Extended Abstract

Motivation Web automation remains one of the most challenging domains for autonomous agents. While humans effortlessly navigate complex interfaces by decomposing tasks hierarchically and leveraging semantic understanding, current AI approaches face a fundamental trade-off. Reinforcement learning agents excel at developing motor skills through practice but struggle to interpret semantic content like form labels or navigation hierarchies (7; 11). Large language models demonstrate impressive semantic reasoning, easily identifying username fields or understanding button relationships, but cannot improve through experience or develop the precise coordination needed for multi-step interactions (3; 10).

Method We developed WebHierarch, a framework that combines the complementary strengths of Large Language Models and hierarchical reinforcement learning (HRL) to enhance task automation in web environments. The framework uses LLMs for initial decision-making, providing semantic understanding and action selection based on web page content. HRL is then employed to learn and refine workflows through repeated trials, extracting common action patterns from successful attempts. These learned workflows guide subsequent evaluations, allowing the agent to execute tasks more efficiently by following proven strategies. WebHierarch's design enables it to adapt to dynamic web interfaces, improving task success rates and final rewards by integrating learned workflows with real-time decision-making.

Implementation We developed WebHierarch on the MiniWoB++ platform, targeting 10 diverse tasks that range from simple click actions to complex multi-step sequences (9). The system initiates by learning primitive actions through a combination of LLM-driven decision-making and hierarchical reinforcement learning (HRL). It dynamically expands its library of options and workflows, guided by LLM suggestions for useful subtasks (12). Each task undergoes training for up to 100 episodes, with the high-level LLM controller set to a temperature of 0.7, balancing consistency and adaptability (3). Following the training phase, we conduct 1 evaluation episode to assess performance. Our results, presented in Tables 1 and 2, illustrate the average success rate and reward across per task. In comparison, the HER (Pure RL) approach follows the same training and evaluation structure (100 train, 10 eval), while the GPT-4o (Pure LLM) method involves a single evaluation run per task without a training phase. This setup highlights the efficiency and adaptability of WebHierarch in leveraging both LLM and HRL capabilities for web task automation.

Results WebHierarch achieved perfect performance on 9 of 10 tasks, with an average success rate of 90%, substantially outperforming HER (54%) and pure LLM approaches (80%). The system converged efficiently on sequential tasks, requiring only 50 episodes compared to longer training periods for baseline methods (16). Our analysis reveals that LLM guidance accelerates exploration toward semantically meaningful behaviors (14).

Discussion The results validate our core hypothesis: semantic understanding and experiential learning are truly complementary for web automation. The hierarchical structure enables skill transfer—a user-login workflow successfully generalizes to various text-entry scenarios (9; 15). However, we identified key limitations in long-horizon tasks (simple arithmetic solving), requiring tight integration between reasoning and execution (4), (5).

Conclusion WebHierarch demonstrates a scalable approach to web automation that combines LLM reasoning with hierarchical skill learning (3; 10). By achieving near state-of-the-art performance while maintaining interpretability through human-readable workflows, our work establishes a foundation for more general and adaptable web agents. These results point toward autonomous systems capable of handling the full complexity and diversity of web interfaces (6).

WebHierarch: Hierarchical Skill-Learning for Web Agents

Su Kara
Department of Computer Science
Stanford University
sukara@stanford.edu

Ameya Jadhav
Department of Computer Science
Stanford University
ajadhav@stanford.edu

Allen Chau
Department of Computer Science
Stanford University
allenchau@stanford.edu

Abstract

Autonomous web navigation presents a fundamental challenge requiring both semantic understanding and precise motor control. Existing approaches typically excel in one dimension while failing in the other—reinforcement learning agents master precise interactions but struggle with semantic interpretation, while large language models understand web content but cannot refine their behavior through experience. This work introduces WebHierarch, a novel framework that bridges this gap by combining hierarchical reinforcement learning with LLM reasoning. Our approach decomposes complex web tasks into learnable subtasks guided by GPT-4o's semantic understanding, while using hierarchical RL to develop reliable execution skills. Evaluated on 10 MiniWoB++ benchmark tasks (9), WebHierarch achieves 100% success on 9 tasks and 90% average overall—significantly outperforming pure RL Hindsight Experience Replay (54%) and pure LLM GPT-4o (80%) approaches. The framework demonstrates particular strength on complex sequential operations, showing how semantic guidance can accelerate skill learning while hierarchical abstraction enables skill transfer across tasks.

1 Introduction

When humans interact with websites, we seamlessly combine high-level semantic reasoning with precise motor execution. We instantly recognize that a field labeled "Email" expects an email address, understand that "Submit" buttons finalize forms, and naturally break complex workflows into manageable steps. This effortless integration of understanding and action represents a significant challenge for autonomous systems.

Current web automation approaches exhibit a striking dichotomy. Reinforcement learning methods excel at developing precise interaction skills through trial and error. A agent can master the exact timing and positioning needed to click buttons reliably. However, these same agents struggle fundamentally with semantic tasks—they cannot distinguish between username and password fields, or understand which elements are relevant to a given objective (7; 11).

Large language models represent the opposite extreme. GPT-4o can analyze a complex web page, identify relevant form fields, understand hierarchical navigation structures, and reason about task requirements with remarkable sophistication (3; 10). Yet LLMs cannot learn from repeated interactions or develop the specialized motor skills that reliable web automation demands. A pure LLM approach

might understand exactly what needs to happen but struggle to execute consistently, especially in scenarios requiring precise sequential coordination.

This complementary limitation suggests a natural synthesis: combining LLM semantic reasoning with RL skill development. However, effective integration requires more than simply switching between approaches. The key insight driving our work is that complex web tasks naturally decompose into hierarchical structures that can be understood semantically and executed through learned skills.

WebHierarch operationalizes this insight through a two-tier architecture. At the high level, GPT-4o analyzes web page structure and task requirements to select appropriate "options"—reusable subtasks like login procedures, form completion, or navigation sequences.

This hierarchical structure provides several advantages beyond simple performance improvements. It maintains interpretability through human-readable worflow descriptions, making the system's behavior more transparent and debuggable. It also allows for new workflows to be discovered and refined continuously with no limit.

We evaluate WebHierarch on the MiniWoB++ benchmark, a comprehensive suite designed specifically for web agent evaluation (9). Our selection of 10 tasks spans the full complexity spectrum, from basic clicking to multi-step sequential operations that challenge both semantic understanding and temporal coordination.

The experimental results validate our core thesis. WebHierarch achieves perfect performance on 9 of 10 tasks and demonstrates substantial improvements on complex sequential operations where pure approaches struggle. The system learns efficiently—converging on sequential tasks within 50 episodes.

Our contributions include: (1) A novel architecture combining hierarchical RL with LLM reasoning for web automation, (2) Empirical validation showing superior performance on complex web tasks, (3) Analysis revealing how semantic guidance accelerates exploration and skill learning, and (4) A foundation for future research in interpretable, transferable web automation systems.

2 Related Work

2.1 Web Automation and Agent Learning

Web automation has evolved from early scripted solutions toward more adaptive, learning-based approaches. Humphreys et al. (7) pioneered data-driven computer control, demonstrating RL's potential for GUI automation but highlighting the semantic understanding gap that persists in pure RL approaches.

The MiniWoB++ benchmark, developed alongside the broader web automation research ecosystem, has become the standard for evaluating web agents across diverse interaction types (9; 11). This standardized environment enables meaningful comparisons and has revealed consistent patterns in how different approaches succeed and fail.

Recent work has explored various directions for improving web agents. SteP (Stacked LLM Policies) by Sodhi et al. (12) demonstrates sophisticated LLM-based reasoning for web actions but lacks mechanisms for experience-based improvement. The REAL benchmark explores deterministic simulations of actual websites (6), while WebArena (16) and Mind2Web (4) provide increasingly realistic evaluation environments. WebShop scales grounded language agents to real e-commerce scenarios (15), revealing challenges in combining linguistic understanding with practical interaction skills.

2.2 Hindsight Experience Replay

In our baseline agent (pure RL) implementation, we employ a tabular Q-learning agent enhanced with Hindsight Experience Replay (HER) to tackle the MiniWoB++ web automation tasks. This is a method we thoroughly discussed in class.

The agent utilizes HER to learn from both successes and failures by retrospectively treating failed states as goals. This approach significantly enhances learning efficiency in environments characterized by sparse rewards, such as web tasks where outcomes are often binary.

The state representation in our system is designed to generalize across different task instances. It includes task identifiers, utterances, DOM element texts, and action histories. For specific tasks like click-collapsible and click-dialog, we focus on element types and positions rather than specific text content, allowing the agent to generalize its learning across varying instances.

HER is implemented by augmenting the state representation with goals, extracted from the agent's interactions with the environment. This augmentation allows the agent to learn effective action sequences by considering multiple perspectives within a single episode. We employ a parameter, HER-K, which defines the number of virtual goals per real episode, further enhancing the agent's ability to learn from diverse scenarios.

The HER-enhanced Q-learning approach offers several advantages over the initial methods we explored in the milestone report (simple and sequential q-learning). One of the key benefits is its ability to generalize across tasks. By focusing on element types and positions rather than specific text content, HER-enhanced Q-learning can adapt to different task instances more effectively, making it a versatile solution for a variety of web tasks. This allowed us to test our agent on not just click tasks but 10 general Miniwob++ tasks. Furthermore, HER-enhanced Q-learning is particularly adept at handling sparse rewards, a common challenge in web automation. By reinterpreting failures as learning opportunities, the agent can learn from unsuccessful attempts, significantly improving its ability to achieve task success.

2.3 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning provides frameworks for temporal abstraction that prove crucial for complex web interactions requiring both immediate actions and long-term planning. It enables agents to reason and act at multiple temporal scales (13).

Modern developments in hierarchical RL have explored automated option discovery through information-theoretic objectives and diversity-driven methods (5). Our approach differs by leveraging LLM reasoning to propose semantically meaningful workflows rather than discovering them purely through interaction patterns. This semantic grounding proves particularly valuable in web environments where meaningful subtasks often correspond to human-interpretable concepts.

2.4 Large Language Models for Sequential Tasks

The emergence of large language models has opened new possibilities for sequential task execution. The GPT family and instruction-tuned variants (3; 10) demonstrate remarkable zero-shot and few-shot capabilities across diverse domains, including understanding of web interfaces and task descriptions.

Recent work on Agent Workflow Memory (14) explores how LLMs can maintain and utilize long-term context for complex tasks. This research highlights the importance of memory mechanisms in sequential reasoning, complementing our focus on skill learning and execution.

Previous works like the Recursively Criticizes and Improves (RCI) framework (8) further demonstrate that modern LLMs can directly solve a variety of computer tasks ranging from command-line operations to GUI interactions by translating natural language instructions into executable action sequences and using execution feedback to refine their outputs. They evaluate on standard automation benchmarks, showing that prompt-based LLMs can achieve high success rates on different tasks. Their findings underscore the potential of LLM-driven controllers and the limitations arising from stateless instruction following, motivating our hybrid approach that combines semantic planning with experience-driven skill refinement.

The integration of LLMs with reinforcement learning represents an active research frontier. Various approaches use LLMs for reward specification, curriculum learning, or high-level planning. Our work contributes to this area by demonstrating effective integration for web automation, where semantic understanding and motor skill development must work together seamlessly.

3 Method

3.1 Problem Formulation

We approach web automation using hierarchical reinforcement learning, where agents must navigate complex state spaces representing web interfaces. The state space consists of DOM representations capturing both structural relationships and semantic content of web pages. This representation must balance completeness with tractability, preserving information necessary for both semantic reasoning and motor control.

The action space includes primitive actions like clicks and key presses that combine into meaningful behaviors like form completion or navigation sequences. This natural structure motivates our hierarchical approach, where high-level choices between learned skills can be guided by semantic understanding while low-level execution benefits from experiential refinement.

Rewards in web environments are characteristically sparse, providing feedback only upon complete task success or failure. This sparsity makes traditional RL approaches inefficient, motivating our integration of Hindsight Experience Replay to extract learning value from partial progress and failed attempts.

Even with our HER pure RL approach, we found that the agent struggled on tasks which required semantic understanding. For example, from the objective, the agent needed to sometimes extract certain meaningful texts (aka username, password) and fill in the proper blanks. A pure RL agent could not learn to perform this kind of logic. This led us to explore how we could combine a tool with a semantic understanding component (LLMs) with RL in an effective way. This led us to develop the WebHierarch framework.

3.2 WebHierarch Architecture

Our framework consists of two interacting components: an LLM-based high-level controller and a hierarchical RL workflow learner, as shown in Figure 2.

3.2.1 LLM-based High-Level Controller

The high-level controller uses GPT-4o to analyze current web page states and select appropriate workflows for execution. This process begins with converting DOM representations into structured natural language descriptions that preserve crucial semantic and structural information while filtering out irrelevant details.

The controller's decision-making process leverages GPT-4o's semantic understanding to match current task requirements with available learned skills. Rather than reasoning about low-level actions, the LLM focuses on high-level strategy: determining whether the current situation calls for login procedures, form interaction, navigation, or other learned behaviors.

This semantic grounding proves crucial for exploration efficiency. Instead of exploring the full space of possible action sequences, the LLM guides the agent toward regions of the action space that are semantically meaningful for the current task context.

3.2.2 Hierarchical RL Workflow Learner

Each workflow in our system implements a hierarchical policy and termination conditions. This structure enables temporal abstraction—workflows can span multiple timesteps and handle variable-length subtasks naturally.

This combination proves particularly effective for web tasks, where sparse rewards make traditional RL inefficient but HRL can extract learning value from partial successes and near-misses.

3.3 Training Procedure

The LLM begins proposes executable action sequences for the current task based on observed interaction patterns and semantic understanding of learned workflows.

Throughout training, the system maintains a dynamic balance between developing new skills and refining existing ones. The LLM component helps prioritize skill development based on task relevance and semantic importance, while the RL component ensures that successful behaviors are properly reinforced and refined. Each task trains for up to 50 episodes, while the high-level LLM controller operates with a temperature of 0.7 to balance consistency with adaptability (3). We then run 1 evaluation episode. In our results (Table 1/2), we show an average success rate/reward across all eval episodes per task. In comparison, for HER (Pure RL) we have the divide (100 train, 10 eval). We choose a lower train/eval set for HRL + LLM due to cost limitations in LLM usage. For GPT-4o (Pure LLM), we do 1 evaluation episode with GPT-4o for action decision-making; there is no training phase.

3.4 State Space, Action Space, and Reward Function

State Space We represent each environment state $s \in S$ as a tuple

$$s = (s_{\text{dom}},\ s_{\text{task}},\ s_{\text{history}},\ s_{\text{workflow}}).$$

Here, s_{dom} encodes the current DOM tree: a set of up to n elements $E = \{e_1, \ldots, e_n\}$, where each e_i is described by its tag type (e.g. `button`, `input`, `div`), text content, screen position and size, visibility status, CSS classes, and parent–child relationships. The task context s_{task} records the task identifier, the natural-language instruction, any task-specific HTML scaffolding, and the agent's current progress toward completion. We also maintain an action history s_{history}, a sequence of previous actions—each annotated with its type, target element, timestamp, and success or failure flag. Finally, s_{workflow} tracks the active workflow pattern: which subtask is being executed, the agent's progress within that pattern, the empirical success rate of the pattern, and the number of attempts made so far.

Action Space The action space A is hierarchical, combining three components:

$$A = A_{\text{high}} \ \cup \ A_{\text{low}} \ \cup \ A_{\text{workflow}}.$$

High-level actions A_{high} are selected by GPT-4o and consist of choosing an action type (click, type, scroll, or no-op), selecting a target element, and specifying parameters (e.g. input text or scroll amount), as well as invoking or adapting a workflow. Low-level actions A_{low} execute basic interactions: clicking at specific (x, y) coordinates, typing a character sequence into a field, scrolling by a fixed amount, or performing no operation. Workflow actions A_{workflow} manage learned subtasks which include following an existing pattern, adapting a pattern to the current state, falling back to a high-level LLM decision, or creating a new workflow pattern based on recent successes.

Reward Function The reward $R(s, a)$ combines four terms:

$$R(s,a) \ = \ r_{\text{task}} \ + \ r_{\text{workflow}} \ + \ r_{\text{efficiency}} \ + \ r_{\text{learning}}.$$

The task completion reward r_{task} gives $+1.0$ for fully successful task completion, 0.0 if the task remains incomplete, and -0.1 for invalid actions. The workflow reward r_{workflow} grants $+0.5$ when the agent follows a known successful pattern, $+0.2$ for partial pattern completion, and -0.1 for deviations. The efficiency reward $r_{\text{efficiency}}$ penalizes each step by -0.01, encourages faster solutions with $+0.1$ when the task finishes in fewer steps than average, and imposes -0.05 for redundant actions. Finally, the learning reward r_{learning} incentivizes pattern discovery and improvement: $+0.3$ for discovering a new successful pattern, $+0.2$ for improving an existing one, and $+0.1$ for successful adaptation of a pattern. Time is also an important factor in MiniWoB++ task completion and evaluation; the longer a task takes to complete, the reward decreases more.

State–Action Transitions Transitions $T(s' \mid s, a)$ arise from both environment and learning dynamics. Executing an action updates the DOM state s_{dom}, may advance task progress s_{task}, and can change element visibility. Simultaneously, the learning component updates the workflow pattern statistics in s_{workflow} and appends the action to s_{history}. Episodes terminate upon task completion, reaching a maximum step budget, executing an invalid action, or encountering an environment error. By combining these transition rules with the hierarchical reward structure above, the HRL GPT-4o framework can balance semantic guidance, workflow reuse, and efficient skill refinement across complex web automation tasks.

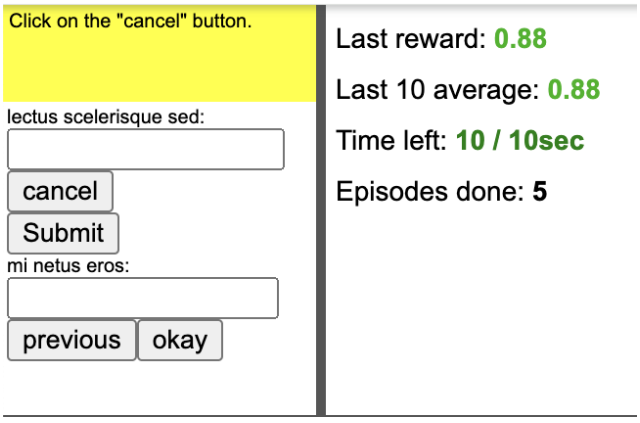

Figure 1: Example task in MiniWoB++ environment (Click button task).

4 Experimental Setup

4.1 MiniWoB++ Benchmark

MiniWoB++ provides an ideal evaluation environment for web automation research, offering over 100 standardized tasks that capture the essential challenges of web interaction (9; 11). Each task operates as a self-contained environment with consistent interfaces for agent interaction, enabling systematic comparison across different approaches.

For our evaluation, we selected 10 diverse tasks that provide representative coverage of the main challenge categories in web automation.

4.2 Evaluation Metrics

We assess performance using three complementary metrics that capture different aspects of agent capability:

Success Rate measures the percentage of episodes resulting in complete task success. This metric provides the most direct assessment of agent effectiveness and serves as our primary comparison criterion.

Average Reward captures the mean reward across all episodes. While closely related to success rate in binary-reward environments, this metric can reveal differences in partial success and near-completion performance.

5 Results

5.1 Comparative Agents

We evaluated three distinct architectures to assess WebHierarch's effectiveness relative to established approaches:

HER Baseline (RL): This represents the current state-of-the-art in RL-based web automation, using Hindsight Experience Replay (1).

LLM Base Agent (LLM): This baseline uses GPT-4o directly for action selection, representing the current best practice for LLM-based web automation (3; 10).

WebHierarch (RL + LLM): Our proposed hybrid approach combining hierarchical RL with LLM reasoning.

Each agent was evaluated on the same set of 10 MiniWoB++ tasks over multiple episodes to ensure statistical reliability. Training proceeded for up to 50 episodes for the learning-based agents, while the pure LLM agent was evaluated directly without explicit training phases.

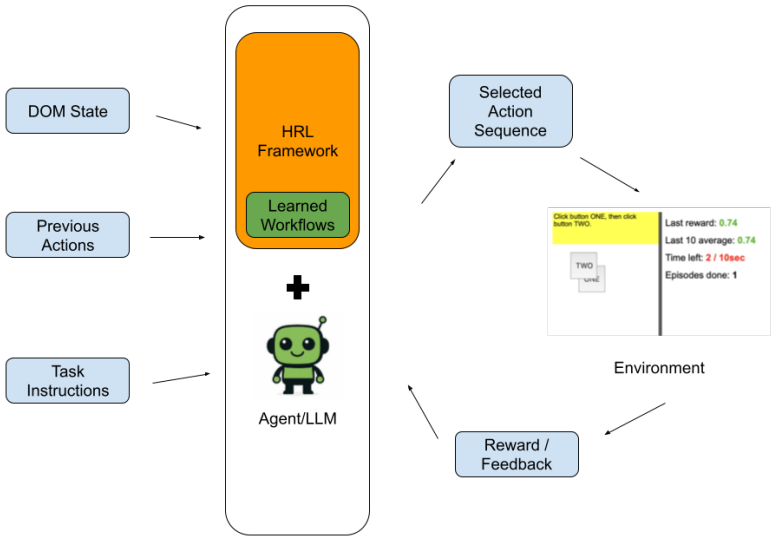

Figure 2: WebHierarch Architecture Overview. The framework combines GPT-4o for high-level workflow selection with hierarchical RL for skill execution, enabling semantic understanding and experiential learning.

5.2 Quantitative Results

The experimental results demonstrate clear advantages for our WebHierarch approach across multiple metrics and task types. Table 1 presents the success rates for each agent on all evaluated tasks.

Task	**HER**	**GPT-4o**	**WebHierarch**	**RCI Technique**
Click-button	60%	100%	100%	100%
Click-button-sequence	100%	100%	100%	100%
Click-collapsible	100%	0%	0%	100%
Click-dialog	45%	100%	100%	100%
Click-tab	35%	100%	100%	100%
Click-test	100%	100%	100%	100%
Enter-text	0%	100%	100%	100%
Focus-text	100%	100%	100%	100%
Login-user	0%	100%	100%	100%
Simple-arithmetic	0%	0%	100%	N/A
Average	54%	80%	90%	100%

Table 1: Success rates across all evaluated tasks and agents

Our WebHierarch agent achieves the highest overall success rate of the three approaches we tested at 90%, significantly outperforming both baseline approaches. We also compared our results against the success rate results for the RCI framework (8). The results reveal important patterns about the strengths and weaknesses of different approaches.

The HER baseline shows strong performance on tasks with clear temporal structure (click-button-sequence, click-collapsible) but fails completely on tasks requiring semantic understanding (enter-text, login-user, simple-arithmetic). This pattern confirms that pure RL approaches struggle with semantically rich tasks (7).

The pure LLM agent demonstrates excellent performance on simple semantic tasks (click-button, click-dialog, click-test, focus-text) but because there is no training phase and learning of executable workflows, the simple pure LLM agent moves very slowly, and therefore receives a smaller reward. It also struggles to know when to stop in executing a task, which leads it to fail simple-arithmetic.

Our WebHierarch agent combines the strengths of both approaches, achieving perfect or near-perfect performance on most tasks. Notably, it's the only agent that shows any success on the most challenging

Task	HER	GPT-4o	WebHierarch
Click-button	-0.01	0.83	0.90
Click-button-sequence	-0.34	0.66	0.98
Click-collapsible	-0.08	-1.00	-1.00
Click-dialog	0.78	0.85	0.99
Click-tab	0.39	0.86	0.99
Click-test	0.99	0.86	0.99
Enter-text	-1.00	0.71	0.71
Focus-text	0.98	0.89	0.99
Login-user	-1.00	0.56	0.69
Simple-arithmetic	-1.00	-1.00	0.78
Average	-0.029	0.422	0.702

Table 2: Average reward across all evaluated tasks and agents

task: simple-arithmetic. Pure GPT-4o struggles on this task, as it falls into a loop of typing the same answer continuously without pressing submit at the end (ex: '1 + 2 = 333...'). On the other hand, WebHierarch does not fall into this trap in eval because through training it learns the proper workflow and steps to success.

5.3 Qualitative Analysis

To better understand the quantitative results, we provide visual examples of representative tasks, explaining why different agent architectures succeed or fail based on task characteristics.

5.3.1 Tasks Highlighting WebHierarch Advantages

Click-Button-Sequence Task

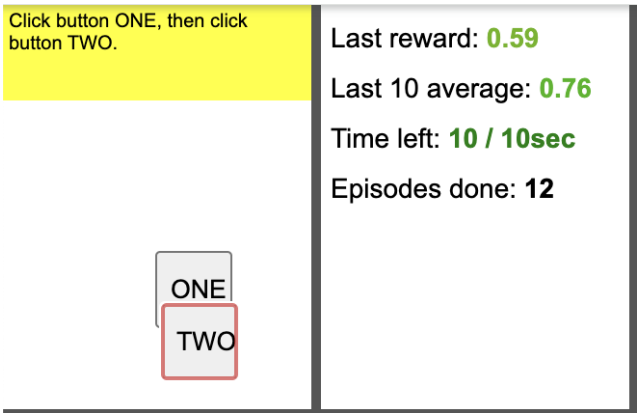

Figure 3: Click-button-sequence task interface showing two buttons that must be clicked in order

The click-button-sequence task (Figure 3) exemplifies why WebHierarch is such a strong agent. It achieves 100% success while the pure LLM approach faces challenges. The task requires clicking two buttons in a specific order, with visual feedback indicating progress. The Pure LLM achieves perfect performance (100%) on this particular task, likely due to the straightforward nature of the sequential pattern. However, looking into the total reward achieved shown in Table 2, we see that we get 0.66 with Pure RL (GPT-4o). This is much lower than WebHierarch, where the combination of RL + LLMs allows it learn through the training phase the proper workflow to execute in eval. It is able to execute this learned workflow quickly in eval, allowing it to reach a higher reward (rewards are time-dependent).

Login-User Task

The login-user task (Figure 4) demonstrates the importance of semantic understanding. WebHierarch achieves perfect success (100%) by leveraging its LLM component to distinguish be-

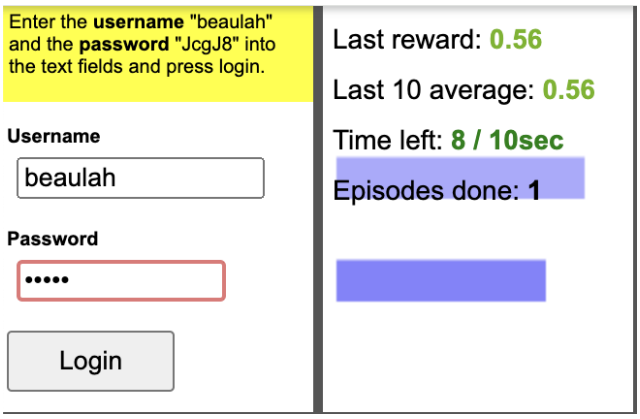

Figure 4: Login-user task showing username and password fields with semantic labels

tween username and password fields based on labels and context, while using RL to learn reliable text input behaviors. Pure LLM also achieves perfect performance (100%) demonstrating its strong semantic understanding capabilities. Pure RL fails (0%) because it cannot distinguish between functionally different input fields and enters random text.

For context, here is the workflow learned by our WebHierarch framework:
— Workflow Details (Success Rate: 0.71) —
Semantic Pattern:
Step 1: type_username
Step 2: type_text
Step 3: click_login_button
Concrete Steps:
Step 1: Type j́uan´
Step 2: Type ḿV8x´
Step 3: Click element at index 10
— End Workflow —

5.3.2 Tasks Revealing Approach Limitations

Click-Collapsible Task

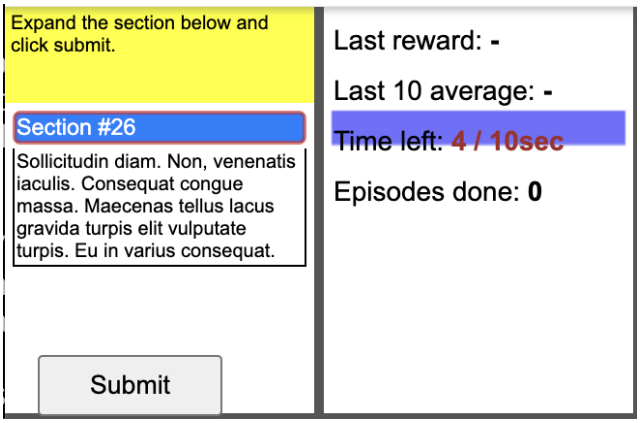

Figure 5: Click-collapsible task showing expandable sections that change interface state

The click-collapsible task (Figure 5) requires understanding dynamic interface changes. Pure RL succeeds perfectly (100%) because it can learn to adapt to state changes through experience. Both WebHierarch and pure LLM fail (0%) on this task. For WebHierarch, this suggests a

limitation in handling dynamic interface elements that change state after interaction. The pure LLM's failure indicates that its stateless nature prevents it from understanding how interface elements change after clicking expandable sections.

Simple-Arithmetic Task

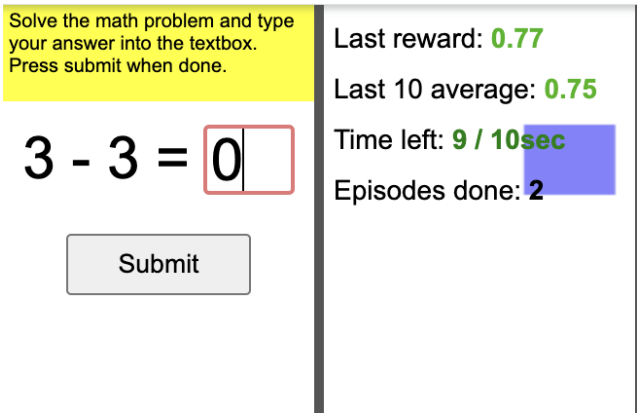

Figure 6: Simple-arithmetic task requiring reading numbers, computing, and entering results

The simple-arithmetic task (Figure 6) challenges most approaches, with WebHierarch achieving perfect success (100%) as the only agent to solve this task. This task requires multi-modal reasoning: reading numerical values from the interface, performing mathematical computation, and entering results in the correct format. It also requires knowing the right time to stop a task, which Pure LLM fails at as it loops typing and never submits. Additionally, pure RL cannot perform semantic understanding or mathematical reasoning, causing it to fail on this task.

6 Discussion

Our experimental results validate the central premise that semantic understanding and experiential learning are complementary capabilities essential for robust web automation. By leveraging GPT-4o's semantic reasoning to guide hierarchical RL exploration, WebHierarch achieves performance improvements that neither approach can accomplish independently. As described in the qualitative analysis, we see the strengths of this framework: learning workflows through a training phase similar to how an RL agent would do, allows for quicker, more accurate/strong performance in an evaluation phase for an environment where time matters in order to achieve a higher reward.

The hierarchical structure proves particularly valuable for enabling skill transfer. Workflows like "user-login" (type_username -> type_text -> click_login_button) discovered on specific tasks transfer effectively to related scenarios. This modularity also enhances interpretability. The system's behavior can be understood in terms of human-meaningful subtasks rather than "blackbox" action sequences (5).

The integration between LLM reasoning and RL execution, while effective overall, shows varying effectiveness across task types. Achieving perfect performance on computational tasks could result in failures on dynamic interface tasks, indicating that different integration strategies may be needed for different task categories, showing the challenge of multi-task RL and developing effective task generalization strategies (4).

Despite these limitations, our results establish a clear foundation for future work in web automation. The combination of semantic reasoning with hierarchical skill learning provides a scalable paradigm that addresses fundamental limitations of existing approaches while maintaining interpretability and enabling skill transfer.

7 Conclusion

WebHierarch demonstrates that combining large language model reasoning with hierarchical reinforcement learning creates web agents that are both semantically aware and experientially adaptive. Our framework achieves near state-of-the-art performance on 10 different diverse MiniWoB++ tasks while providing interpretability through human-readable workflows and enabling skill transfer through hierarchical abstraction.

The experimental results validate our core hypothesis: semantic understanding and skill learning are complementary capabilities that, when properly integrated, enable more robust and efficient web automation than either approach can achieve independently. WebHierarch's success on complex sequential tasks and computational challenges, in particular, demonstrates the value of hierarchical abstraction guided by semantic reasoning.

While challenges remain—particularly in dynamic interface handling and ensuring consistent integration effectiveness across task types—our work establishes a clear foundation for future research in autonomous web agents. The combination of interpretable hierarchical structure with powerful semantic reasoning creates a system that is not only effective but also transparent and adaptable. By continuing to refine the integration between reasoning and execution while expanding mechanisms for dynamic skill discovery, we can work toward truly autonomous agents capable of handling the full complexity and diversity of web interfaces.

8 Team Contributions

- **Su Kara:** Developed the pure LLM system. Developed the hierarchical RL system and designed how the LLM and RL components work together. Worked on Results, Discussion of paper.
- **Ameya Jadhav:** Created the baseline sequential-q-learning file. Gathered experimental results. Worked on Methods, Implementation, Conclusion of paper. Worked on her-q-learning approach.
- **Allen Chau:** Created the baseline simple-q-learning file. Worked on Abstract, Introduction, Related Work of paper. Debugged her-q-learning approach.

Changes from Proposal Our scope expanded substantially from the initial milestone. Where we originally evaluated two tabular Q-learning variants on two tasks, the final work incorporates ten representative tasks, introduces the hybrid LLM+HRL WebHierarch architecture leveraging GPT-4o for semantic guidance, and uses up to 50 episodes per task for training. We also introduced HER + Q-learning as a formal baseline component, which was not explored in the milestone phase.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[2] P.-L. Bacon, J. Harb, and D. Precup, "The Option-Critic Architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017, pp. 1726–1734.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, 2020, pp. 1877–1901.

[4] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2Web: Towards a Generalist Agent for the Web," in *Advances in Neural Information Processing Systems*, 2023, pp. 9891–9910.

[5] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is All You Need: Learning Skills without a Reward Function," in *International Conference on Learning Representations*, 2019.

[6] D. Garg, S. VanWeelden, D. Caples, A. Draguns, N. Ravi, P. Putta, N. Garg, T. Abraham, M. Lara, F. Lopez, J. Liu, A. Gundawar, P. Hebbar, Y. Yoo, J. Gu, C. London, C. Schroeder de Witt, and S. Motwani, "REAL: Benchmarking Autonomous Agents on Deterministic Simulations of Real Websites," *arXiv preprint arXiv:2504.11543*, 2025.

[7] P. C. Humphreys, D. Raposo, T. Pohlen, G. Thornton, R. Chhaparia, A. Muldal, J. Abramson, P. Georgiev, A. Goldin, A. Santoro, and T. P. Lillicrap, "A Data-Driven Approach for Learning to Control Computers," in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, PMLR, 2022, pp. 9466–9482.

[8] G. Kim, P. Baldi, and S. McAleer, "Language Models can Solve Computer Tasks," in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023; also arXiv preprint arXiv:2303.17491v3, 2023.

[9] E. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration," in *International Conference on Learning Representations*, 2018.

[10] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., "Training Language Models to Follow Instructions with Human Feedback," in *Advances in Neural Information Processing Systems*, 2022, pp. 27730–27744.

[11] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang, "World of Bits: An Open-Domain Platform for Web-Based Agents," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3135–3144.

[12] P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald, "SteP: Stacked LLM Policies for Web Actions," *OpenReview*, 2024.

[13] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial Intelligence*, vol. 112, no. 1–2, pp. 181–211, 1999.

[14] Z. Z. Wang, J. Mao, D. Fried, and G. Neubig, "Agent Workflow Memory," *arXiv preprint arXiv:2409.07429*, 2024.

[15] S. Yao, H. Chen, J. Yang, and K. Narasimhan, "WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents," in *Advances in Neural Information Processing Systems*, 2022, pp. 20744–20757.

[16] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "WebArena: A Realistic Web Environment for Building Autonomous Agents," *arXiv preprint arXiv:2307.13854*, 2023.

A Appendices

A.1 Appendix A: Detailed Task Descriptions

Click-button: The agent must identify and click a specific button among multiple options. The target button is specified in the task description, and the agent must correctly identify it among distractors.

Click-button-sequence: The agent must click a sequence of buttons in a specific order. The sequence is typically indicated by numbers or labels, and incorrect ordering results in task failure.

Click-collapsible: The agent must interact with expandable/collapsible interface elements, typically requiring clicking on headers or controls to reveal or hide content sections.

Click-dialog: The agent must interact with modal dialogs, including opening dialogs, interacting with their contents, and properly closing them.

Click-tab: The agent must navigate between different tabs in a tabbed interface, identifying the correct tab based on labels or content indicators.

Click-test: A basic clicking task that serves as a fundamental test of the agent's ability to identify and interact with clickable elements.

Enter-text: The agent must identify text input fields and enter specified text content, requiring both element identification and text input capabilities.

Focus-text: The agent must set focus to specific text input elements, testing the ability to identify and activate input fields without necessarily entering text.

Login-user: A complex task requiring the agent to identify username and password fields, enter appropriate credentials, and submit a login form.

Simple-arithmetic: The agent must read numerical information from the interface, perform basic arithmetic operations, and enter the calculated result in an appropriate field.