grounding capabilities, we provide certain agents access to perfect information (such as bounding boxes) of interactive elements on the webpage. Such agents are said to possess "oracle" information.

Random Agent: This agent serves as a fundamental baseline. In Structured mode, the agent performs one of the following actions uniformly at random: `scroll_up`, `scroll_down`, `go_back`, and `click` (for each of the DOM elements). In Screen mode, the agent chooses one of the following actions: `scroll_up`, `scroll_down`, `go_back`, and `click` on a random pixel with weights 0.09, 0.09, 0.02, and 0.80, respectively.

Heuristic Random Agent: This agent assumes access to the perfect bounding-box information of interactive DOM elements on the webpage. The agent selects an action uniformly at random from the list of `click` actions for each of the interactive DOM elements, along with `scroll_up`, `scroll_up` and `go_back`. In the Screen mode, the agent limits the click actions to only the elements visible on the screen.

URL-Space Search (BFS & DFS): These baselines operate in the Structured mode by extracting hyperlinks from the DOM tree. We apply both Breadth-First Search (BFS) and Depth-First Search (DFS) strategies to systematically explore the discovered URL space.

GUI-Bee‡: As a baseline from the literature, our implementation adapts the GUI-Bee algorithm (Fan et al., 2025). It extracts click, scroll, and go-back actions based on Q-value estimates derived from a Q-table. To maintain simplicity and focus on the core exploration dynamics, our version omits the model-based Q-value reestimation step. This agent is primarily designed for and evaluated in the Screen mode.

Human Evaluation: To provide a critical reference point, we include performance data from human evaluators. Three participants with varying levels of familiarity with the GitLab instance were involved: a proficient user very familiar with the specific GitLab instance, a developer familiar with the underlying technology, and an amateur non-technical user. Participants were asked to identify all the functions that Gitlab provides, and their interactions were captured for 1 hour. More details can be found in Appendix B.

Algorithm 1 UIEXPLORE-ALGO: Exploration Loop

```
Require: start state $s_0$, $N_M$ number of macro-actions, $N_A$ number of atomic-actions
$G \leftarrow \{\}$ {knowledge graph with visited states}
$s \leftarrow s_0$
while True do
    for $i = 1$ to $N_M$ do
        $(s_d, A) \leftarrow$ describe_state_and_macro_actions($s$)
        $a_M \leftarrow$ choose_novel_macro_action($s$, $A$, G)
        while not finished($a_M$) and step $< N_A$ do
            $a_A \leftarrow$ act_agent.execute($a_M$)
        end while
        $G \leftarrow G \cup \{s, s_d, A, a_{A_i}..\}$
    end for
    $s \leftarrow$ goto(choose_frontier_macro_action($G$))
end while
```

4.2. UIExplore-AlGo

Our algorithm is a Go-Explore–style framework that treats UI exploration as goal-directed map-building. At every step we prompt GPT-4o to produce a compact natural-language description of the state and possible next actions. These textual outlines, rather than raw screenshots, become the keys in our novelty table; this makes the scorer resilient to visual drift (e.g., GitLab's theme switch) while still rewarding unseen functionality.

Exploration proceeds hierarchically. GPT-4o first suggests macro-actions (e.g., "create project") that can unlock whole new regions of the state space but would be prohibitively slow to discover with atomic actions alone. Within a macro-action, a smaller model (Claude) executes up to N_A atomic steps, giving us the best of both worlds: deliberate high-level search as well as fast, low-level control.

A candidate macro-action is ranked by a tri-objective score, (i) predicted novelty of its endpoint, (ii) dissimilarity from past trajectories (to avoid rabbit holes), and (iii) an importance prior from GPT-4o that favours actions likely to unlock potentially new areas of functionality. This mixture biases the agent toward cumulative, long-horizon payoff instead of one-step curiosity.

Although we can issue a `goto` to return to a previously visited state, the restored environment may not perfectly reproduce the original (e.g., form inputs may be lost). To mitigate this, the agent executes several macro-actions before "teleporting" again, amortizing the cost of resets and enabling batch exploration of multiple potential frontiers.

Algorithm 1 summarises the loop; full prompts and an annotated trajectory appear in Appendix F.

5. Experimental Setup

Our experimental setup evaluates six agents across three GitLab environment levels (Abundant, Moderate, and Sparse) and two interaction modalities: Structured (with DOM access) and vision-based Screen mode. We use hUFO as the primary metric, normalized against human reference scores specific to each mode and environment level. Each agent performs a single exploration trajectory of up to 2,000 atomic actions. We also report a multi-seed analysis for the Screen-Moderate setup detailed in Section 6.2.

We leveraged the BrowserGym framework with Playwright for environment control and Docker for containerization. Performance is normalized by atomic action count to ensure agent-centric fairness, isolating policy efficiency from hardware or network latencies.

For our proposed agent, UIExplore-AlGo, the state and macro-action descriptions are generated by *gpt-4o-2024-11-20*, while the action execution relies on *claude-3-5-sonnet-v2@20241022*, chosen for its Screen coordinate prediction capabilities. UIExplore-AlGo uses a maximum of 6 and 12 atomic actions per macro-action in the Structured and Screen modes, respectively, reflecting the differing action complexities.

6. Results

This section presents key findings from our benchmark, including agent performance, ablation studies, and qualitative insights into exploration patterns and challenges.

6.1. Exploration Performance

Our primary results, summarized in Table 1 and visualized in Figure 2, report human-normalized UI Functional Discovery (hUFO). The scores are normalized using the average human exploration performance (Table 2), highlighting the significant variability between novice and expert human runs, establishing an expert-level performance benchmark of 114% (Structured) and 117% (Screen) for hUFO. The absolute values for UFO are reported in Table 4., We also report UFT at $2,000$ steps in Table 3 for the agents. The current implementation UFT is based on the Structured mode action space, and would require further development to support the Screen mode.

Table 1. Human-normalized scores (%) for UI Functional Discovery (hUFO) at 2,000 steps for each level (Abundant(A), Moderate(M), Sparse(S)), and mean normalized scores across all levels at 500, 1,000, and 2,000 steps. Results for a single seed.

				Levels Average		
Agent	**A@2k**	**M@2k**	**S@2k**	**@500**	**@1k**	**@2k**
Structured Mode						
DFS	15.7	21.3	2.4	7.5	10.1	13.2
random	44.1	17.4	3.7	14.1	18.1	21.7
h-random*	70.9	25.3	5.0	22.0	27.3	33.7
BFS	**89.5**	30.5	5.8	27.2	38.6	41.9
UIEx-AlGo	80.4	**74.9**	**76.4**	**49.5**	**60.8**	**77.2**
Screen Mode						
random	37.6	9.0	3.1	10.4	13.5	16.6
h-random*	67.6	23.3	5.2	21.5	26.8	32.1
GUI-Bee‡*	**69.9**	25.8	5.3	**22.9**	28.1	33.7
UIEx-AlGo	59.3	**60.1**	**57.6**	21.0	**37.3**	**59.0**

* agent has access to oracle information

Table 2. UI Functional Discovery counts after 1 hour of exploration in *Abundant* level for human participants. We report results for three levels of expertise: novice, intermediate, and expert, alongside the average used as base to normalize hUFO scores.

Mode	**Novice**	**Intermediate**	**Expert**	**Base (Avg)**
Structured	2196	2228	2722	2382
Screen	1458	1481	1880	1606

UIExplore-AlGo achieves the highest mean normalized scores across difficulty levels at 1,000 and 2,000 steps in both Structured and Screen modes, surpassing all baseline agents, even those leveraging oracle information. However, no single agent consistently outperforms the rest in all scenarios. UIExplore-AlGo, however does outperform the rest on UFT, highlighting the fact that the algorithm interacts with different types of elements without repetition in the Structured mode.

In Abundant level, simple heuristic agents such as heuristic-random and BFS, outperform UIExplore-AlGo, particularly in the early stages of exploration (first 500 steps). This result suggests that abundant interactive elements al-

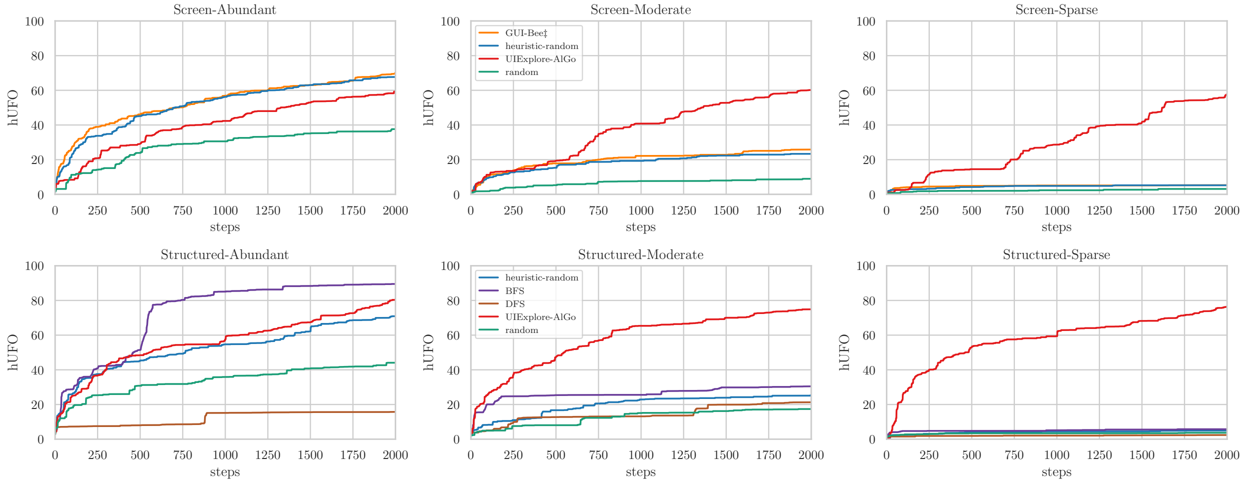


Figure 2. Progress of human-normalized UI Functional Discovery (UI-Functionalities Observed) over exploration steps (atomic-actions). Top row: Screen mode; bottom row: Structured mode. Columns: Abundant, Moderate, and Sparse difficulty levels.

Table 3. Scores for UI-Functionalities Tested (UFT) at 2,000 steps for each level (Abundant(A), Moderate(M), Sparse(S)), and mean scores across all levels at 500, 1,000, and 2,000 steps. Results are reported for a single seed for agents in Structured mode and Screen mode for agents having access to perfect bounding box information.

				Levels Average		
Agent	**A@2k**	**M@2k**	**S@2k**	**@500**	**@1k**	**@2k**
Structured Mode						
DFS	1.4	4.5	0.2	1.3	1.7	2.0
random	8.8	5.6	2.7	2.4	3.6	5.7
h-random*	17.2	9.2	3.0	4.5	6.9	9.8
BFS	16.8	7.7	1.7	5.4	8.0	8.7
UIEx-AlGo	**34.2**	**26.4**	**29.6**	**13.8**	**19.4**	**30.1**

* agent has access to oracle information

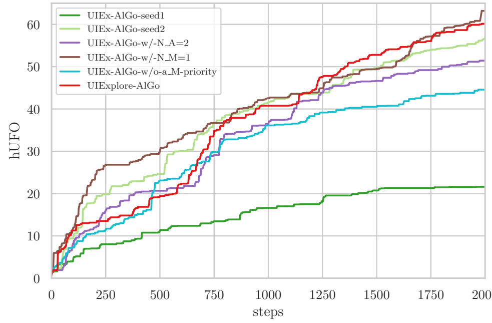


Figure 3. Ablation results on the Screen–Moderate setting: human-normalized UI Functional Discovery (UI-Functionalities Observed) over 2,000 atomic actions. We compare the default UIEx-AlGo (solid green) against three ablations (i) single-macro horizon (N_M=1), (ii) tight atomic-action budget (N_A=2), and (iii) no priority/novelty, as well as two additional random seeds.

low heuristic strategies to rapidly discover functionalities. Specifically, BFS excels initially due to quick access to diverse functionalities directly reachable within a few levels of URL depth, plateauing only when delving into deeper content exploration (e.g., repository internals). Conversely, DFS performs poorly, becoming quickly trapped in repetitive depth interactions due to an early redirect to the vast help section of Gitlab.

UIExplore-AlGo demonstrates significant advantages in the Sparse level, where deliberate, multistep actions (e.g., creating users or projects) are essential to discover deeper functionalities. This indicates its ability to prioritize meaningful interactions effectively. UIExplore-AlGo achieves notably lower performance in the Screen mode compared to the Structured mode, underscoring the greater challenges posed by purely visual interaction and the need for improved visual grounding in agents.

Interestingly, the random selection baseline achieves relatively high scores in Abundant level (up to 44% hUFO), emphasizing that even naive strategies can produce substantial exploration in highly interactive environments. GUI-Bee‡ remains slightly more effective than the heuristic-random approach, likely due to the complexity and the large action space of GitLab's interface.

Overall, while UIExplore-AlGo consistently delivers strong exploration performance across modes and levels, the results reveal significant room for improvement (up to 22% behind expert human exploration in Structured and 40% in Screen modes). These results set a clear benchmark and motivate future advancements in UI exploration algorithms.

6.2. Ablation & analysis

To assess the robustness and design choices of UIExplore-AlGo, we performed three targeted ablations in the Screen–Moderate setting (Figure 3). We also report results over three random seeds to characterize variance.

Seed Variability. Across three runs of UIEx-AlGo, two seeds achieved comparable performance (within $\pm 5\%$ of the default), while Seed 1 underperformed, matching GUI-Bee‡ and heuristic-random. Inspection revealed that Seed 1 failed to execute the `create_project` macro in all three high-priority attempts (blank project, template import, continuation), due to novelty-based avoidance of retrying incomplete macros. This suggests that our current macro-completion feedback is insufficiently corrective, introducing high variance that lowers confidence in single-seed ablation interpretations. However, apart from Seed 1, all runs, including all ablations, successfully created a repository within the first 250 actions, indicating that early bootstrap success is a key driver of performance gap.

Macro-Action Horizon ($N_M = 1$). Setting N_M=1 (one macro before selecting a new frontier) yields performance nearly on par with baseline heuristics, as expected: frequent frontier re-selection increases the chance of quickly discovering high-value UI segments.

Atomic-Action Budget ($N_A = 2$). Capping atomic actions per macro to 2 produces a slight drop in overall coverage (–5% at 2 k steps). Qualitative traces show that the agent generates partial macros, trying to continue the previous intent, requiring stitching tasks such as "Create repository" via multiple macros. With only two atomic steps, completion intent often spills over additional selections, reducing efficiency. However, the decline is modest, which implies that our prompting encourages continuation of the previous intent (Sect. F).

Priority & Novelty Removal. When both macro-action prioritization (LLM importance) and novelty scoring are ablated, selecting frontier and in-state macros uniformly at random, we observe a significant performance collapse (–10% at 2 k steps). This confirms that prioritization is critical: without it, the agent drifts into redundant or low-value UI macro-actions.

7. Call for applications

We encourage the community to look beyond leaderboard scores and use UIExplore-Bench and its exploration dataset as a platform for broader research, including but not limited to:

Downstream Exploitation with Shared Exploration Data. Develop and compare algorithms that leverage our fixed exploration dataset for solving downstream tasks. For example, explore retrieval-augmented planning (RAG) for conditioning agents on prior exploratory experience to improve task-solving performance.

Self-Supervised Metrics. Design and evaluate new self-supervised metrics to measure exploration progress at test time, including metrics not reliant on privileged structural information.

Reinforcement learning. Train agents with RL to maximize UFO and learn a better exploration policy, without handcrafted novelty heuristics.

Intrinsic Reward Learning. Train agents to maximize exploration-specific rewards (e.g., novelty, coverage, or "assumption discovery"). For instance, our dataset enables agents to analyze macro-action outcomes: if an agent predicts a button is for "merge" but the actual outcome differs, it can detect such mistaken assumptions and build intrinsic rewards for discovering surprises. We can train policies that seek out and correct the agents own misconceptions.

We welcome creative uses of UIExplore-Bench for new agent designs, metric development, or experimental protocols. All resources are publicly released to enable reproducibility and community-driven advancement.

8. Discussion and Future Work

We introduced UIExplore-Bench, a dedicated benchmark for evaluating user interface exploration by autonomous agents. Motivated by a growing need in the community, UIExplore-Bench provides a standardized environment for isolating and studying exploration behaviors. We launch a suite of baselines alongside our proposed algorithm UIExplore-AlGo, to highlight key challenges and establish reference points for progress. UIExplore-AlGo combines novelty- and preference-based prioritization with a hierarchical approach to efficient, structured exploration.

Our current setup focuses on a single application (GitLab), which, while realistic, may not fully capture the diversity of UI exploration challenges. Moreover, our algorithm does not yet incorporate policy-based backtracking, which could further enhance performance.

Future extensions of UIExplore-Bench could include additional applications and introduce more subtle or less conventional interactive elements to probe agent generalization and robustness.

We believe that our work aligns with recent calls to move beyond static datasets and toward continuous, experience-driven learning (Silver & Sutton, 2025), which holds promise for agents capable of long-term adaptation and discovery.

We invite the community to contribute with new environments, metrics, and agents to UIExplore-Bench, and to consider our call for action Section 7. Strengthening UI exploration is a key step toward building truly autonomous and robust agents for real-world digital tasks.

Software, Data and Leaderboard

Benchmark Release. We will release the full UIExplore-Bench benchmark environments, our exploration data set, open source tools to load and explore the collected dataset and the reporting code, within two months of acceptance. **Public Leaderboard.** A continuously updated leaderboard will be hosted, ranking submissions by UI-Functionalities Observed at 2,000 steps, with additional reporting required at 500 and 1,000 steps. Scores must be provided separately for the DOM and GUI modes, and each submission must include a link to the trajectory data and, preferably, the code.

Limitations

Budget Constraints. Due to the limitations of the compute and API rate, our experimental evaluation was limited in the number of seeds, agent variants and ablation runs. For efficiency, we used a hybrid setup: GPT-4o for state description and macro-action generation, and Claude for action selection, enabling balanced querying under usage caps. Although this configuration was fixed across runs, it would be valuable future work to systematically evaluate different combinations of models for each component to better understand their respective contributions.

Scope. The benchmark currently targets a single application (GitLab) and supports exploration with goto commands. Although this reflects real-world usage, it may reduce exploration challenge compared to stricter navigation constraints. We plan to release a goto-free version to support more rigorous future comparisons.

Impact Statement

Positive Impact. This work aims to advance the study of structured exploration in user interfaces, enabling agents to autonomously build actionable knowledge from real-world environments. Potential applications include more reliable task automation, better user accessibility testing, and scalable collection of experience data for training downstream models.

Potential Risks. As with any tool that improves understanding of the environment, there is a risk of misuse such as aggressive scraping and automating spam-like interactions. However, our benchmark is designed for sandbox environments and focuses on measuring functional discovery rather than real-world automation. We encourage future work to build safeguards when deploying such agents broadly.

Acknowledgments

We thank Surbhi Kamboj and Tomasz Religa for their assistance in collecting the exploration data.

References

Agashe, S., Wong, K., Tu, V., Yang, J., Li, A., and Wang, X. E. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

Anthropic. Introducing computer use, a new claude 3.5 sonnet, and more. `https://www.anthropic.com/news/3-5-models-and-computer-use`, October 2024. Accessed: 2025-05-18.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

Bai, H., Zhou, Y., Pan, J., Cemri, M., Suhr, A., Levine, S., and Kumar, A. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37: 12461–12495, 2024.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Boisvert, L., Thakkar, M., Gasse, M., Caccia, M., Chezelles, T. L. S. D., Cappart, Q., Chapados, N., Lacoste, A., and Drouin, A. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks, 2025. URL `https://arxiv.org/abs/2407.05291`.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Chezelles, D., Le Sellier, T., Gasse, M., Lacoste, A., Drouin, A., Caccia, M., Boisvert, L., Thakkar, M., Marty, T., Assouel, R., et al. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024.

Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., Del Verme, M., Marty, T., Boisvert, L., Thakkar, M., Cappart, Q., Vazquez, D., et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847): 580–586, 2021.

Fan, Y., Zhao, H., Zhang, R., Shen, Y., Wang, X. E., and Wu, G. Gui-bee: Align gui action grounding to novel environments via autonomous exploration. *arXiv preprint arXiv:2501.13896*, 2025.

He, H., Yao, W., Ma, K., Yu, W., Zhang, H., Fang, T., Lan, Z., and Yu, D. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. *arXiv preprint arXiv:2410.19609*, 2024.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

Koh, J. Y., McAleer, S., Fried, D., and Salakhutdinov, R. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

Lee, S., Choi, J., Lee, J., Wasi, M. H., Choi, H., Ko, S., Oh, S., and Shin, I. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 1119–1133, 2024.

Li, Y., Zhang, C., Yang, W., Fu, B., Cheng, P., Chen, X., Chen, L., and Wei, Y. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*, 2024.

Lu, C., Hu, S., and Clune, J. Intelligent go-explore: Standing on the shoulders of giant foundation models. *arXiv preprint arXiv:2405.15143*, 2024.

Marreed, S., Oved, A., Yaeli, A., Shlomov, S., Levy, I., Sela, A., Adi, A., and Mashkif, N. Towards enterprise-ready computer using generalist agent. *arXiv preprint arXiv:2503.01861*, 2025.

Nica, A., Khetarpal, K., and Precup, D. The paradox of choice: Using attention in hierarchical reinforcement learning. *arXiv preprint arXiv:2201.09653*, 2022.

OpenAI. Computer-using agent (cua). `https://openai.com/index/computer-using-agent/`, 2025. Accessed: 2025-05-18.

Pahuja, V., Lu, Y., Rosset, C., Gou, B., Mitra, A., Whitehead, S., Su, Y., and Awadallah, A. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. *arXiv preprint arXiv:2502.11357*, 2025.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Qi, Z., Liu, X., Iong, I. L., Lai, H., Sun, X., Zhao, W., Yang, Y., Yang, X., Sun, J., Yao, S., et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227, 1991.

Schmidhuber, J. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on anticipatory behavior in adaptive learning systems*, pp. 48–76. Springer, 2008.

Shi, T. T., Karpathy, A., Fan, L. J., Hernandez, J., and Liang, P. World of bits: an open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 3135–3144. JMLR.org, 2017.

Silver, D. and Sutton, R. S. Welcome to the era of experience. *Google AI*, 2025.

Steccanella, L., Totaro, S., Allonsius, D., and Jonsson, A. Hierarchical reinforcement learning for efficient exploration and transfer. *arXiv preprint arXiv:2011.06335*, 2020.

Su, H., Sun, R., Yoon, J., Yin, P., Yu, T., and Arık, S. Ö. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. *arXiv preprint arXiv:2501.10893*, 2025.

Thrun, S. B. *Efficient exploration in reinforcement learning*. Carnegie Mellon University, 1992.

Wu, J., Yin, W., Jiang, Y., Wang, Z., Xi, Z., Fang, R., Zhang, L., He, Y., Zhou, D., Xie, P., et al. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*, 2025.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Yu, X., Peng, B., Vajipey, V., Cheng, H., Galley, M., Gao, J., and Yu, Z. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. *arXiv preprint arXiv:2410.02052*, 2024.

Zhang, R., Qiu, M., Tan, Z., Zhang, M., Lu, V., Peng, J., Xu, K., Agudelo, L. Z., Qian, P., and Chen, T. Symbiotic cooperation for web agents: Harnessing complementary strengths of large and small llms. *arXiv preprint arXiv:2502.07942*, 2025a.

Zhang, S., Chen, J., Gao, J., Gao, Z., Yi, X., and Li, H. Characterizing unintended consequences in human-gui agent collaboration for web browsing. *arXiv preprint arXiv:2505.09875*, 2025b.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023a.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023b.

Table 4. Scores for UI-FUNCTIONALITIES OBSERVED (UFO) at 2,000 steps for each level (Abundant, Moderate, Sparse), and mean scores across all levels at 500, 1,000, and 2,000 steps. Results are reported for a single seed.

				Levels Average		
Agent	**A@2k**	**M@2k**	**S@2k**	**@500**	**@1k**	**@2k**
Structured Mode						
DFS	375.0	508.0	57.0	179.3	241.0	313.3
random	1050.0	414.0	88.0	335.7	431.3	517.3
BFS	2132.0	726.0	137.0	648.3	919.7	998.3
heuristic-random	1689.0	602.0	118.0	523.3	651.3	803.0
UIExplore-AlGo	1914.0	1784.0	1821.0	1180.0	1447.7	1839.7
Screen Mode						
random	604.0	144.0	50.0	167.0	217.3	266.0
heuristic-random	1086.0	375.0	84.0	346.0	430.3	515.0
GUI-Bee‡	1123.0	415.0	85.0	368.3	450.7	541.0
UIExplore-AlGo	952.0	966.0	925.0	337.3	598.7	947.7

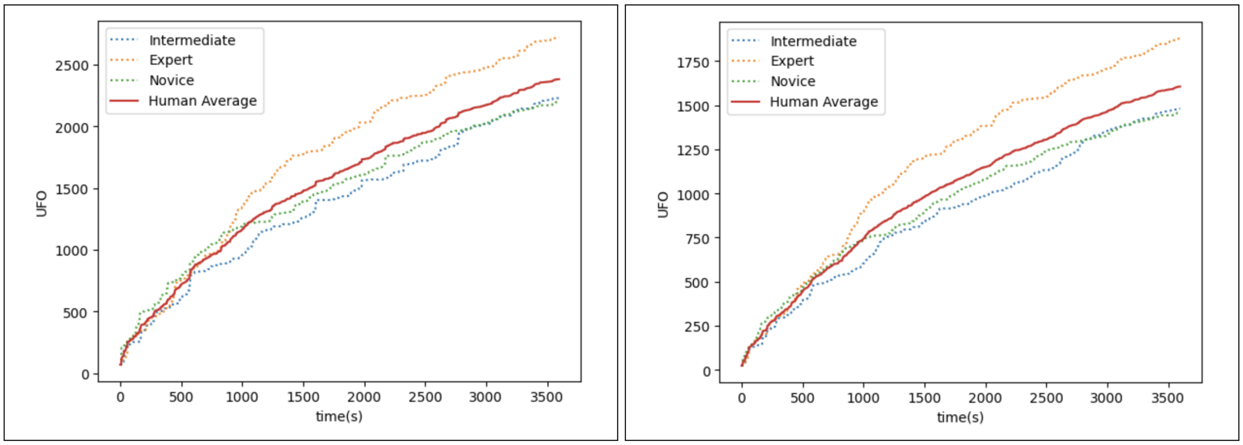


Figure 4. UFO by time in Structured mode (left) and Screen mode (right) for three human participants having various levels of expertise

A. Extended results

The absolute UI-FUNCTIONALITIES OBSERVED counts can be found in Table 4.

B. Human evaluation

We collected exploration trajectories from three human participants with different levels of expertise: novice, intermediate, and expert. Participants were asked to "explore the Gitlab website to discover the functionalities exposed by the platform" in Abundant level. The data were collected for approximately one hour. Observations were captured at a rate of at most one observation per second. The observation captures were paused while the web pages were being loaded to avoid redundant data captures.

The graph of human exploration over 1 hour can be found in Fig. 4. Since the different human annotators captured a slightly different number of observations, the x-axis has been renormalized to be 1 hour.

C. Metric Formalism

Our primary metric, UFO, is based on functionalities exposed by an application in a given observation. A functionality is defined as a group of individual actions that share transitional and visual characteristics.

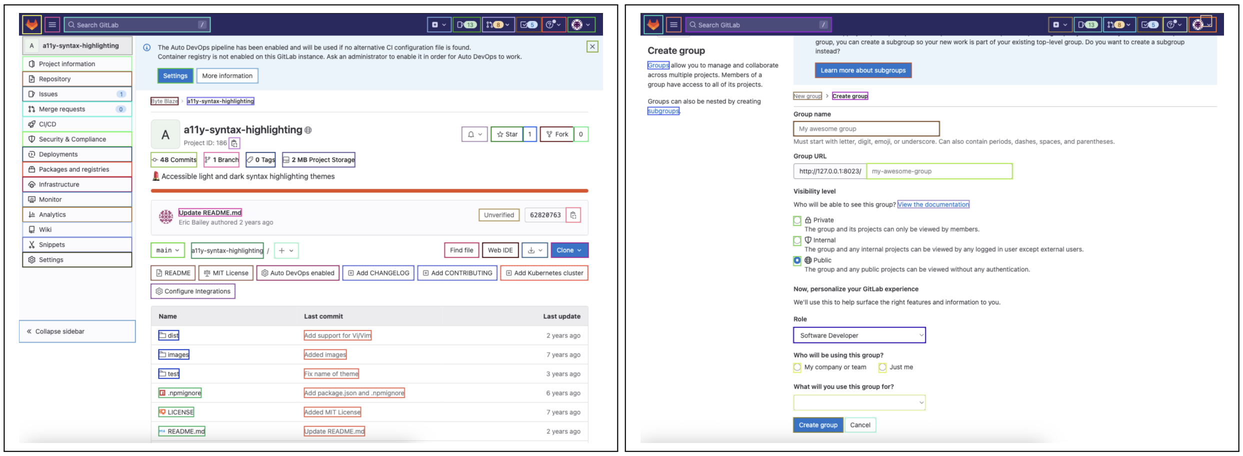


Figure 5. Functionality groupings in project page (left) and create group page (right). Within each image, boxes of the same color are grouped under the same functionality.

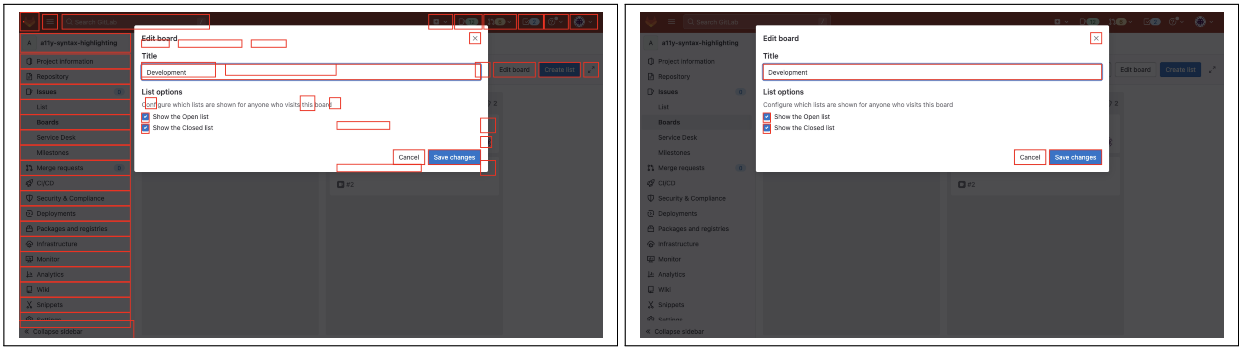


Figure 6. UFO Structured Mode (left) includes all functionalities present in the DOM regardless of their visibility and interactivity status and UFO Screen Mode (right) includes functionalities that are visible and interactive

For Gitlab environment, the transitional characteristics are extracted using DOM tags. All `button`, `input`, `select`, and `textarea` tags that are not disabled are considered as providing inputs for web state transitions. The `a` (link) tags are grouped based on URL-patterns specific to Gitlab. For example, links to `org1/project1` and `org2/project2` are treated to provide the same functionality – navigation to a project, `{org}/{project}`. The visual characteristics are extracted primarily by observing the class attribute associated with the DOM tags. However, this is insufficient for addressing complex styling within Gitlab. To address the corner cases, hand-crafted rules are applied for grouping visual characteristics based on DOM attributes. Additional examples of groupings on the Gitlab "project page" and "create group" can be found in Fig. 5.

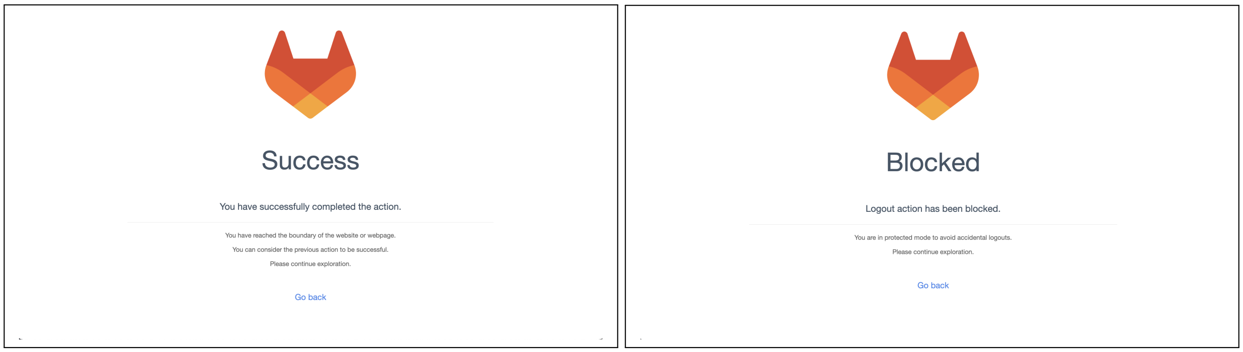
Our metrics can be applied for both the Structured mode and the Screen mode. The Structured mode considers all the functionalities that are present in the DOM structure of the webpage. Screen mode only includes functionalities that are visible and actionable within the browser's viewport. An example showing this difference can be found in Fig. 6.

D. Environment Details

UIEXPLORE-BENCH uses a customized version of the Webarena (Zhou et al., 2023b) Gitlab environment.

In order to contain agents within the environment, we redirect all access to external websites and file-upload dialog boxes to a boundary page. The boundary page displays a message stating that the action is to be considered successful, along with a home button and a go-back button. Additionally, in the Abundant and Moderate modes, all attempts to log-out are disabled and redirected to a log-out-disabled page. Fig. 7 shows the visual representation of these two pages.

Environment reset details.



Success

You have successfully completed the action.

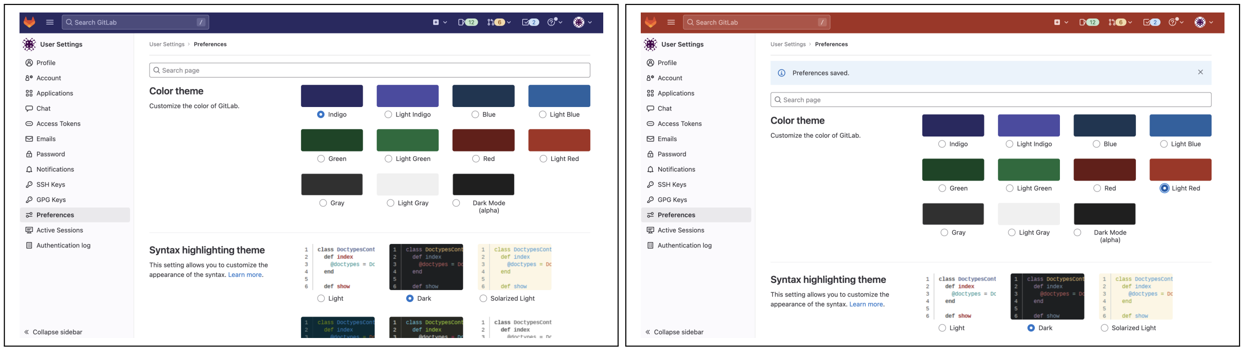
You have reached the boundary of the website or webpage.
You can consider the previous action to be successful.
Please continue exploration.

Blocked

Logout action has been blocked.

You are in protected mode to avoid accidental logouts.
Please continue exploration.

Figure 7. Boundary Page (left) and Logout-disabled Page (right)



User Settings › Preferences

Preferences saved.

Color theme

Customize the color of GitLab.

● Indigo (left) / ● Light Red (right)

Syntax highlighting theme

This setting allows you to customize the appearance of the syntax. Learn more.

● Dark

```
class DoctypesCont
  def index
    @doctypes = Dc
  end

  def show
```

Figure 8. Indigo theme (left) and Light Red theme (right) updates the DOM at `body` tag (depth 1)

1. Abundant mode resets to the default Webarena's Gitlab instance.
2. Moderate mode resets to the default Webarena's Gitlab instance followed by the deletion of all Gitlab projects. All users are retained.
3. Sparse mode resets to the default Webarena's Gitlab instance followed by the deletion of all Gitlab projects and users.

E. D3C Discussion

Fig. 8 shows an example in which changes in the user preference of `theme` affects D3C computation. Updating the `theme` modifies the `class` attribute of the `body` tag. Since this change occurs at depth 1, all pages visited under one `theme` is treated to be distinct from the same pages visited under another `theme`, thereby incentivizing the exploration agent to change themes often.

F. UIExplorer Agent Details

Our proposed algorithm, UIEXPLORE-ALGO, is designed for efficient and structured UI exploration, combining insights from hierarchical exploration strategies and novelty-driven search inspired by Go-Explore (Ecoffet et al., 2019). Its core features include macro-actions for high-level interaction, novelty-based prioritization, and systematic frontier state selection. We provide the pseudocode in Algorithm 1, and further detail its primary components and computation methodologies here.

Core Exploration Loop. The algorithm operates by repeatedly describing the current UI state using GPT-4o to generate textual descriptions of states and macro-actions (higher-level UI interaction tasks). These macro-actions encapsulate sequences of atomic UI interactions (e.g., filling a form, creating a repository). The agent alternates between exploring novel macro-actions from the current state and periodically revisiting frontier states identified as promising based on their exploration potential.

Knowledge Graph and Dataset. All states, macro-actions, and their resulting state transitions are recorded in a semi-structured dataset G, organized as a knowledge graph. We publicly release this dataset as part of our benchmark to facilitate downstream research, enabling agents to utilize past exploration experiences.

Novelty-Based Macro-Action Selection. Novelty-driven macro-action selection involves three primary considerations:

- Predictive State Novelty GPT-4o generates an anticipated future state description s_f for each candidate macro-action. Let $\mathcal{F}$ represent embeddings of predicted future state descriptions and $\mathcal{R}$ embeddings of actually experienced state descriptions. We compute novelty via cosine similarity:

$$\text{Novelty}_{state}(a) = 1 - \max_{r \in \mathcal{R}} \frac{f_a \cdot r}{\|f_a\|\|r\|}, \quad \text{where } f_a \in \mathcal{F}$$

- State-Action Pair Novelty We calculate novelty between candidate state-action embeddings $\mathcal{C}$ and embeddings of previously executed state-action pairs $\mathcal{P}$:

$$\text{Novelty}_{action}(a) = 1 - \max_{p \in \mathcal{P}} \frac{c_a \cdot p}{\|c_a\|\|p\|}, \quad \text{where } c_a \in \mathcal{C}$$

- Importance Ranking by GPT-4o GPT-4o also ranks macro-actions based on their predicted complexity and exploration depth potential. The top-ranked actions by GPT-4o are given additional priority to balance purely novelty-driven selection.

Macro-action Choice Macro-actions are selected in two stages: a novelty-based filtering step followed by GPT-based prioritization.

- Filtering.

- In the *frontier-based setting*, we select the top 20 macro-actions with the highest novelty scores.
- In the *local exploration setting*, we select the top 30% most novel macro-actions available at the current state.

- Prioritization. From the filtered set $\mathcal{A}_{\text{novel}} \subseteq \mathcal{A}$, the final macro-action is chosen based on GPT-4o's ranking:

$$a_M^* = \arg\max_{a \in \mathcal{A}_{\text{novel}}} \text{Rank}_{GPT}(a)^{-1}$$

That is, we select the most important macro-action (according to GPT-4o) among the most novel ones.

Frontier State Selection Periodically, the algorithm selects frontier states—previously encountered states judged promising based on novelty and potential exploration gains. We reuse the novelty computation described above, applied to all recorded states and their available macro-actions, choosing frontier states-actions with maximal exploration potential.

Macro-action Execution and Atomic Actions The execution of macro-actions is handled by a secondary agent, Claude, optimized empirically for the direct execution of low-level UI actions. Claude's responsibility is executing sequences of atomic actions until the macro-action goal is completed or a step limit N_A is reached.

Backtracking and Goal-Conditioned Navigation To efficiently revisit frontier states, we leverage a built-in `goto` function. For applications lacking this capability, future extensions could incorporate learned goal-conditioned navigation policies as suggested in related work (Ecoffet et al., 2021).

Implementation and Hyperparameters Full algorithm details, including prompt templates and additional hyperparameter settings, are available publicly alongside the benchmark release to facilitate reproducibility and extension.

F.1. Prompts

Describe state and macro-actions prompt:

```
<system_prompt>
You are an autonomous UI explorer.  Your goal is to map out an application's full feature set by
systematically driving it from its current state into new states that unlock yet more functionality.
```

~~~
Guiding principles:
1.  **Depth-first discovery** Order your actions by their likelihood of leading you deeper into the
app|unlocking new screens, flows, or features.
2.  **Complex over trivial** Prefer composite tasks (e.g.  \submit registration form") over atomic ones
(\click this icon").
3.  **Cover all unique interactive types** Ensure that each type of interactive element (e.g.  buttons,
links, icons, inputs) is exercised by at least one action.  You don't need a separate task for every
instance|group similar elements into a single representative action (e.g.  \click one repository link" or
\try clicking a like button").  If needed, use actions like \click one of the X buttons" or \click each X
button" to generalize.  Even if you don't know what an icon or button does, include an action to \interact
with" it.
4.  **Keep it in-app** Actions that navigate outside the app (external links, file uploads, downloads) go at
the end.
5.  **ASCII only** Use plain ASCII for all descriptions and actions.
6.  **Fallback on dead-ends** If you hit a missing page or error boundary, issue a \go back" action.
When you receive a state you'll reply in JSON:
```json  "state":  "<text description of the current screen, its inputs, icons, buttons and links>",
"actions":  [ "<highest-priority action>", "...  next action ...", "...  lowest-priority action ..." ]  ```
Your goal is to ensure all unique interactive element types are covered by at least one action, with grouping
used when multiple similar elements are present.
</system_prompt>
<user_prompt>
Current application state
state_description
Last action taken  last_action
Generate your JSON response based on the system rules above.  Focus first on sequences that open up new app
modules (e.g.  \complete registration"), then on other in-app features, then on external links.
</user_prompt>
~~~

Act-agent prompt for Structured Mode:

~~~
<system_prompt>
You are **AxBrowser-Agent**, a deterministic web-automation specialist.  You only see the page through an
**AXTree** snapshot that is refreshed after every action.
TASK
1.  Fulfil the user's goal.
2.  If the goal seems reached, finish with '<action>send_msg_to_user("completed")</action>'
3.  If you are stuck (e.g.  the last 2-3 actions had no effect, or the required element is missing) finish
with '<action>report_infeasible()</action>'
ACTION FORMAT
* Use the element number id (without the 'browsergym_id_' prefix) * Return **exactly** the following XML tags
per reply, nothing else:
```xml <previous_action_outcome>...</previous_action_outcome>
<action_description>...</action_description>
<action>click("element_id")</action> ```
Action space is provided below:
action_space
REASONING GUIDELINES (keep your chain-of-thought PRIVATE)
Always inspect the new AXTree (**Current page**) after every action.
Detect dropdowns/menus:  If the clicked element now has expanded="true" or you see new list / menu items,
assume a sub-menu opened.
Prefer selectors that match AX name, label, value, or role.  When several candidates exist, choose the one
closest in the tree to the element you just interacted with.
Avoid loops:  if the URL AND top-level AXTree snippet have not changed after two actions, rethink or quit.
Stop if the **Current page** content reflects the ambiguous goal, otherwise, continue exploring.
Tasks can be ambiguous or slightly wrong; interpret generously.
</system_prompt>
<user_prompt>
# Mini-task
goal_object
Recent history (last steps, from oldest to newest.  - n steps before.)
<action_history>
$forhinaction_history[:: -1]$
<step  h.idx >
acted= <previous_action h.idx >  h.action  </previous_action h.idx >
saw = <previous_observation h.idx >  h.observation  </previous_observation h.idx >
</step>
</action_history>
Current page
URL:  open_tabs
state_description
**Reply with the following XML tags, according to the instructions and action space:**
```xml
<previous_action_outcome>...</previous_action_outcome>
<action_description>...</action_description>
<action>...</action>
~~~
```
```

```
</user_prompt>
```

Act-agent prompt for Screen Mode:

~~~
<system_prompt>
You are **GUIBrowser-Agent**, a deterministic web-automation specialist.  You only see the page through a
**screenshot**.  You are a computer use agent.
TASK
1.  Fulfil the user's goal.
2.  If the goal seems reached, finish with `<action>send_msg_to_user("completed")</action>`
3.  If you are stuck (e.g.  the last 2-3 actions had no effect, or the required element is missing) finish
with `<action>report_infeasible()</action>`
ACTION FORMAT
Return **exactly** the following XML tags per reply, nothing else:
```xml <previous_action_outcome>...</previous_action_outcome>
<action_description>...</action_description>
<action>click("element_id")</action> ```
Action space is provided below:
action_space
REASONING GUIDELINES (keep your chain-of-thought PRIVATE)
Always inspect the new screenshot (Current page) after every action.
Detect dropdowns/menus:  If the clicked element now has expanded or you see new list / menu items, assume a
sub-menu opened which could give you further options for the goa.
Avoid loops:  if the visual state has not changed after two actions, rethink or quit.
Tasks can be ambiguous or slightly wrong; interpret generously.  You can also take actions that could lead
later to the goal.  Keep exploring until you think the goal has been reached.
</system_prompt>
<user_prompt>
# Mini-task
goal_object
Recent history (last steps, from oldest to newest.  - n steps before.)
<action_history>
<step  h.idx >
acted= <previous_action h.idx >  h.action  </previous_action h.idx >
saw = <previous_observation h.idx >  h.observation  </previous_observation h.idx >
</step>
</action_history>
Current page
URL:  open_tabs
state_description
Analyze first the **Current page** content to understand very well what was the outcome of the last action
and what is the current state of the page.
**Reply with the following XML tags, according to the instructions and action space:**
```xml
<previous_action_outcome>...</previous_action_outcome>
<action_description>...</action_description>
<action>...</action>
```
</user_prompt>
~~~

F.2. Agent exploration trajectory example

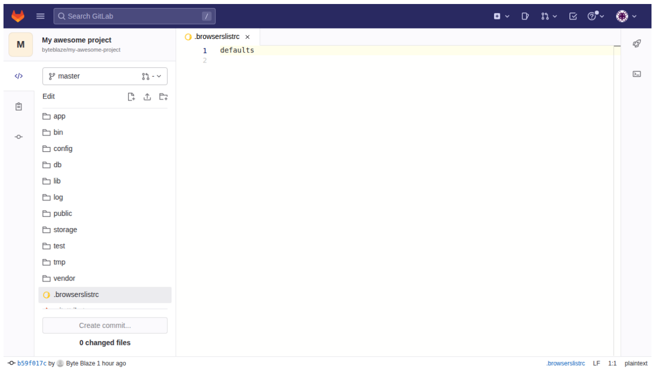
Partial Trajectory: Realizing Action Failure and Retrying

Step 1: Agent Observes the Current UI State

State Description and Macro-Actions

State: The application is currently displaying a GitLab repository pa titled ’My awesome project’. It is showing the file structure on the left (e.g., app, bin, config, db, etc.), a file editor in the center displaying the contents of a file named ’.browserslist’, and a highlighted commit button (’Create commit...’) at the bottom. The top navigation bar features search bar (’Search GitLab’), a project dropdown for branches or tags (currently ’master’), and various icons like settings, issues, and merge requests..
Available Macro-Actions:

- ”Interact with the branch/tag dropdown to explore branch-related features.”
- ”Click the ’settings’ icon in the top navigation bar to access project settings.”
- ”Click the ’issues’ icon to explore issue-related functionality.”
- ”Click the ’merge requests’ icon to explore merge request-related functionality.”



```
defaults
```

Figure 9. UI State: GitLab repository view with file tree and top navigation bar.

- ”Click on a specific file or folder (e.g., ’app’) in the file tree to navigate to its content.”
- ”Edit the text in the ’.browserslistrc’ file.”
- ”Interact with the commit history link in the bottom left corner to se previous commits.”
- ”Interact with the ’...’ menu next to the file name to explore additional file options.”
- ”Access the left sidebar toggle button to explore collapsing or expanding sidebar functionality.”
- ”Click the ’help’ or question mark icon in the top navigation bar to explore help/documentation options.”

STEP 2: AGENT SELECTS A MACRO-ACTION

Chosen Macro-Action

Goal: Click the ’settings’ icon in the top navigation bar to access project settings.

STEP 3: AGENT EXECUTES FIRST ATTEMPT

Attempt 1

Previous Outcome:

I am looking at a GitLab project page with a file browser interface. The top navigation bar contains several icons including notifications, merge requests, and settings.

Action Description:

I will click the settings/cog icon located in the top right corner of the navigation bar.

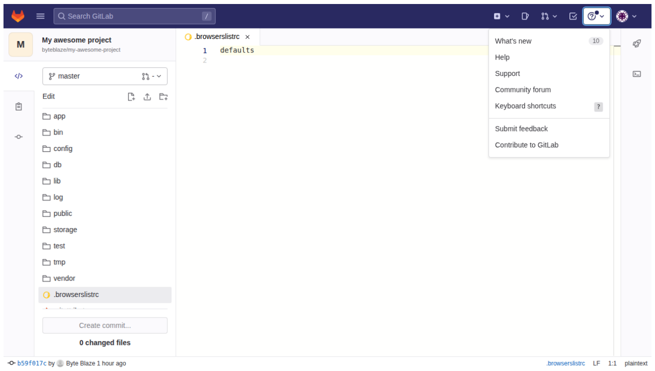
Atomic Action: `mouse_click(1171, 23)`

STEP 4: AGENT RECOGNIZES FAILURE AND RETRIES

Attempt 2 – Retry

Previous Outcome:

The previous click did not open the project settings. I’m still in the repository view showing the file browser interface. There’s a settings icon in the top navigation bar that may be in a different position than previously attempted.



```
defaults
```

Figure 10. Post-action: The UI did not change, agent remains in repository view.

Action Description:

I I will try clicking the settings icon (gear/cog icon) that appears to be located in the top navigation bar in a different position than previously attempted.

Atomic Action: `mouse_click(1170, 80)`