WEBSIGHT: A Vision-First Architecture for Robust Web Agents

Tanvir Bhathal*
Stanford University
tanvirb@stanford.edu

Asanshay Gupta*
Stanford University
asanshay@stanford.edu

Abstract

We introduce WEBSIGHT, *a vision-based autonomous web agent, designed to interact with web environments purely through visual perception, eliminating dependence on HTML or DOM-based inputs. Central to our approach we introduce our new model,* WEBSIGHT-*7B, a fine-tuned vision-language model optimized for UI element interaction, trained using LoRA on a web-focused subset of the Wave-UI-25K dataset.* WEBSIGHT *integrates this model into a modular multi-agent architecture, comprising planning, reasoning, vision-action, and verification agents, coordinated through an episodic memory mechanism.*

WEBSIGHT-*7B achieves a top-1 accuracy of 58.84% on the Showdown Clicks benchmark, outperforming several larger generalist models while maintaining lower latency. The full* WEBSIGHT *agent achieves a 68.0% success rate on the WebVoyager benchmark, surpassing systems from labs such as OpenAI (61.0%) and HCompany (Runner H, 67.0%). Among tasks completed,* WEBSIGHT *answers correctly 97.14% of the time, indicating high precision. Together,* WEBSIGHT *and* WEBSIGHT-*7B establish a new standard for interpretable, robust, and efficient visual web navigation.*

1. Introduction

Autonomous web agents capable of performing complex web navigation tasks—such as automated form filling, online shopping, and dynamic information retrieval—have emerged as a critical area of study within artificial intelligence. Over recent years, substantial progress has been driven by leveraging large language models (LLMs), enabling browser agents to interpret web content primarily through textual representations like HTML code, Document Object Model (DOM) trees, and accessibility metadata [39, 65, 67]. Despite their successes, these approaches present critical challenges when confronted with real-world scenarios. Specifically, websites frequently feature incomplete or incorrect metadata, dynamic layouts, and complex designs that degrade the reliability of structurally-dependent agents [58, 66]. Furthermore, the computational demands of processing extensive textual and structured inputs limit their scalability and interpretability, hindering practical deployment [71, 35, 20].

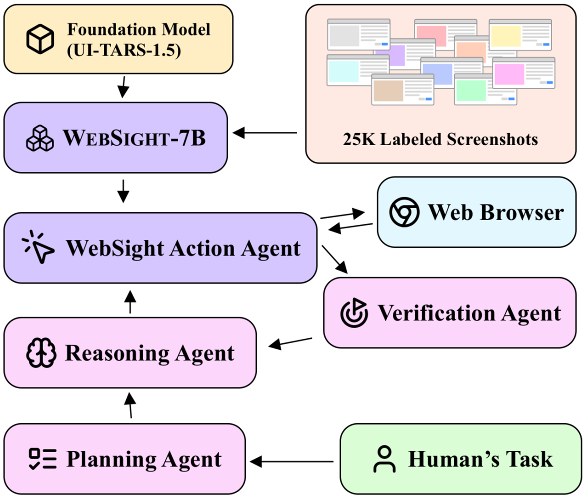


Figure 1. The WEBSIGHT Architecture

In contrast, human users rely almost exclusively on visual perception, effortlessly recognizing actionable interface elements like buttons, input fields, and navigation bars based solely on visual layout and design cues, irrespective of underlying metadata [17, 70]. Inspired by this natural modality, we present two tightly integrated contributions: WEBSIGHT, a vision-based autonomous browser agent; and WEBSIGHT-7B, a task-specific vision-language model trained to identify and interact with user interface elements directly from rendered web screenshots.

The WEBSIGHT agent is built on a robust multi-agent orchestration framework that mimics human cognitive processes. At its core are dedicated planning agents, which devise comprehensive strategies outlining task sequences and continually track progression towards goals [29, 50]. Concurrently, specialized reasoning agents perform detailed analyses to pinpoint and articulate precise subsequent actions necessary for task advancement [6, 20]. The critical component of our architecture, the vision agent, leverages the WEBSIGHT-7B model [45, 27] to make decisions on actions in the browser. Post-action, dedicated verification agents rigorously evaluate the resultant webpage state changes to ascertain accuracy and effectiveness of each interaction [58, 66]. This integrative process is further bolstered by an episodic memory mechanism, dynamically updating and iteratively refining agent strategies through continuous verification cycles until completion criteria are achieved [52, 35, 71]. Collectively, this orchestrated multi-agent architecture closely mirrors human cognitive and perceptual workflows, significantly enhancing interpretability, adaptability, and robustness [17, 70].

At the core of this system, WEBSIGHT-7B is a fine-tuned version of UI-TARS that we adapt to the web domain [45]. Trained on augmented web-based GUI datasets using LoRA fine-tuning, WEBSIGHT-7B demonstrates substantial improvements in English-language understanding and precision UI interaction over generalist vision-language models [27]. We show that the combination of architectural modularity and a domain-specialized foundation model enables both strong performance and practical deployability.

To validate our contributions, we evaluate WEBSIGHT and WEBSIGHT-7B on two challenging benchmarks: WebVoyager and Showdown Clicks [22, 55, 59]. WEBSIGHT achieves a 68.0% success rate on WebVoyager, outperforming comparable agents by 1–7 percentage points. On the Showdown Clicks benchmark, WEBSIGHT-7B attains a top-1 accuracy of 58.84%, surpassing a range of larger general-purpose vision-language models by 4–7 points. These results underscore the benefits of a vision-first architecture and domain-specialized model design for effective and efficient web interaction.

Through WEBSIGHT, we introduce a dual innovation: a vision-first web agent and an accompanying domain-optimized vision-language model. Together, advance autonomous web agent research by explicitly integrating human-like visual perception into agent design. Our approach offers a path toward robust, interpretable, and computationally efficient autonomous agents capable of generalizing across diverse web environments, aligning closely with human browsing behavior and intuitions.

We publicly share WEBSIGHT's code on Github: https://github.com/SuperAce100/websight and WEBSIGHT-7B on HuggingFace: https://huggingface.co/tanvirb/websight-7B.

2. Related Works

Early web navigation agents predominantly relied on reinforcement learning techniques, demonstrating basic task execution capabilities [33, 60, 7]. The advent of deep reinforcement learning further enhanced agent capabilities, enabling more sophisticated interaction strategies [38, 51, 25, 61]. However, these agents often required extensive training environments and struggled with generalization to unseen websites.

More recent advancements in large language models (LLMs) such as GPT series [8, 42, 40] have significantly improved autonomous agents' abilities to generalize and handle complex instructions [73, 36, 65]. Nonetheless, these approaches remain heavily reliant on structured textual inputs such as DOM trees and accessibility metadata, limiting their robustness against incomplete or inaccurate metadata [58, 66, 71, 35].

Parallel to textual methods, computer vision approaches have significantly advanced, particularly in semantic segmentation and object detection [24, 49, 14, 12, 69]. Recent progress in visual transformer architectures and self-supervised learning has further pushed the boundary of vision-based perception capabilities [18, 13, 46, 4, 62].

Integrating visual perception into web navigation has also garnered attention. Early work utilized vision heuristics for webpage understanding [11, 31, 16]. More recent methods have employed deep learning models to directly interpret visual webpage layouts, significantly improving navigation robustness [70, 30, 48, 37].

Our multi-agent orchestration approach builds upon foundational theories in multi-agent systems and planning [29, 50, 64], cognitive architectures [2, 32], episodic memory integration [5, 44], and human-inspired verification strategies [38, 54, 34] to achieve more robust, interpretable, and efficient autonomous web agents.

3. Data

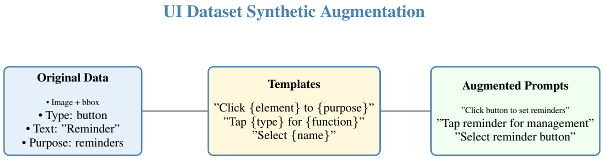

UI Dataset Synthetic Augmentation
Original Data
• Image + bbox
• Type: button
• Text: "Reminder"
• Purpose: reminders
Templates
"Click {element} to {purpose}"
"Tap {type} for {function}"
"Select {name}"
Augmented Prompts
"Click button to set reminders"
"Tap reminder for management"
"Select reminder button"

Figure 2. Synthetic augmentation pipeline for UI vision-language model training. Original structured UI data is processed through natural language templates to generate diverse instruction variations.

We utilize AgentSea's Wave-UI-25K dataset [1]. Wave-UI-25K comprises of 24,978 entries. Of this, we use the subset of 22,994 web-based screenshots with annotated browser interaction elements, as we are not focusing on alternative subsets like mobile and desktop.

Each dataset entry includes a screenshot, the bounding box of a particular UI element within it, OCR extracted text from the image, and a natural language description of the purpose of the element and expected result from interacting with the element.

In Figure 2 we optimize the responsiveness of WebSight to a variety of natural language prompts by synthetically augment the dataset with procedurally generated natural language prompts based on the semantic labels.

Fine-tuning allows the UI-TARS model to refine its attention and enhance task-specific efficacy, leveraging existing visual representation capabilities to robustly handle nuanced web interactions [15, 26, 47, 45]. We also ensure the model's proficiency in delivering precise English instructions and interactions specifically tailored for web environments (Section 4.2.1).

We evaluate WebSight using established benchmarks such as Showdown Clicks [59], which provides a robust test to determine accuracy of click locations of a VLM used in a Browser Agent. Additionally, we utilize Skyvern AI's filtered WebVoyager dataset [55], an enhanced evaluation set derived from the original WebVoyager dataset [23]. Skyvern's dataset specifically removes tasks deemed impossible due to outdated web page structures, ensuring more accurate and meaningful performance assessments.

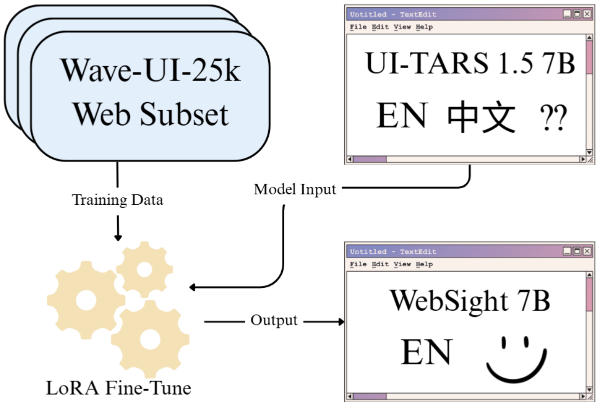


Figure 3. UI-TARS LoRA Fine Tuning Process to develop WebSight-7B. Leverages Web Subset of Wave-UI-25k to make UI-TARS English and browser specific.

4. Methods

WebSight is a visually-grounded multi-agent framework specifically designed to emulate human browser navigation behaviors. Unlike traditional LLM-based agents that rely heavily on structured textual inputs such as HTML or accessibility metadata, WebSight leverages purely visual inputs—webpage screenshots—to interact effectively with web environments.

4.1. Preliminaries

Baselines To effectively benchmark WebSight's performance we determine the effectiveness of both WebSight-7B and the WebSight agent. For WebSight-7B we compare its performance against state-of-the-art (SOTA) Vision-Language Models (VLMs), all of which are transformer-based and natively support multimodal input. These include models such as GPT-4o, Gemini 2.0 Flash, and Claude 3.7 Sonnet, as well as other large-scale VLMs and vision-capable LLMs. For the WebSight agent, we compare it to other SOTA web agents, whether they utilize vision or not. These agents include, Browserable, Browser Use, Skyvern 2.0, Agent-E, as well as many more by other cutting edge startups.

Problem Definition We formalize the problem using the following notation. Given an initial task β specified by the user, we assign it to the agent's episodic memory $\mathcal{M}$ and initialize the task state as $\mathcal{T}_0 = \beta$. Based on $\mathcal{T}_0$, the system generates a plan $\mathcal{P}$ that WebSight executes. To select actions a_t, the reasoning agent f processes the current status inputs, incorporating relevant information from $\mathcal{V}$. Once f determines that the task is complete, we mark the final state as $\mathcal{T}_{complete}$.

4.2. WebSight-7B

We present WebSight-7B, a Vision Language Model trained for targeted interaction with UI elements on screen given natural language prompts. We integrated this model into our subsequent multi-agent pipeline. We detail our process of developing the model below:

4.2.1 Fine-tuning

In Figure 3, we fine-tune UI-TARS-1.5-7B [45] using the Wave-UI-25K dataset [1]. Each training sample consists of a procedurally generated natural language instruction and a website screenshot. Originally, UI-TARS outputs frequently in Chinese and is too generalized to GUI interactions, leading to performance issues in English settings and the web domain. Our fine-tuning explicitly enhances its capability to respond to focused English instructions and accurately perform browser-specific interactions. Fine-tuning improves domain-specific performance by adapting general-purpose visual representations to specialized tasks [26].

We fine-tune UI-TARS using LLaMA-Factory [72] with bf16 quantization and a LoRA approach on the attention layers for faster training. Our fine-tuning leverages supervised learning with a cross-entropy loss for instruction generation. Within this setting, training on 2 NVIDIA L40S GPUs takes about 6 hours.

Further, from Figure 9 we see that a majority of the learning is done in the beginning of the process, with the returns tapering quickly. This is due to the web browser and english support already existing in UI-TARS and the fine tuning process only supporting and enhancing this support. Further, the process adds a limited amount of important new tokens. We detail this in more detail in the appendix Section 7.2.

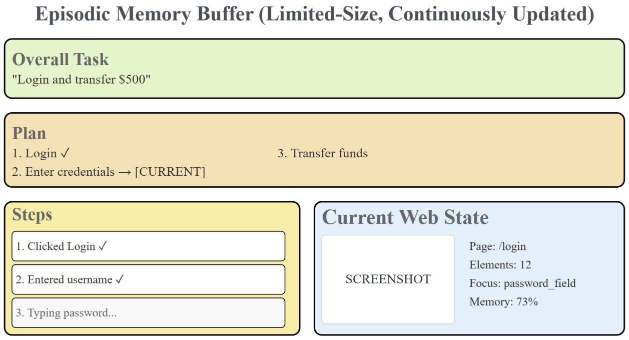


Figure 4. Episodic Memory Diagram dictates Task, Planning, Step Tracking, and Current Web State that form the Episodic Memory.

4.3. Multi-Agent Framework

WebSight implements a modified version the ReAct agent framework augmented with a planning agent and separated reasoning and action sub-steps visualized in Algorithm 1 [68]:

1. **Planning Agents.** These agents are responsible for constructing high-level action plans based on user instructions or task objectives. Leveraging classic chain-of-thought planning methodologies [63], our planning agents formulate task sequences that provide long-term context for the reasoning agents. These plans are dynamically updated based on feedback from verification agents.

2. **Reasoning Agents.** Situated below planning agents, reasoning agents determine precise next-step interactions necessary to progress the overall task. Utilizing transformer-based reasoning architectures [15, 47], these agents translate high-level plans into specific actionable steps such as ”click the login button” or ”fill in a text box.”

3. **Action Agent.** Central to WebSight is our custom trained WebSight-7B based agent, which interprets semantic instructions from the reasoning agents and translates them into visual interactions directly on screenshots.

4. **Verification Agents.** After the vision agent executes an action, verification agents rigorously evaluate the resulting changes in the webpage state. Utilizing visual reasoning, these agents verify task progress, update memory, and determine what the next step to be taken is–which was seen.

4.4. Episodic Memory

Integral to human-like interaction, in Figure 4 we incorporate a short-term episodic memory that records recent interactions and webpage states [57]. This memory allows our agents to iteratively refine action strategies, detect errors quickly, and prevent repetitive mistakes [5, 44]. The episodic memory is structured as a limited-size buffer storing recent interaction-action-outcome tuples, continuously updated and pruned based on task relevance.

Algorithm 1 WebSight Agent Loop

```
Initialize episodic memory $\mathcal{M} = \emptyset$ and task state $\mathcal{T}_0$
**Planning:** Create plan $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$
**while** $\mathcal{T}_t \neq \mathcal{T}_{complete}$ **do**
    **Reasoning:** Determine next action $a_t = f(\mathcal{P}, \mathcal{M}, \mathcal{T}_t)$
    **Action:** Execute WebSight visual action $\mathcal{V}(a_t)$
    **Verification:** Assess visual changes $\Delta\mathcal{V}_t$ and compute $\mathcal{T}_{t+1}$
    **Episodic Memory Update:** $\mathcal{M} = \mathcal{M} \cup \{(a_t, \Delta\mathcal{V}_t, \mathcal{T}_t, \mathcal{T}_{t+1})\}$
    **if** $\mathcal{T}_{t+1} \succ \mathcal{T}_t$ **then**
        $\mathcal{T}_t \leftarrow \mathcal{T}_{t+1}$
    **else**
        $\mathcal{P} \leftarrow \text{UpdatePlan}(\mathcal{P}, \mathcal{M}, \mathcal{T}_t)$
    **end if**
**end while**
**return** $\mathcal{T}_{complete}$
```

4.5. Alternative Approaches Considered

We initially explored purely monolithic architectures, such as directly using large multimodal transformer models [46, 18], to handle both perception and reasoning simultaneously. However, we found that decomposing perception and reasoning into specialized agents provided greater interpretability, modularity, and accuracy. Additionally, preliminary experiments indicated that fine-tuning domain-specific vision models resulted in better visual interaction accuracy than generalized multimodal models.

Our multi-agent orchestration framework, combined with targeted fine-tuning strategies, provides a robust and efficient solution, explicitly mirroring human-like browsing behaviors and ensuring practical applicability in diverse web navigation tasks.

5. Experiments

We conduct two experiments to evaluate WEBSIGHT-7B and the WEBSIGHT agent. First, the Showdown-Clicks benchmark, which isolates low-level click accuracy, is used to evaluate WEBSIGHT-7B compared to other leading VLMs. The WebVoyager benchmark, which measures end-to-end task completion in a realistic multi-page web environment, is used to evaluate the WEBSIGHT browser agent.

5.1. Showdown/Clicks

Showdown is a recent suite of offline and online tests for computer-use agents, released by General Agents in early 2025. The showdown-clicks track contains 5 679 human-collected left-click events on macOS, with a public dev subset of 557 examples. The benchmark is composed of a wide variety of challenging UI tasks requiring complex skills like understanding icons, processing semantic intent, and acting in situations where there are multiple viable tasks[59].

Published baselines vary widely. The SOTA, OpenAI's o3-based CUA reaches 64.27% top-1 accuracy, whereas a vanilla GPT-4o agent achieves only 5.2%, underscoring the difficulty of ambiguous UI contexts. Using the same test conditions, WEBSIGHT-7B attains **58.84%** accuracy, achieving higher accuracy than VLMs with almost 10x more parameters. Furthermore, WEBSIGHT-7B served on a single Nvidia H100 GPU is significantly faster than OpenAI's CUA for a slight drop in accuracy.

Model	Top-1 Accuracy (%)	Latency (ms)
WEBSIGHT-7B	**58.84**	2841
OpenAI CUA (o3-based)	64.27	6385
Molmo-72B-0924	54.76	6599
Claude 3.7 Sonnet	53.68	9656
UI-TARS-72B-SFT	54.40	1977
OmniParser V2 + GPT-4o	51.71	12642
Gemini 2.0 Flash	33.39	3069
Qwen2.5-VL-72B-Instruct	24.78	3790
GPT-4o	5.21	2500

Table 1. Top-1 Accuracy on the Showdown/Clicks Benchmark [59]

5.2. WebVoyager

The WebVoyager benchmark is a large-scale, real-world evaluation suite designed to measure the capabilities of autonomous web agents in handling interactive tasks across dynamic websites [22]. It spans hundreds of user intents on popular domains and tests abilities such as DOM reasoning, form filling, and multi-step navigation. Unlike synthetic tasks, WebVoyager emphasizes natural interaction and robustness in the open web, making it a strong benchmark for evaluating practical utility and generalization in vision-language agents.

Due to computational constraints and to ensure consistent evaluation, we use the filtered subset of 50 tasks introduced by Skyvern [55], which excludes outdated live tasks that are no longer possible. Table 2 shows performance across a variety of state-of-the-art agents evaluated on either the full or filtered WebVoyager set [56]. WEBSIGHT achieves a Success Rate of 68% on Skyvern's filtered WebVoyager Benchmark [55].

Agent	Success Rate (%)
WEBSIGHT	**68.0**
Browserable	90.4
Browser Use	89.1
Skyvern 2.0	85.8
Claude Computer Use	77.5
Agent-E	73.1
Runner H 0.1	67.0
OpenAI Operator (GPT-4o)	61.0
WebVoyager Agent	57.1

Table 2. WebVoyager Benchmark Success Rates [10, 56, 9, 19, 21, 3, 41]

From Table 2 we see that the SOTA Browserable achieves a success rate of 90.4%. Frontier labs like OpenAI and Anthropic achieve results of 61% and 77.5% respectively, highlighting the difficulty of developing a sophisticated browser agent even for labs with many resources.

5.3. Discussion & Analysis

5.3.1 WEBSIGHT-7B

We analyze screenshots of the failures in WEBSIGHT-7B and identify 3 key failure modes:

1. **Visual Grounding**: While WEBSIGHT-7B excels at identifying buttons and elements with text labels. Figure 5 shows such a use case, where WEBSIGHT-7B identifies the correct text but misses the interactivity of the icon.

2. **Extended Action Space**: The action space of WEBSIGHT-7B extends beyond clicks to include

actions like text-input, scrolling, etc. Sometimes, WEBSIGHT-7B chooses those actions when it does not find a direct answer to the task. Figure 6 shows such a failure mode, where the model chose to scroll to see more posts instea fo clicking the visible one.

3. **Icon Understanding**: WEBSIGHT-7B also struggles to distinguish between ambiguous icons that may have been out of the training distribution. Figure 7 shows a case where WEBSIGHT-7B misidentifies an icon and therefore makes an incorrect click.

Two of these failure modes tend to deal with interaction and understanding of icons, which highlights a key weakness in the choice of training data. Appendix 7.3 contains more examples of failures. It is important to note that these failure modes are all fairly uncommon in normal web browsing, and WEBSIGHT-7B excels at most clicking tasks.

5.3.2 WEBSIGHT Agent

In Figure 8, we observe that agents developed by startups exhibit the highest success rates. This performance likely reflects a concentrated focus on productization and rapid iteration. In contrast, research-oriented labs tend to pursue broader agendas, which may contribute to comparatively lower performance metrics. Notably, our agent outperforms those from both Frontier Labs and several younger labs, as shown in Table 2.

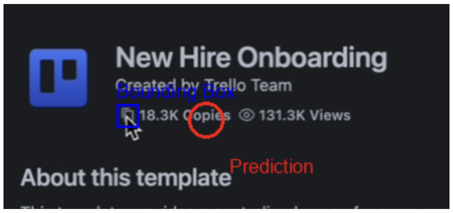


Figure 5. Task: Click on the button to copy

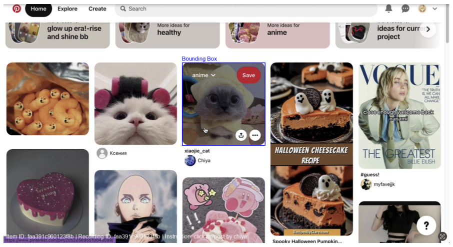


Figure 6. Task: Click on the post by Chiya

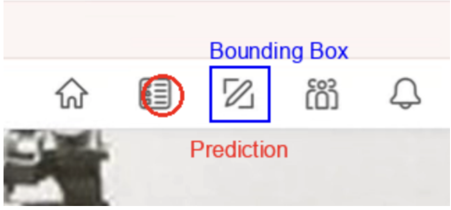


Figure 7. Click on the Answer icon

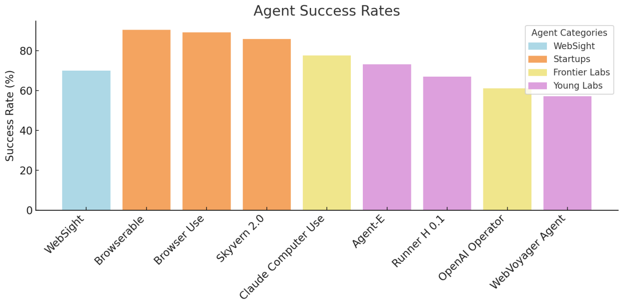


Figure 8. WEBSIGHT performance against other agents

Due to the small size of the Skyvern WebVoyager being only 50 tasks, we manually analyze the 16 tasks that fail to determine insights. We manually test each Planning agent's plan, Reasoning agent's proposed action, and Action agent's step to determine who was at fault, to glean interpretability on where WEBSIGHT could be improved. We know that if at any point WEBSIGHT produced an incorrect result, the Verification Agent has failed, since it did not catch the error.

Upon analysis of trajectories, we see that 15 of the 16 failures were due to the agent timing out, given our 10 minute restriction for WEBSIGHT to complete tasks. We saw that these 15 failures all had infinite loops. Table 3 highlights that the LLMs that power the Planning and Reasoning agent are WEBSIGHT's weakest links. Some potential ways infinite loops arose were:

- Planning agents sending WEBSIGHT into a place with no results, where WEBSIGHT would click a button, navigate backwards and repeat
- Reasoning agents repeatedly making the wrong decision, sending WEBSIGHT to a page, navigating backwards, and repeating
- Action agents repeatedly being unable to identify buttons or correct click locations as they are too small or fail to understand icons, and repeatedly do nothing

Component	Failure Count
Planning Agent	5
Reasoning Agent	7
Action Agent	3

Table 3. Failure Analysis of WEBSIGHT Agent by Component

Further, we understand that of the 35 tasks that WEBSIGHT provided an answer to, 34 were correct. This is a remarkable 97.14% accuracy, highlighting that WEBSIGHT only provided an answer to the task if it is confident about its answer.

6. Conclusion

6.1. Key Results

WEBSIGHT advances the capabilities of vision-based web agents by integrating our new specialized vision-language model (WEBSIGHT-7B) into a modular multi-agent system. Our approach demonstrates several key results:

- State-of-the-art performance for small models: WEBSIGHT-7B achieves a Top-1 Accuracy of **58.84%** on the challenging Showdown/Clicks benchmark, outperforming many larger-scale models while operating with lower latency.

- Competitive end-to-end task completion: On the WebVoyager benchmark, the full WEBSIGHT agent achieves a **68.0% success rate**, surpassing several industry and lab-developed agents including those from OpenAI and HCompany.

- High accuracy on completed tasks: Of the tasks WEBSIGHT attempted within the time limit, it achieved a **97.14% accuracy**, highlighting the precision of its decision-making pipeline.

- Interpretable modular diagnostics: Through a component-wise failure analysis, we identify Planning and Reasoning agents as primary sources of failure—primarily due to infinite loops—and offer targeted improvements, including upgraded language models and tighter agent coupling.

Together, these results demonstrate that vision-first agents are not only viable but highly competitive in practical web environments. WEBSIGHT offers a blueprint for interpretable, accurate, and efficient browser agents.

6.2. Directions for Future Work

6.2.1 WEBSIGHT-7B Model

To address the failure modes explained above, future work can be done in further fine-tuning the WEBSIGHT-7B model on a more diverse dataset of UI elements, including icons and scenes with multiple items with similar semantics. Further dataset augmentation with collected actions like scrolling can help with identification of the correct action from the action space as well.

As is often the case with transformers, we can expect an increase in performance from a scaling of parameters. Every model that performs similarly to WEBSIGHT-7B on the showdown/clicks dataset is more than 10 times larger, so a larger model WEBSIGHT-72B or similar could elicit more fine-grained reasoning capabilities.

6.2.2 WEBSIGHT Agent

To address infinite loops in WEBSIGHT and improve overall performance, we focus on enhancing the capabilities of its individual agents. As discussed in the previous section, improvements to the WEBSIGHT-7B model can directly benefit the Action Agent. Additionally, providing the Action Agent with richer context from the Reasoning Agent may enhance decision quality. For the Planning and Reasoning Agents, employing higher-quality language models could yield significant gains. Currently, due to cost constraints, we use GPT-4.1-mini; however, more capable models such as Claude-Sonnet-4 are likely to mitigate many of the observed limitations in planning and reasoning.

A promising future direction is enabling self-improvement within the agent system. Ideally, an agent should be able to detect when it is caught in an infinite loop—whether through repeating actions, failing to progress toward a goal, or exhausting relevant content—and respond by generating a new plan, shifting its strategy, or exploring an alternative page. This would require mechanisms for pattern recognition across trajectories and meta-level learning to adapt based on past failures. Recent work on self-improving and reflective agents provides a foundation for such capabilities [53, 28, 43].

6.2.3 WEBSIGHT Fused

With the emergence of reinforcement learning as a way to improve the utility of test-time compute for agentic flows, a final goal for WEBSIGHT would be a fully integrated agentic model that combines the clicking accuracy and action selection of WEBSIGHT-7B with the reasoning and planning capabilities of frontier LLMs. A fused model could be trained using offline RL based on a collected dataset of web browsing trajectories using GRPO, resulting in a single model that acts end to end in exactly the same way as a human. This is similar to what OpenAI has done with their “Operator”.

References

[1] AgentSea. Wave-ui-25k: A dataset for fine-tuning web interaction agents, 2024.

[2] J. R. Anderson. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, 2009.

[3] Anthropic. Claude 3.5 sonnet: Computer use evaluation results, 2024. Accessed: 2025-06-03.

[4] H. Bao et al. Beit: Bert pre-training of image transformers. *ICLR*, 2022.

[5] C. Blundell et al. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[6] R. Bommasani et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[7] S. Branavan et al. Reinforcement learning for mapping instructions to actions. In *ACL*, 2009.

[8] T. B. Brown et al. Language models are few-shot learners. *NeurIPS*, 2020.

[9] Browser Use Team. Sota technical report on webvoyager evaluation, 2024. Accessed: 2025-06-03.

[10] Browserable Team. Webvoyager benchmark: Browserable's state-of-the-art results, 2024. Accessed: 2025-06-03.

[11] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Extracting content structure for web pages based on visual representation. *Proceedings of the 5th Asia-Pacific Web Conference*, 2001.

[12] N. Carion et al. End-to-end object detection with transformers. *ECCV*, 2020.

[13] M. Caron et al. Emerging properties in self-supervised vision transformers. *ICCV*, 2021.

[14] L.-C. Chen et al. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL*, 2019.

[16] H. Dong et al. Towards automatic extraction of visually structured data from web pages. In *CIKM*, 2010.

[17] H. Donker and P. Markopoulos. Visual layout perception in website usability evaluation. *CHI Extended Abstracts*, 2002.

[18] A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[19] Emergence AI. Agent-e: A text-only agent outperforming multimodal agents, 2024. Accessed: 2025-06-03.

[20] D. Ganguli et al. Predictability and surprise in large generative models. *ICLR*, 2022.

[21] H Company. A research update: Runner h 0.1 performance on webvoyager, 2024. Accessed: 2025-06-03.

[22] H. He, W. Yao, K. Ma, W. Yu, Y. Dai, H. Zhang, Z. Lan, and D. Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

[23] J. He et al. Webvoyager: Building end-to-end agents for real-world web tasks with human instructions. *ICLR*, 2024.

[24] K. He et al. Mask r-cnn. In *ICCV*, 2017.

[25] M. Hessel et al. Rainbow: Combining improvements in deep reinforcement learning. *AAAI*, 2018.

[26] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *ACL*, 2018.

[27] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[28] Y. Huang, A. T. Chen, S. Bubeck, and Y. Zhang. Language models as tool makers: Teaching llms to invent and use tools for reasoning. *arXiv preprint arXiv:2305.17126*, 2023.

[29] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[30] A. Kirillov et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[31] J. Kumar et al. Extracting structure from web pages using vision-based page segmentation. *WWW*, 2007.

[32] J. E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.

[33] M. Lauer and M. Riedmiller. Learning for web navigation with reinforcement learning. *Adaptive Behavior*, 12(1):21–35, 2004.

[34] Y. LeCun. A path towards autonomous machine intelligence. *OpenReview preprint*, 2022.

[35] X. Li et al. Agentbench: Evaluating general-purpose agents across thousands of tasks. *arXiv preprint arXiv:2402.00047*, 2024.

[36] H. Liu et al. Webagent: Instruction-following agents for web navigation with large language models. *arXiv preprint arXiv:2305.15368*, 2023.

[37] H. Liu et al. Webui: A dataset for benchmarking ui understanding models. *arXiv preprint arXiv:2305.16333*, 2023.

[38] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[39] R. Nakano et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[40] OpenAI. Gpt-4 technical report, 2023.

[41] OpenAI. Openai operator with gpt-4o for web interaction tasks, 2024. Accessed: 2025-06-03.

[42] L. Ouyang et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[43] Z. Pan, R. E. Liu, D. Scherlis, et al. Automated debugging of language model programs. *arXiv preprint arXiv:2305.13266*, 2023.

[44] A. Pritzel et al. Neural episodic control. *ICML*, 2017.

[45] Y. Qin, Y. Ye, J. Fang, H. Wang, S. Liang, S. Tian, J. Zhang, J. Li, Y. Li, S. Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[46] A. Radford et al. Learning transferable visual models from natural language supervision. *ICML*, 2021.

[47] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

[48] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[49] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[50] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[51] J. Schulman et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[52] W. Shi et al. Learning generalizable policies for interactive web navigation. *ICML*, 2022.

[53] N. Shinn, J. Liu, Y. Du, and P. Abbeel. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

[54] D. Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[55] Skyvern AI. Skyvern filtered webvoyager dataset, 2024.

[56] Skyvern Team. Skyvern 2.0: State-of-the-art web navigation with 85.8% on webvoyager eval, 2024. Accessed: 2025-06-03.

[57] T. R. Sumers, S. Yao, K. Narasimhan, and T. L. Griffiths. Cognitive architectures for language agents, 2024.

[58] J. Sun et al. Recovering missing accessibility metadata for robust web agents. *EMNLP*, 2022.

[59] G. A. Team. The showdown computer control evaluation suite, 2025.

[60] G. Tesauro. Online resource allocation using decompositional reinforcement learning. *AAAI*, 2005.

[61] O. Vinyals et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019.

[62] W. Wang et al. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*, 2022.

[63] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2023.

[64] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2009.

[65] C. Xu et al. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

[66] K. Xu et al. Focal: Fine-grained open-world clickable elements detection for web agents. *ACL*, 2022.

[67] W. Yang et al. Sphynx: Augmenting language models with visual context for web navigation. *arXiv preprint arXiv:2310.00123*, 2023.

[68] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models, 2023.

[69] H. Zhang et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

[70] Y. Zhang et al. Screen2words: Screen content description with multimodal transformers. *ACL*, 2021.

[71] Z. Zhang et al. Autoagents: Unleashing the power of llms in autonomous agents. *arXiv preprint arXiv:2309.00062*, 2023.

[72] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, Z. Feng, and Y. Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[73] X. Zhou et al. Webarena: A realistic web environment for building autonomous agents. *NeurIPS*, 2023.

Appendix

7.1 Prompting

We share below the prompts that WEBSIGHT uses to achieve its results. Through fine prompt engineering, we iterated to the following versions that we found to work best.

7.1.1 Planning Agent

The planning agent is only used at the beginning of WEBSIGHT's operations to determine the plan it will execute. This agent's system prompt is as follows:

```
You are a web automation
planner.  Your job is to
break down web tasks into
simple steps that a browser
can follow.

Key points:
- Break tasks into basic steps
- Keep steps clear and direct
- Account for page loading

Keep your plans simple and
focused on the main goal.
```

The system prompt establishes context and instructions for the planner. The agent's user prompt for task specific instructions is as follows:

```
 Create a detailed plan for
a browsing agent to complete
the following task.  Break it
down into specific, actionable
steps.

Task:  {task}

For each step, include:
1.  The specific action to
take, referring to specific
elements on the page

Format your response as a
numbered list of steps.  Be
specific about URLs, element
types, and expected outcomes.
Respond in this format:
<step> STEP GOES HERE </step>
<step> STEP GOES HERE </step>
...
```

7.1.2 Reasoning Agent

The reasoning agent determines next steps and executes fine details between interactions of the browsers. This agent's system prompt is as follows:

```
You are a web automation
agent using ReAct framework.
Your goal:  complete tasks
efficiently and handle
failures gracefully.

CRITICAL RULES:
- Analyze screenshot carefully
before each action
- Use specific selectors and
exact text
- Wait for dynamic content
when needed
- Try alternatives if primary
approach fails
- When you're done, return
"FINISHED" and then your final
response
- Don't scroll unless
absolutely necessary

RESPONSE FORMAT:
<reasoning>Brief analysis
of current state and why
this action advances the
goal</reasoning>
<action>Specific action
(e.g., "Click the blue
'Login' button", "Type
'user@email.com' in email
field", "Navigate to
https://site.com")</action>

IF YOU ARE FINISHED:
<reasoning>Your reasoning
here</reasoning>
<action>FINISHED + your final
response</action>

Handle common patterns:
loading states, forms, modals,
authentication.  Each Action
should be a single step and be
atomic (e.g.  don't click on a
button and then type in a text
field).
```

The system prompt establishes a lot of the key rules and format and instructions for the reasoner. The agent's user prompt for state specific instructions is as follows:

```
TASK: {plan}
HISTORY: {history}
SCREENSHOT: [Current page
state]

ANALYSIS REQUIRED:
1.  What’s on screen right
now?
2.  What’s the next logical
step toward the goal?
3.  What could go wrong and
how to handle it?

<reasoning>
Current state:  [What you see]
Next step:  [Why this action
moves toward goal]
Risk mitigation:  [Backup plan
if this fails]
</reasoning>

<action>[Precise action
instruction]</action>
BE CONCISE. BE ACCURATE.
HANDLE EDGE CASES.
```

7.1.3 Vision WEBSIGHT-7B Agent

The vision agent must deliver a simple actionable task to the browser wrapper to complete. Therefore, we ensure it is in the action space through the prompt. As this is not a chat model, there is no system prompt, and only one prompt that is displayed below:

~~~
 You are a GUI agent.  You are
given a task and your action
history, with screenshots.
You need to perform the next
action to complete the task.

Output Format
```
Thought:  ...
Action:  ...
```

Action Space

click(point=’<point>x1
y1</point>’)
left_double(point=’<point>x1
y1</point>’)
right_single(point=’<point>x1
y1</point>’)
drag(start_point=’<point>x1
y1</point>’,
end_point=’<point>x2
y2</point>’)
hotkey(key=’ctrl c’)  Split
keys with a space and use
lowercase.  Also, do not use
more than 3 keys in one hotkey
action.
type(content=’xxx’)  Use
escape characters \\’, \\\",
and \\n in content part to
ensure we can parse the
content in normal python
string format.  If you want
to submit your input, use \\n
at the end of content.
scroll(point=’<point>x1
y1</point>’, direction=’down
or up or right or left’)
Show more information on the
`direction` side.
wait() Sleep for 5s and take
a screenshot to check for any
changes.
finished(content=’xxx’)  Use
escape characters \\’, \\", and
\\n in content part to ensure
we can parse the content in
normal python string format.

Note
- Use {language} in `Thought`
part.
- Write a small plan and
finally summarize your next
action (with its target
element) in one sentence in
`Thought` part.

DO NOT REPEAT ACTIONS. If an
action is not successful, try
something else.  If you’ve
already clicked on something,
don’t click on it again,
either try another action or
do something else like typing.

If you are stuck or a website
is blocked, use the finished
action to stop the agent with
the argument "STUCK"

User Instruction
{instruction}
~~~

7.2 Fine-tuning

The finetuning process completes a majority of the learning very quickly. There are a multitude of reasons for this. First, websites are a large part of the UI-TARS training data, and are not too far different from mobile and desktop application data. Therefore, there is not too much learning needed. Second, UI-TARS also supports English as it's outputs are a mix of English and Chinese, so fine tuning the language for interpretability should be quick.

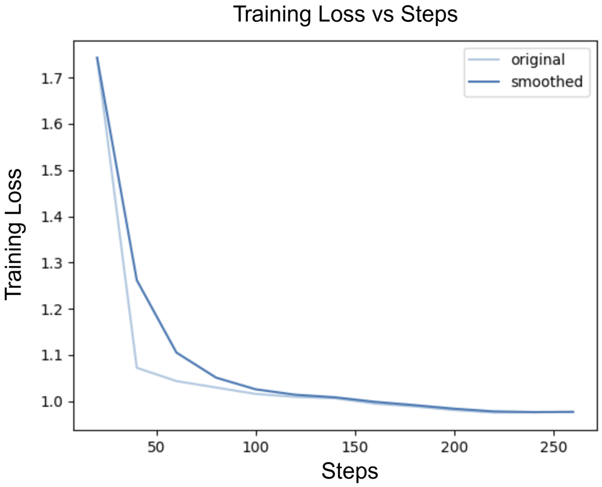


Figure 9. UI-TARS LoRA Fine Tuning process Training Loss vs Steps Graph

During the LoRA fine-tuning process, several special tokens were introduced to extend the model's capabilities for structured inputs, tool usage, and multimodal interactions. These include `<|im_start|>` (151644) and `<|im_end|>` (151645) for denoting the start and end of instruction-style prompts, as well as `<|object_ref_start|>` (151646) and `<|object_ref_end|>` (151647) for marking object references in input sequences. Visual and spatial data are supported through tokens such as `<|box_start|>` (151648), `<|box_end|>` (151649), `<|quad_start|>` (151650), and `<|quad_end|>` (151651), along with `<|vision_start|>` (151652), `<|vision_end|>` (151653), and `<|vision_pad|>` (151654) for vision input boundaries and padding.

To handle image and video input formats, the tokens `<|image_pad|>` (151655) and `<|video_pad|>` (151656) were added. For managing tool interactions, the tokens `<tool_call>` (151657) and `</tool_call>` (151658) were introduced. Code-related modifications are supported via `<|fim_prefix|>` (151659), `<|fim_middle|>` (151660), `<|fim_suffix|>` (151661), and `<|fim_pad|>` (151662), enabling more flexible handling of function-in-the-middle completions. Finally, `<|repo_name|>` (151663) and `<|file_sep|>` (151664) were included to represent code repository metadata and file separation, respectively. The standard `<|endoftext|>` token (151643) remains as a sentinel for sequence termination. These additions enable the fine-tuned model to operate effectively in a variety of structured, interactive, and multimodal scenarios.

7.3 WEBSIGHT-7B Failure Modes on Showdown/Clicks

Predicted click location highlighted in and ground truth bounding box highlighted in

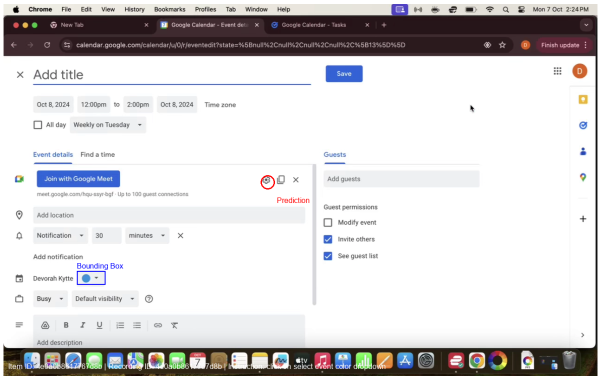


Figure 10. Click on select event color dropdown

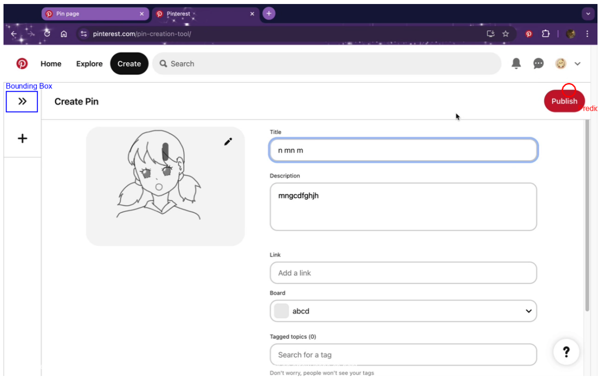


Figure 11. Click on arrow icons on right

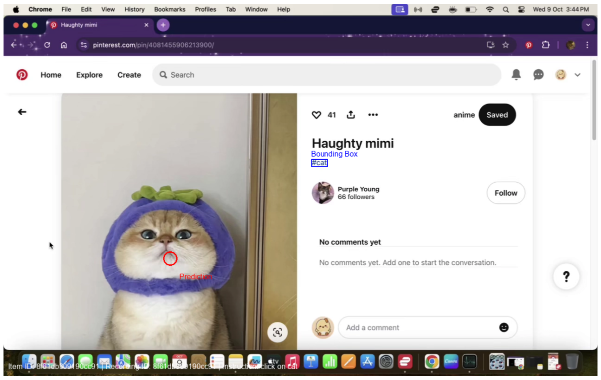
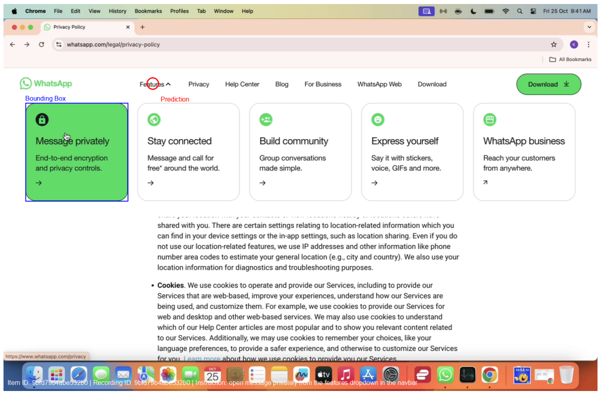


Figure 12. Click on cat

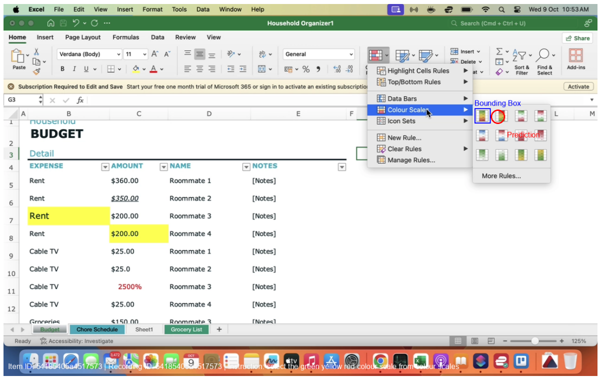

WhatsApp
Features
Privacy
Help Center
Blog
For Business
WhatsApp Web
Download
Prediction
Bounding Box
Message privately
End-to-end encryption and privacy controls.
Stay connected
Message and call for free* around the world.
Build community
Group conversations made simple.
Express yourself
Say it with stickers, voice, GIFs and more.
WhatsApp business
Reach your customers from anywhere.

shared with you. There are certain settings relating to location-related information which you can find in your device settings or the in-app settings, such as location sharing. Even if you do not use our location-related features, we use IP addresses and other information like phone number area codes to estimate your general location (e.g., city and country). We also use your location information for diagnostics and troubleshooting purposes.

- **Cookies**. We use cookies to operate and provide our Services, including to provide our Services that are web-based, improve your experiences, understand how our Services are being used, and customize them. For example, we use cookies to provide our Services for web and desktop and other web-based services. We may also use cookies to understand which of our Help Center articles are most popular and to show you relevant content related to our Services. Additionally, we may use cookies to remember your choices, like your language preferences, to provide a safer experience, and otherwise to customize our Services for you. Learn more about how we use cookies to provide you our Services.

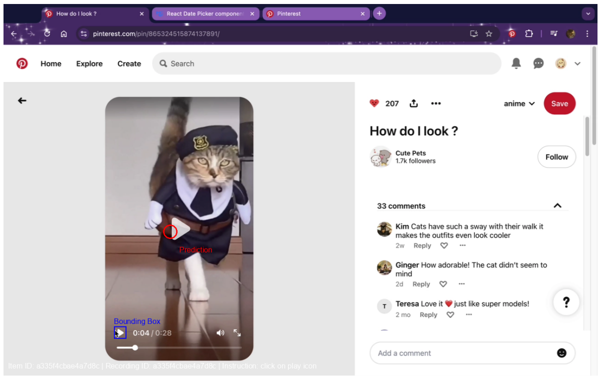
Instruction: open message privately from the features dropdown in the navbar

Figure 13. Open message privately from the features dropdown in the navbar



EXPENSE	AMOUNT	NAME	NOTES
Rent	$360.00	Roommate 1	[Notes]
Rent	$350.00	Roommate 2	[Notes]
Rent	$200.00	Roommate 3	[Notes]
Rent	$200.00	Roommate 4	[Notes]
Cable TV	$25.00	Roommate 1	[Notes]
Cable TV	$25.0	Roommate 2	[Notes]
Cable TV	2500%	Roommate 3	[Notes]
Cable TV	$25.00	Roommate 4	[Notes]

Figure 14. Select the green yellow red colour scale from colour scales



Kim Cats have such a sway with their walk it makes the outfits even look cooler

Ginger How adorable! The cat didn't seem to mind

Teresa Love it ❤️ just like super models!

Figure 15. Click on play icon