Inference-Time Scaling for GUI Agents with Process Reward Models and Internal World Models

Bella Wang yuw170@ucsd.edu
Rita Yujia Wu yuw172@ucsd.edu
Shuchang Liu shl153@ucsd.edu
Ziyu Huang zih029@ucsd.edu
Kun Zhou kuzhou@ucsd.edu
Zhiting Hu zhh019@ucsd.edu

Abstract

This project explores inference-time scaling for GUI agents—agents that operate directly on complex graphical user interfaces such as OSWorld (Xie et al. 2024) and AndroidWorld (Rawles et al. 2024). We investigate how to improve a frozen vision–language GUI policy (Qwen3-VL-4B) by adding smarter test-time reasoning rather than simply training larger models.

On the external side (Part A), we build a Process Reward Model (PRM) (Lightman et al. 2024) that scores each step of a GUI trajectory given the task, screenshots, and actions. Using ZeroGUI (Yang et al. 2025) and OSWorld, we first generate tasks and collect trajectories, then label each step with a scalar reward using a multimodal LLM. These labeled trajectories are used to fine-tune a PRM that can evaluate partial solutions and serve as an external signal for reranking and filtering candidate trajectories. We also integrate AndroidWorld into the ZeroGUI evaluation pipeline to support online evaluation of PRM-guided agents across multiple environments.

On the internal side, we construct a world model that builds a concrete understanding of the environment and current website before taking any action. Unlike reactive approaches that respond to observations without foresight, our world model proactively retrieves and analyzes similar historical trajectories both successful and failed—to form a mental model of what actions typically lead to success versus failure in comparable situations.

By combining external PRM scoring with internal world-model rollouts, we hypothesize that the agent will achieve higher success rates, need fewer environment interactions, and generalize better to unseen tasks than a baseline GUI agent without inference-time scaling.

Code: https://github.com/RitaYujiaWu/DSC180-A08-GUI-Project

1 Introduction

Graphical User Interface (GUI) agents are an emerging class of AI systems that operate directly on real applications—opening apps, clicking buttons, typing into forms, and navigating complex workflows on operating systems, browsers, and mobile devices. Recent progress in vision–language models (VLMs) such as Qwen3-VL (Bai et al. 2025) has made it feasible to treat GUI control as a sequence modeling problem: the model observes screenshots and instructions, then outputs actions one step at a time. However, simply scaling backbone models is expensive and provides limited control over how the agent reasons and explores during deployment.

Inference-time scaling is typically understood as improving a model's performance by increasing the amount of computation it can expend at test time, without changing its parameter count. Classic examples include sampling more candidate solutions and reranking them, running a deeper tree search (Yao et al. 2023), or unrolling longer chains of thought (Wei et al. 2023) before committing to an answer. In all of these cases, the central object being scaled is the inference trace—the sequence of intermediate states and reasoning steps the model traverses—rather than the size of the network itself.

In this project, we adopt a compatible but slightly broader view of inference-time scaling tailored to GUI agents. Rather than focusing on a single mechanism, we study how different components can help the agent use and benefit from longer trajectories at inference time. In Part A, a PRM provides dense, step-wise supervision within each trajectory, rewarding actions that make genuine progress toward the goal and penalizing those that are redundant or harmful; this training signal encourages the policy to organize its behavior over many steps, rather than optimizing only for short, myopic sequences that happen to reach terminal success. In Part B, an internal world model is used to structure the agent's multi-step behavior more explicitly, by learning contrastive representations of success and failure patterns from historical trajectories, then dynamically retrieving state-relevant examples at each step to guide action selection and avoid previously observed failure modes. Taken together, these components reshape the policy and its internal state so that longer rollouts become meaningful, structured "reasoning traces" instead of random wandering, allowing the agent to genuinely benefit from a larger inference-time budget without increasing model size.

2 Methods

2.1 Part A – Process Reward Model and PRM-Guided Agent Training

2.1.1 Environments and Base Agent

We build on the OSWorld (Xie et al. 2024) and ZeroGUI (Yang et al. 2025) frameworks as our primary testbed:

- OSWorld provides realistic multi-step desktop GUI tasks with an environment API, action space, and screenshot capture.
- ZeroGUI provides utilities for task generation, metadata management, and evaluation.
- AndroidWorld is later used for evaluating mobile-style tasks using a similar interface.

For the base policy, we use Qwen/Qwen3-VL-4B-Instruct as a vision–language GUI agent, hosted locally via vLLM (Kwon et al. 2023). The agent observes task instructions and screenshots and outputs actions (e.g., click, type, open app) until success, failure, or a step limit.

2.1.2 Task Generation and Trajectory Collection

We first generate a pool of diverse tasks and trajectories in OSWorld:

1. Step-0 Screenshot Collection
   - Run OSWorld with max_steps = 0 to collect initial screenshots for each meta-task.
   - These screenshots provide visual context for task generation (e.g., what apps are open, where buttons are).
2. Task Generation via ZeroGUI
   - Use a ZeroGUI-style task generator to create natural-language task descriptions conditioned on the step-0 screenshots.
   - Examples: "Open Chrome and search for 'DSC180 syllabus'," "Rename this file to 'report_final.pdf'," etc.
   - Consolidate generated tasks into a meta file (e.g., test_generated.json) used by the OSWorld runner.
3. Trajectory Collection with Qwen3-VL-4B
4. Host Qwen3-VL-4B with vLLM on one or more GPUs.
5. Run OSWorld on the generated task set with a non-zero max_steps (e.g., 25) and parallel environments.
6. For each episode, record: 1. The task instruction; 2. A sequence of (observation screenshot, action, environment feedback); 3. The final success/failure outcome.

2.1.3 Step-Wise Reward Annotation with a Multimodal LLM

End-of-episode success labels are too coarse to train a detailed process reward model, so we use a multimodal LLM (e.g., GPT-5-mini) to annotate each step:

- Input to the annotator (per step):
  - Task description.
  - The "before" screenshot at step t.
  - The action taken at step t.
  - Optionally, the "after" screenshot at step t+1 or short environment feedback.
- Output: A scalar reward (in [0, 1]), where:
  - High rewards indicate progress toward the goal.

  - Low or negative rewards indicate mistakes, regressions, or no progress.
  - A short rationale explaining why the reward was assigned.

We typically:

1. Run a dry-run mode to verify file paths and formatting.
2. Run the full annotation script with multiple workers to parallelize LLM calls.
3. Store the resulting annotations as a JSONL file, where each trajectory contains a list of step-wise rewards and rationales aligned with the recorded steps.

2.1.4 Fine-Tuning the Process Reward Model (PRM)

We have completed PRM fine-tuning under this setup, using Qwen3VL-4B as the PRM backbone and LLaMA-Factory as our code base. The training data consisted of OSWorld trajectories that were generated and then format-corrected to ensure consistent annotations, including step-level action descriptions and associated observations. This fine-tuned PRM is used to produce clipped, scaled step rewards (and short rationales when enabled) from a fixed-length history window, providing a reliable dense reward signal for downstream agent training and evaluation.

2.1.5 Implementation Details and Data Standardization

In practice, PRM training depends heavily on consistent trajectory formatting and clean step alignment across environments. We therefore added a lightweight standardization layer before reward annotation and PRM fine-tuning:

- **Unified trajectory schema.** We convert raw OSWorld logs into a unified JSONL format containing: (i) task instruction; (ii) per-step screenshot path(s); (iii) normalized action (type + arguments); (iv) optional environment feedback; and (v) episode-level success/failure metadata.
- **Action normalization and filtering.** A single model response can be parsed into multiple low-level UI actions (e.g., mouse_move + scroll, or type + enter). Many micro-actions are semantically redundant for PRM supervision and can introduce noise. We apply a rule-based filter to remove or merge such actions, then re-index steps to obtain a strictly increasing, semantically meaningful step sequence.
- **Schema validation.** Before running annotation or training, we run automated checks for missing keys, malformed action fields, broken screenshot paths, and step-index inconsistencies. This prevents silent training failures and makes reruns reproducible.

2.1.6 Agent Training Reinforcement Learning with PRM

We built an end-to-end reinforcement learning pipeline that trains a GUI agent in OSWorld using PPO, with dense step-level supervision provided by a Process Reward Model (PRM). The environment is executed in a headless setup and exposes screenshot observations together with a GUI action interface. At each interaction step, the agent proposes an action

using a vision-language policy, executes it in the environment, and then queries the PRM to score the step based on a short history window of recent observations and actions. The PRM outputs a scalar reward (clipped and scaled for stability), optionally accompanied by a brief rationale for debugging. These PRM-derived rewards are then used to compute advantages and perform PPO updates, while logging and periodic checkpointing track training progress and support iterative experimentation.

2.1.7 Future Steps

We will continue improving agent performance by scaling up training and tuning key PPO (or other policies) and PRM-related hyperparameters to stabilize learning and increase sample efficiency. In parallel, we will strengthen the trajectory pipeline by improving data formatting consistency and expanding coverage to more diverse behaviors and edge cases, enabling the PRM reward signal to generalize better.

2.2 Part B – Internal World Model for GUI Agents

2.2.1 Environments and Base Agent

We build on the CoMEM (Continuous Memory) framework (Wu et al. 2025) as our foundation:

- CoMEM provides a GUI agent infrastructure with experience memory, FAISS-indexed trajectory retrieval (Johnson, Douze and Jégou 2019), and CLIP-based multimodal embeddings (Radford et al. 2021).
- The framework supports multiple benchmarks including MMInA (Zhang et al. 2024), WebVoyager (He et al. 2024), and Mind2Web (**?**) for web-based GUI tasks.
- Browser automation is handled via Playwright with a standardized action space (click, type, scroll, etc.).

For the base policy, we use Qwen/Qwen2.5-VL-7B-Instruct as the vision–language GUI agent, hosted locally via vLLM (Kwon et al. 2023). The agent observes task instructions and screenshots, receives guidance from the world model, and outputs actions until the task is completed or the step limit is reached.

2.2.2 Trajectory Storage with Contrastive Indexing

The original CoMEM retrieves only successful trajectories as positive examples. We extend this to support contrastive retrieval:

1. Trajectory Organization
   - Trajectories are organized by outcome:
     - `training_data/{dataset}/{domain}/success/*.jsonl`
     - `.../failure/*.jsonl`

- Each trajectory contains: task description, action sequence, screenshots at each step, and final success/failure label.

2. Dual FAISS Indexing
   - We maintain **separate FAISS indices** for success and failure trajectories, enabling independent top-k retrieval from each category.
   - Embeddings are computed using CLIP (text-only or multimodal with first screenshot) and L2-normalized for cosine similarity.
   - Queries use a prefixed format (`"{dataset}_{domain}: {task}"`) for domain-aware matching, consistent with the original memory system.
3. Contrastive Pair Retrieval
   - Given a task and screenshot, we retrieve the top-k most similar success trajectories and the top-k most similar failure trajectories.
   - This enables the world model to analyze what distinguishes successful attempts from failed ones on similar tasks.

2.2.3 Contrastive Analysis and Guidance Generation

We implement an LLM-based analyzer to extract actionable insights from contrastive pairs:

- Input to the analyzer:
  - Task description.
  - Retrieved success trajectories (actions, outcomes).
  - Retrieved failure trajectories (actions, failure points).
- Output: A set of contrastive insights, including:
  - **Success patterns**: Common action sequences that lead to task completion (e.g., "use search bar directly instead of browsing categories").
  - **Failure patterns**: Common mistakes to avoid (e.g., "adding to cart without verifying product variant").
  - **Divergence points**: Critical decision points where success and failure trajectories differ.

These insights are summarized into a compact text guidance ($\sim$200 tokens) and injected into the agent's prompt at each step. During development, we further refined the contrastive pipeline to improve robustness in ambiguous cases. In particular, we evaluated different evidence aggregation strategies (e.g., maximum similarity vs. top-k average similarity) and introduced confidence-aware filtering to prevent the model from overreacting to weak or conflicting retrieval signals. These refinements help stabilize the guidance when retrieved neighbors are visually similar but semantically misaligned with the current task.

2.2.4 Dynamic Step Guidance with State-Aware Retrieval

Unlike static retrieval that queries once at task start, we implement **dynamic retrieval** at each step:

1. Initial Guidance (Step 0)

- Retrieve contrastive pairs using task description + initial screenshot.
- Generate initial guidance summarizing success patterns and common pitfalls.
- Inject into the agent's system prompt.

2. Step Guidance (Steps 1+)
   - **Re-retrieve** contrastive pairs using the **current screenshot**, not the initial one.
   - This finds trajectories relevant to the agent's current page state, not just the starting state.
   - Generate step-specific guidance based on freshly retrieved examples.
   - Inject into the user message alongside the current observation.

This dynamic approach ensures guidance remains relevant as the agent navigates through different pages and states.

2.2.5 Current Status and Future Steps

We have implemented the core world model infrastructure with text-based contrastive retrieval, dynamic guidance generation, and robustness refinements for ambiguous retrieval cases. The system currently:

- Maintains separate FAISS indices for success and failure trajectories.
- Performs dynamic retrieval at each step using the current visual state.
- Generates contrastive text summaries injected into agent prompts.
- Applies confidence-aware filtering to mitigate weak or ambiguous retrieval signals.

We extend the world model with **latent contrastive learning** as an ongoing direction:

1. **Contrastive Embedding Space**: Train a shared encoder that maps both successful and failed trajectories into a latent space where success/failure patterns are explicitly separated, using contrastive losses (e.g., InfoNCE (**?**)) to push apart embeddings of success and failure trajectories for the same task.
2. **Latent State Prediction**: Use the learned embeddings to predict whether a proposed action sequence is likely to succeed or fail, enabling the agent to evaluate candidate actions before execution.
3. **Integration with Continuous Memory**: Combine the contrastive world model with CoMEM's Q-Former architecture (Li et al. 2023) to encode both positive and negative examples into fixed-length embeddings, reducing context overhead while preserving contrastive information.

This helps unify the text-based contrastive guidance with learned latent representations, enabling more nuanced action selection based on both explicit patterns and implicit embedding similarities. We note that retrieval based primarily on visual similarity can occasionally surface trajectories that are not fully aligned with the underlying task intent; improving task-aware query representations remains an important direction for future work.

3 Experiment

3.1 Part A – Process Reward Model and PRM-Guided Agent Training

3.1.1 Task domains and generated tasks.

Following the OSWorld-Verified setup, our experiments cover ten concrete application domains:

- LibreOffice Calc (spreadsheets)
- LibreOffice Writer (text documents)
- LibreOffice Impress (presentations)
- Google Chrome (web browsing)
- VLC (media playback)
- Thunderbird (email and messaging)
- GIMP (image editing)
- VS Code (software development)
- OS-level file management (file explorer, file I/O tasks)
- OS/system tasks and multi-app workflows (e.g., opening apps, switching windows, cross-app coordination)

Using the ZeroGUI + OSWorld task-generation pipeline, we instantiated tasks for each of these domains. We successfully generated approximately 20 candidate tasks per domain (around 200 in total) and manually filtered them for executability and diversity, resulting in a curated set of 184 tasks that we use as our main PRM fine-tuning dataset.

3.1.2 Initial OSWorld small 39-task benchmark and trajectory collection.

As a first diagnostic benchmark for both the base agent and the reward-labeling pipeline, we selected a subset of 39 tasks that spans all of the above domains and includes a mixture of short and long-horizon behaviors. On this 39-task subset:

- We ran the Qwen3-VL-4B agent multiple times per task under two environment configurations:
  - sleep_after_execution = 0, max_steps = 15
  - sleep_after_execution = 0.5, max_steps = 25
- These runs produced multiple trajectories per task, allowing us to compare how different step budgets and sleep settings affect trajectory length, success rate, and qualitative behavior.

3.1.3 Reward labeling and PRM supervision statistics.

We then applied our LLM-based reward-labeling pipeline to all trajectories from the 39 tasks:

Table 1: Fine-tuning hyperparameters used in LLaMA-Factory.

Hyperparameter	**Value**
Learning rate	1×10^{-5}
Train batch size	1
Eval batch size	8
Gradient accumulation steps	16
Total train batch size (effective)	16
Epochs	5
Optimizer	`adamw_torch` (β_1=0.9, β_2=0.999, ϵ=10^{-8})
LR scheduler	Linear
Seed	42
Mixed precision	Native AMP

- For each step in each trajectory, we queried a multimodal LLM multiple times, using both GPT-4o-mini and GPT-5-mini as annotators.
- Each query returned a scalar step-wise reward along with a short natural-language rationale.

For each step, we later aggregated the rewards from different runs/models over the 39-task subset and computed descriptive statistics (mean, median, standard deviation) for analysis.

3.1.4 Trajectory logging and preprocessing.

To make the collected rollouts usable for step-wise PRM supervision, we standardized the raw OSWorld logs into a clean training-ready trajectory format:

- For each rollout, we logged the full interaction trace, including: task instruction, per-step screenshot path(s), the model response, the parsed low-level action(s) executed in the environment, and optional environment feedback.
- We applied an action normalization and filtering step to remove semantically redundant micro-actions (e.g., cursor movements that do not change the UI state) and to merge multi-action bursts when they represent a single logical decision.
- After filtering, we re-indexed the remaining steps to obtain a strictly increasing step sequence and ran schema validation checks (missing keys, broken screenshot paths, malformed action fields) to prevent format-related failures during reward labeling and PRM fine-tuning.

3.1.5 PRM Fine-tuning.

We then fine-tuned the PRM using LLaMA-Factory as the training framework and the 184 generated tasks (after format correction) as the training dataset. The PRM was trained with the hyperparameters listed below in Table 1.

3.1.6 Agent Reinforcement Learning with PRM.

Agent training was then conducted in the OSWorld environment using a headless Docker and Ubuntu setup with a PyAutoGUI action space and screenshot observations at 1920×1080 resolution. We trained a Qwen3-VL-4B-Instruct policy on GPUs, generating up to 256 tokens per action with sampling parameters temperature=0.7 and top-p=0.9. Step rewards were provided by locally serving the PRM we trained (as stated in 3.1.5) using vllm, which evaluates each decision using a 3-step history window (window_steps=3) and returns both a scalar reward and a short rationale (capped at 300 characters). PRM rewards were scaled by 1.0 and clipped to [0,1], with a 30s timeout and up to 2 retries per query. We currently optimized the policy with PPO using γ=0.99, GAE λ=0.95, learning rate 1e−5, clip ratio 0.2, value loss coefficient 0.5, entropy coefficient 0.01, and gradient clipping 1.0. Training artifacts (policy updates/PRM rewards) were saved, with logging and checkpointing performed at designated steps/episodes.

3.2 Part B – Internal World Model for GUI Agents

3.2.1 Task domains and trajectory datasets.

Our experiments focus on web-based GUI navigation tasks using the WebVoyager benchmark (He et al. 2024):

- **WebVoyager**: Real-world web tasks spanning 15+ domains including:
  - E-commerce: Amazon, Allrecipes
  - Search and information: Google Search, Google Maps, Google Flights, ArXiv, Wolfram Alpha
  - Developer tools: GitHub, Huggingface
  - Media and news: ESPN, BBC News
  - Travel and booking: Booking, Apple

For trajectory collection, we use Qwen2.5-VL-7B as the base agent with a Playwright-based browser environment (viewport 1280×720). Each trajectory records: task instruction, sequence of (screenshot, action, observation) tuples, and final success/failure outcome determined by ground-truth evaluation.

3.2.2 Contrastive trajectory corpus and FAISS indexing.

We constructed a contrastive trajectory corpus by collecting agent rollouts and labeling outcomes:

- For each domain, we organized trajectories into `success/` and `failure/` directories based on task completion.
- We built separate FAISS indices for success and failure trajectories using CLIP embeddings:

  - **Text-only index**: Embeddings from prefixed task descriptions (`"webvoyager_{domain}: {task}"`).
  - **Multimodal index**: Concatenated text and first-screenshot embeddings for richer similarity matching.
- Index statistics (example for Amazon domain):
  - Success trajectories indexed: ∼150
  - Failure trajectories indexed: ∼80
  - Average trajectory length: 8.3 steps (success), 12.1 steps (failure, often hitting step limit)

3.2.3 World model retrieval and guidance injection.

We evaluated the world model's retrieval quality and guidance generation:

- **Retrieval configuration**: Top-k = 3 for both success and failure trajectories at each step.
- **Dynamic retrieval analysis**: We compared retrieval results between initial screenshot (step 0) and subsequent screenshots (steps 1+):
  - On average, 40% of retrieved trajectories changed between step 0 and step 5, confirming that dynamic retrieval surfaces different, state-relevant examples as the agent navigates.
  - Tasks with multi-page navigation showed higher retrieval turnover compared to single-page tasks.
- **Guidance generation**: For each retrieval, the world model produces:
  - Initial guidance (∼200 tokens): Success patterns, failure patterns, and key divergence points.
  - Step guidance (∼150 tokens): State-specific recommendations based on current-screenshot retrieval.

3.2.4 Preliminary ablation: retrieval strategy comparison.

We conducted a preliminary comparison of retrieval strategies on a subset of WebVoyager tasks:

- **No world model** (baseline): Agent receives only task instruction and screenshots.
- **Success-only retrieval**: Retrieve top-3 successful trajectories (original CoMEM approach).
- **Contrastive retrieval (static)**: Retrieve success + failure pairs once at step 0, use cached insights throughout.
- **Contrastive retrieval (dynamic)**: Re-retrieve success + failure pairs at each step using current screenshot.

Full quantitative results and success rate comparisons will be reported after completing evaluation runs across all domains.

4 Results and Discussion

4.1 Part A – Process Reward Model and PRM-Guided Agent Training

Our initial experiments on the 39-task subset highlight several practical issues in the OS-World setup and in our reward-labeling design. First, by inspecting individual trajectories, we observed that using sleep_after_execution = 0 often led to screenshots being captured before the last GUI action had fully completed, which in turn corrupted the visual state that the agent and annotator see. In addition, a maximum of 15 steps was insufficient for certain longer-horizon tasks, causing premature termination even when the agent was progressing reasonably. We therefore switched to a more conservative configuration with sleep_after_execution = 0.5 and max_steps = 25, and found that this change substantially reduced visual race conditions and allowed most of the previously truncated cases to complete successfully.

Second, we identified an important issue in the scope of individual "actions" as they appear in the trajectories. A single natural-language response from the agent can be parsed into multiple low-level computer-use actions (as shown in Figure 1, a "scroll" sequence might be decomposed into mouse_move followed by scroll, or a text entry sequence into type followed by enter). As a result, the same logical step may correspond to multiple action records sharing the same step_num, which complicates both step counting and reward assignment.

Figure 1: (Up) Useful separation of "type" and "enter"; (Down) Useless separation of "mouse_move" and "scroll".

Third, we conducted an analysis of the LLM-based reward statistics on the 39-task subset. For each step, we ran the reward annotator multiple times using both GPT-4o-mini and GPT-5-mini, collected the resulting scalar rewards, and (offline) computed descriptive statistics—mean, median, and standard deviation—for analysis. Across tasks and anno-

Table 2: PRM evaluation on OSWorld test set (505 three-step windows). Confusion matrix counts are reported along with derived metrics.

Model / Setting	TP	FP	FN	TN	Acc.	Prec.	Rec.	F1
Evaluation protocol: After PRM fine-tuning on 184 generated training tasks, we evaluated on the OSWorld 39-task test split. Each task trajectory was truncated into 3-step windows, yielding 505 PRM-evaluated examples. Here, *pred=True* corresponds to the PRM predicting a positive step (reward > 0), and *pred=False* corresponds to predicting a non-positive step (reward ≤ 0).								
Qwen3-VL-4B (zero-shot PRM)	111	9	304	81	38.02%	92.50%	26.75%	41.50%
Fine-tuned PRM (LLaMA-Factory)	313	41	102	49	71.68%	88.42%	75.42%	81.40%

tator models, the empirical reward distribution is heavily skewed toward the upper range (approximately 0.9–1.0), as seen in the histograms in Figure 2. We hypothesize that this saturation effect arises because the annotator, when given only the local "step-before" context, finds it difficult to confidently label most actions as strongly harmful; conditioned on a reasonably coherent history, many next actions appear at least "acceptable," so the model rarely outputs low scores. To mitigate this, we are revising the annotation scheme in two ways: enlarging the context window so the annotator sees a longer history (and optionally the "after" state), and simplifying the reward space from a continuous range to a binary 0/1 label. Experiments with this revised design are currently in progress, and we expect the resulting rewards to be less saturated and more discriminative for PRM finetuning.

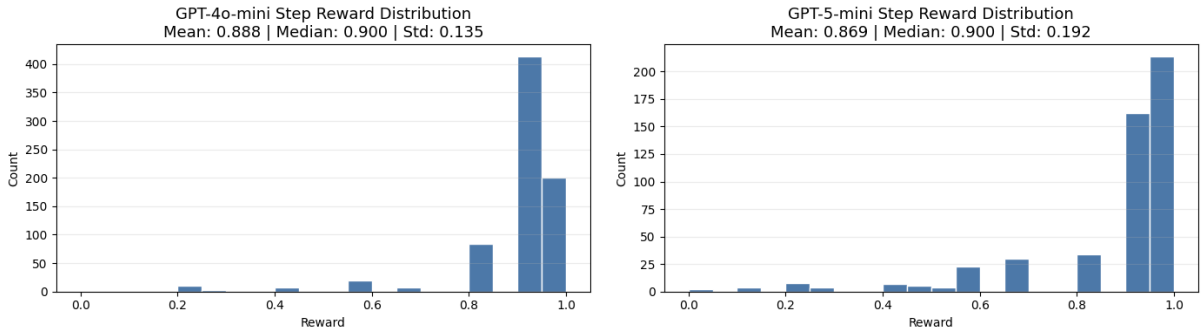

Figure 2: Reward Distribution for GPT-4o-Mini and GPT-5-mini

Finally, as shown in Table 2, fine-tuning yields a clear improvement on OSWorld, increasing accuracy from 38.02% to 71.68%. The main change is a substantial reduction in false negatives (higher recall), while precision remains high with a modest increase in false positives. Overall, the fine-tuned PRM provides a much more reliable signal for identifying beneficial intermediate steps.

4.2 Part B – Internal World Model for GUI Agents

Our initial experiments on four WebVoyager domains reveal both promising improvements and areas requiring further investigation. Table 3 summarizes the success rates comparing

the baseline (success-only retrieval, as in original CoMEM) against our contrastive world model with dynamic retrieval.

Table 3: Success rates (%) on WebVoyager domains. Baseline uses success-only retrieval; World Model uses contrastive retrieval with dynamic step guidance. Δ shows percentage point change.

Domain	**Baseline**	**World Model**	**Δ**
Google Map	16.67%	26.83%	**+10.16**
Amazon	19.51%	39.02%	**+19.51**
Allrecipes	15.91%	11.11%	-4.80
Coursera	2.38%	9.52%	**+7.14**

First, we observe substantial improvements on three of the four domains. Amazon shows the largest gain, with success rate doubling from 19.51% to 39.02% (+19.51 percentage points). Google Map improves by 10.16 percentage points (from 16.67% to 26.83%), and Coursera—despite starting from a very low baseline of 2.38%—quadruples to 9.52%. These gains suggest that contrastive guidance, which explicitly highlights both successful patterns and common failure modes, helps the agent avoid previously observed mistakes and make more informed action selections.

Second, we identified a regression on Allrecipes, where success rate dropped from 15.91% to 11.11%. By inspecting individual trajectories, we hypothesize two contributing factors: (1) the failure trajectory corpus for Allrecipes may be insufficiently diverse, causing the world model to inject overly specific warnings that do not generalize to the current task; and (2) the additional guidance text (∼200 tokens per step) may occasionally distract the agent from the primary task instruction on simpler navigation tasks. We plan to address this by implementing domain-specific retrieval thresholds and filtering low-confidence guidance when retrieved trajectories have low similarity scores.

Third, we note that the current implementation uses text-based contrastive summaries generated by the same LLM (Qwen2.5-VL-7B) that serves as the base agent. This introduces a potential coupling where the summarizer's interpretation of failure patterns may not align with what the agent policy actually needs to avoid. In future work, we will explore using a separate, larger model for contrastive analysis, or alternatively, learning latent contrastive embeddings that bypass natural language summarization entirely.

5 Conclusion

In this report we presented the first stage of our study on inference-time scaling for GUI agents.

For Part A: building a process reward modeling pipeline on top of OSWorld and ZeroGUI. We defined ten concrete GUI domains, generated and curated 184 tasks, and collected trajectories with a Qwen3-VL-4B-based agent under different environment configurations.

Along the way, we identified and fixed several practical issues that materially affect both behavior and supervision quality, including insufficient sleep time between actions, too-short step limits for long-horizon tasks, and overly fine-grained action logging that needed filtering and reindexing. Using multimodal LLMs (GPT-4o-mini and GPT-5-mini), we obtained step-wise rewards on a 39-task subset and analyzed their statistics, which revealed significant saturation in the original continuous reward design. This, in turn, motivated our ongoing shift toward larger context windows and a simplified 0/1 reward scheme, as well as an iterative finetuning loop in which a large PRM is trained on annotated trajectories and later distilled and used to guide PRM-based agent finetuning, followed by evaluation on held-out OSWorld and AndroidWorld tasks without invoking the PRM at test time.

For Part B: building an internal world model with contrastive trajectory retrieval on top of the CoMEM framework. We extended the original success-only memory retrieval to maintain separate FAISS indices for successful and failed trajectories, enabling contrastive pair retrieval that surfaces both positive examples and common failure modes. We implemented dynamic retrieval that re-queries the trajectory store at each step using the current screenshot, ensuring guidance remains relevant as the agent navigates through different page states. Initial experiments on four WebVoyager domains show promising results: Amazon success rate doubled from 19.51% to 39.02%, Google Map improved by 10 percentage points, and Coursera quadrupled from its low baseline. However, we also observed a regression on Allrecipes, highlighting several limitations in our current contrastive approach. First, our current implementation relies on **text-based summarization** of contrastive patterns rather than learned representations—the LLM must verbalize why certain trajectories succeeded or failed, which may lose subtle visual or sequential cues that are difficult to express in natural language. Second, we do not yet employ an **explicit contrastive loss** (e.g., InfoNCE) to learn an embedding space where success and failure trajectories are explicitly separated; instead, we simply retrieve from pre-computed CLIP embeddings that were not optimized for GUI task similarity. Third, the **corpus imbalance** between success and failure trajectories varies significantly across domains—some domains have abundant failure examples while others have few, leading to inconsistent guidance quality. Fourth, using the **same LLM** (Qwen2.5-VL-7B) for both the agent policy and contrastive summarization introduces a potential coupling where the summarizer's interpretation may not align with what the agent actually needs to avoid. Finally, the **additional context overhead** of injecting $\sim$200 tokens of guidance per step may occasionally distract the agent on simpler tasks where minimal guidance would suffice.

In future iterations, we plan to address these limitations by introducing a **latent contrastive learning** framework: training a shared encoder with contrastive losses to map both successful and failed trajectories into an embedding space where task-relevant similarities are explicitly optimized, enabling the agent to evaluate candidate actions based on learned proximity to success versus failure clusters without relying on natural language summarization.

References

Bai, Shuai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge et al. 2025. "Qwen3-VL Technical Report." *arXiv preprint arXiv:2511.21631*

He, Hongliang, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. "WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models." In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*. Association for Computational Linguistics. [Link]

Johnson, Jeff, Matthijs Douze, and Hervé Jégou. 2019. "Billion-Scale Similarity Search with GPUs." *IEEE Transactions on Big Data* 7 (3): 535–547

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. "Efficient memory management for large language model serving with pagedattention." In *Proceedings of the 29th symposium on operating systems principles*.

Li, Junnan, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. 2023. "BLIP-2: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation." *arXiv preprint arXiv:2301.12597*

Lightman, Hunter, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. "Let's Verify Step by Step." In *Proceedings of the International Conference on Learning Representations (ICLR 2024)*.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. "Learning Transferable Visual Models From Natural Language Supervision." In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*.

Rawles, Christopher, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala et al. 2024. "Androidworld: A dynamic benchmarking environment for autonomous agents." *arXiv preprint arXiv:2405.14573*

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." [Link]

Wu, Wenyi, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. 2025. "Auto-Scaling Continuous Memory For GUI Agent." *arXiv preprint arXiv:2510.09038*

Xie, Tianbao, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. "OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments." [Link]

Yang, Chenyu, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. 2025. "ZeroGUI: Automating Online GUI Learning at Zero Human Cost." [Link]

Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." [Link]

Zhang, Ziniu, Shulin Tian, Liangyu Chen, and Ziwei Liu. 2024. "MMInA: Benchmarking Multihop Multimodal Internet Agents."

Appendix(Q2 Project Proposal)

1. Motivation and Problem

Graphical User Interface (GUI) agents are an emerging class of AI systems that operate directly on real applications—opening apps, clicking buttons, typing into forms, and navigating complex workflows on operating systems, browsers, and mobile devices. Such agents would be useful both in industry and in research.

Modern GUI agents based on large vision–language models (LVLMs) can follow natural language instructions in desktop or mobile environments, but they remain unreliable on long-horizon tasks. Even strong models often get stuck, mis-click, or hallucinate progress when the interface changes slightly or when tasks require many coordinated steps. Existing benchmarks such as OSWorld and AndroidWorld (Xie et al. 2024; Rawles et al. 2024) report relatively low success rates for multi-step tasks, and most current systems rely on simple success/failure signals at the end of an episode.

Our project aims to improve the reliability of GUI agents by scaling inference with richer training signals: dense process rewards (PRMs) and lightweight internal world models. PRMs provide step-level supervision that helps agents learn which actions genuinely advance the task and which actions are redundant or harmful. However, PRMs alone do not address a second major source of failure in GUI agents: the inability to reason about *why* certain strategies succeed or fail across different contexts.

This motivates the second component of our approach—an internal world model that enables the agent to leverage past experience during inference. GUI agents often repeat the same failure patterns: clicking distractor elements, navigating to irrelevant pages, or skipping key verification steps. These mistakes arise because a purely reactive LVLM processes each screenshot independently, without recalling how similar tasks have succeeded or failed in the past. An internal world model addresses this gap by retrieving and analyzing both successful and failed trajectories from similar tasks, allowing the agent to recognize common pitfalls, identify critical decision points, and adjust its strategy before making mistakes.

Viewed through the lens of inference-time scaling, the internal world model increases the amount of computation the agent performs at inference—retrieving past trajectories, comparing alternative strategies, and reasoning about likely outcomes—without increasing model size. This transforms the agent's behavior from short-sighted, single-step predictions into a form of structured deliberation grounded in past experience.

Together, dense PRMs and internal world models offer complementary benefits: PRMs strengthen training-time learning signals, while world models enhance inference-time reasoning. By combining both, we aim to substantially improve GUI agent reliability, robustness, and generalization on challenging multi-step tasks.

2. Literature Review

2.1 Part A - Process Reward Model

From the literature, both OSWorld and ZeroGUI (Xie et al. 2024; Yang et al. 2025) highlight that even advanced LVLM-based GUI agents struggle with realistic multi-window, multi-application tasks, with success rates often below 50% on harder categories. OSWorld provides a suite of long-horizon desktop tasks spanning office suites, browsers, media players, email clients, development tools, and system-level workflows, and exposes a standardized API for screenshot capture and action execution. ZeroGUI builds on this by automating task generation, metadata management, and evaluation for large collections of GUI tasks, making it possible to study agent behavior at scale rather than on a small set of hand-designed scenarios. Together, these benchmarks show that current agents frequently mis-click, fail to recover from minor errors, and time out on tasks that require careful, multi-step coordination across windows and applications.

At the same time, work on reinforcement learning from human feedback (RLHF) and process reward models in text domains suggests that *how* feedback is provided can be as important as the model architecture itself. Instead of supervising only the final outcome of a trajectory (e.g., whether a summary is aligned with human preferences), recent methods train models using dense, step-level feedback: annotators or auxiliary models label intermediate reasoning steps, partial solutions, or chain-of-thought fragments with preference signals, and a separate reward model is trained to predict these scores. This process supervision has been shown to improve reasoning quality, reduce reward hacking, and make learning more data-efficient compared to pure outcome-based RLHF, especially on tasks where a single scalar at the end is too coarse to capture nuanced good or bad behavior.

However, these process reward ideas have not been fully explored in the context of GUI agents. Most existing GUI-agent work either optimizes for terminal success/failure labels or uses simple heuristic rewards based on environment signals (e.g., whether a particular pixel region changes or a specific dialog appears). In complex GUI environments, this coarse supervision can be misleading: an agent might take many redundant or even harmful actions, yet still eventually stumble into a successful terminal state, or conversely make several good intermediate moves but fail due to a single late error. In both cases, a binary success/failure label provides little guidance on which parts of the trajectory were actually helpful. This gap motivates designing process reward models that explicitly score each step of a GUI trajectory, rewarding actions that make genuine progress toward the goal and penalizing those that are redundant or harmful.

In our Q1 work, we implemented a baseline Qwen3-VL-4B GUI agent in OSWorld/ZeroGUI and collected trajectories on a subset of ~ 40 tasks. Our preliminary experiments showed:

- The baseline agent can complete some shorter tasks, but performance degrades for longer sequences and tasks requiring careful coordination.
- When we queried GPT-5-mini to label each step with a scalar reward and a rationale, we observed that:
  - Many steps received similar scores, indicating reward saturation with some an-

notators.
  - There were systematic differences between annotators: one model produced a narrower, more discriminative reward distribution, suggesting it may be better suited as a teacher for training a PRM.
- We also found that naïvely decomposing a single natural-language agent response into many low-level actions (mouse moves, scrolls, key presses) can blur what the reward should mean at each step, motivating a cleaner action representation for PRM training.

These observations both confirm the reliability problem in current GUI agents and suggest how to refine action representation, reward design, and PRM training to better handle long-horizon GUI behavior.

2.2 Part B – Internal World Model

From the literature, recent GUI-agent benchmarks such as WebVoyager, Mind2Web, and MMInA (He et al. 2024; **?**; **?**) consistently show that long-horizon, multi-page navigation tasks remain challenging for LVLM-based GUI agents. Even strong models frequently fail due to unstable reasoning, repeated exploration loops, or incorrect assumptions about page structure after small layout changes. Existing memory-augmented systems such as CoMEM retrieve only successful trajectories, which prevents agents from learning about systematic failure modes. Moreover, recent work on LLM-based world-model planning shows that even advanced simulators such as WebDreamer struggle with accurate multi-step state prediction, with hallucinated transitions and error accumulation emerging as key bottlenecks for long-horizon GUI reasoning (**?**).

Our Q1 work confirmed these issues. We implemented an internal world model on top of the CoMEM framework and evaluated it across several WebVoyager domains. Our results demonstrate:

- **Baseline brittleness.** Qwen2.5-VL-7B acting without world-model guidance frequently stalls, repeats ineffective actions, or navigates to irrelevant page regions. These failures are especially common in multi-step tasks requiring coordinated reasoning (e.g., filtering products, navigating hierarchical menus, or verifying item attributes).
- **Limitations of success-only retrieval.** The agent often repeated known failure patterns—such as clicking distractor elements or skipping essential verification steps—because success-only retrieval offered no mechanism to surface common mistakes.
- **Usefulness of contrastive retrieval.** By separately indexing successful and failed trajectories, we uncovered meaningful divergence points. For example:
  - Successful Amazon trajectories consistently verified product variants before adding to cart.
  - Failed trajectories often acted prematurely (e.g., selecting the first visible result without checking details).
- **Dynamic retrieval improves relevance.** Retrieval based on the current screenshot (rather than only the initial one) surfaces different relevant examples as the agent

progresses. About 40% of retrieved examples changed between steps 0 and 5, indicating that state-aware retrieval is crucial for long-horizon reasoning.
- **Empirical performance gains.** Initial experiments showed improvements in three of four domains (e.g., Amazon success rate increased from 19.51% to 39.02% under dynamic contrastive retrieval). This provides direct evidence for the value of internal world models.

These findings show that GUI agents exhibit systematic, repeated failure patterns not captured by success-only retrieval. Contrastive internal world models offer a principled way to address these reliability issues through inference-time scaling.

Additional Evidence from Q1 Analysis. Beyond the core observations above, deeper inspection of trajectories revealed several structural challenges:

- **Latent task structure is often invisible to LVLMs.** Many tasks require implicit multi-step dependencies (e.g., verifying item details before proceeding). Without access to similar examples, the agent often applies incorrect heuristics.
- **Attention misalignment contributes to repeated failure.** Visually salient but irrelevant elements (ads, banners, large thumbnails) frequently draw the agent away from task-relevant regions. Successful trajectories typically include subtle corrective actions (scrolling or ignoring distractors).
- **Failure trajectories come in distinct categories.** Q1 identified:
  1. premature-action failures,
  2. navigation loops,
  3. attention-drift failures,
  4. incomplete-action failures.
- **Static retrieval cannot capture trajectory evolution.** Long tasks involve multiple page changes; retrieval must adapt step-by-step to avoid irrelevant memory recall.
- **Failure cascades dominate long-horizon tasks.** A single early mis-click often leads to drifting into irrelevant states, from which the baseline model rarely recovers.

These extended findings further justify a world model that continually retrieves, contrasts, and reasons over prior experience.

3. Datasets

3.1 Part A - Process Reward Model

Our future work will primarily use the following data sources:

1. **OSWorld (desktop GUI tasks).**
   - We will continue to use OSWorld as our core training and evaluation benchmark, focusing on multi-step navigation, file operations, and web-browsing tasks.
2. **ZeroGUI-generated tasks and metadata.**
   - We will use the ZeroGUI task generator and metadata tools to expand our task set and systematically vary difficulty, interface layouts, and objective types.
3. **Trajectory dataset from the Qwen3-VL baseline agent.**

- This includes sequences of screenshots, natural-language thoughts, actions, and environment feedback for each step in each task.

4. **LLM-based step-wise reward annotations.**
   - For each step in each trajectory, we already have multiple scalar rewards and rationales from GPT-5-mini. We will continue to expand this labeled dataset as we collect more trajectories.
5. **AndroidWorld.**
   - Time permitting, we will test whether PRM-guided training improves generalization to mobile-style tasks using AndroidWorld, which has a similar interface but different visual layouts and action patterns.

3.2 Part B – Internal World Model

For Part B, our work will rely on the following data sources:

1. **WebVoyager trajectory datasets.**
   - We will continue using WebVoyager as our primary benchmark, covering e-commerce, search, travel, news, and developer platforms.
   - Each trajectory contains screenshots, action sequences, and success/failure labels.
2. **Contrastive trajectory corpus (success + failure).**
   - In Q1, we constructed separate FAISS indices for successful and failed trajectories for each domain.
   - We will expand these datasets to improve coverage of common failure patterns.
3. **CLIP and multimodal embeddings.**
   - Each trajectory is represented using CLIP-based text embeddings and multimodal embeddings (task description + first screenshot).
   - These embeddings serve as inputs for retrieval, contrastive learning, and latent world-modeling.
4. **LLM-generated contrastive summaries.**
   - Our Q1 system generated natural-language descriptions of success patterns, failure modes, and divergence points.
   - These will serve as intermediate supervision targets as we move toward a latent learned internal world model.

Additional Data Considerations.

- **Failure categories are richly represented.** Our corpus includes diverse failure signatures across domains (e.g., improper filtering on Amazon, incorrect view switching in Maps).
- **Embedding limitations motivate Q2 improvements.** CLIP embeddings:
  - over-cluster visually similar but semantically different screens,
  - under-cluster semantically similar screens with minor visual variations,
  - fail to encode GUI layout structure.
- **Trajectory metadata can support hierarchical modeling.** Each trajectory contains:

1. screenshot sequences,
2. text observations,
3. actions,
4. page transitions,
5. success/failure labels.

- **Targeted rollouts will enrich the training set.** We will generate:
  - more long-horizon tasks,
  - more failure-heavy examples,
  - more trajectories with subtle divergence.

4. Preliminary EDA

4.1 Part A - Process Reward Model

We have already performed a first round of exploratory data analysis on the collected trajectories and rewards:

- **Reward distributions.** We plotted the distribution of step-wise rewards for GPT-4o-mini and GPT-5-mini. GPT-4o-mini tends to give scores clustered near the top of the scale, while GPT-5-mini uses a wider range and better separates "slightly helpful," "neutral," and "harmful" actions. This suggests that GPT-5-mini's labels may be more informative for training a PRM, while GPT-4o-mini may need calibration or rescaling.

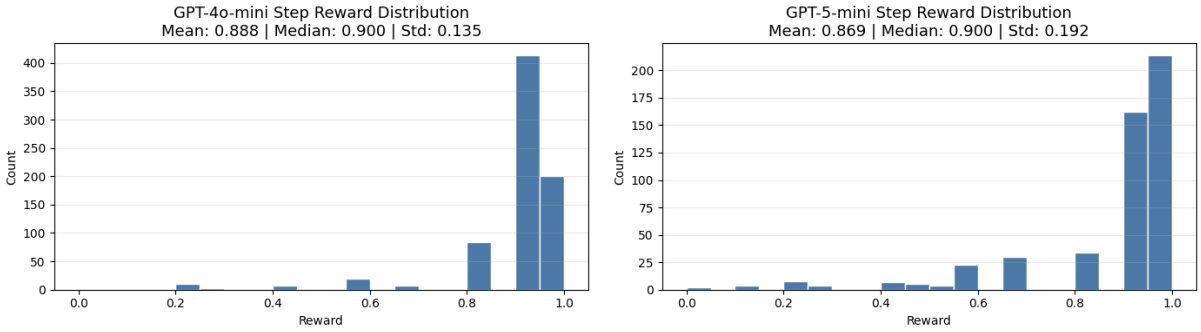

Figure 3: Reward Distribution for GPT-4o-Mini and GPT-5-mini

- **Correlation with task outcomes.** Steps in successful episodes tend to have higher average rewards than steps in failed episodes, but there is significant overlap. This indicates that the reward signal is meaningful but noisy, so our PRM training will likely benefit from techniques such as label smoothing, variance reduction, or pairwise ranking (e.g., preferring "better step vs. worse step") instead of pure regression.
- **Action-type analysis.** We observed that some low-level actions (e.g., very small mouse moves, micro scrolls) often receive ambiguous rewards and contribute little to task progress. In contrast, higher-level actions like "submit search," "confirm dialog," or "open correct file" have strong positive or negative impact. This motivates a more

semantic action representation for PRM training (grouping low-level operations into higher-level "macro-steps") so the PRM predicts rewards over meaningful decisions rather than noisy primitives.

These EDA findings directly shape our training plan: we will (1) choose the more discriminative annotator as the primary teacher, (2) design the PRM loss and architecture to be robust to noisy labels, and (3) define macro-steps that better capture decision points in the trajectories.

4.2 Part B – Internal World Model

Our exploratory analyses of the WebVoyager corpus and the Q1 internal world model reveal several important properties:

- **Success and failure trajectories exhibit distinct structure.** CLIP embeddings show that successful trajectories cluster more tightly than failed ones. Failure trajectories are more dispersed, suggesting that modeling failure explicitly is essential for robust behavior.
- **Contrastive signals are strong but noisy.** Early actions in successful and failed trajectories often look similar, but diverge sharply at critical decision points. This supports the use of:
  - InfoNCE or supervised contrastive losses,
  - divergence-point detection,
  - mid-trajectory similarity-based retrieval.
- **Dynamic retrieval captures evolving task context.** Static (step 0 only) retrieval often surfaces irrelevant examples after a few steps. Dynamic retrieval adapts to state changes and maintains higher semantic relevance throughout long tasks.
- **Natural-language guidance is helpful but sometimes distracting.** For complex tasks, longer textual guidance improves reasoning; for simpler domains (e.g., Allrecipes), too much guidance can distract the agent. This suggests:
  - confidence-weighted guidance,
  - adaptive guidance length based on retrieval similarity,
  - and ultimately, learned latent guidance in place of verbose summaries.

Additional EDA Insights.

- **Similarity drift predicts failure.** In failing trajectories, similarity to successful examples declines steadily across steps; in successful trajectories, similarity stabilizes or increases.
- **Cross-domain generality of divergence patterns.** Failure signatures in Amazon (e.g., premature selection) also occur in Booking, Yelp, and Expedia tasks.
- **Error propagation chains.** Many failures follow a predictable pattern:
  1. early mis-click,
  2. transition to unintended state,
  3. agent confusion,
  4. exploration of irrelevant regions,

5. terminal failure.

- **Retrieval instability reflects agent uncertainty.** Large fluctuations in nearest-neighbor embeddings correlate with low success rates.
- **Embedding space limitations.** Screens with small but crucial differences (e.g., correct vs. incorrect product variant) are often embedded too closely in CLIP space; updating this is a major Q2 goal.

5. Future Works

5.1 Part A - Process Reward Model

Our overarching goal is to demonstrate that inference-time scaling with PRMs and internal world models can significantly improve GUI agent reliability on unseen tasks, compared to a strong LVLM baseline.

Concretely, our future work (mostly training-focused) is structured into several stages:

1. **Refine the trajectory dataset and macro-step representation.**
   - We will first clean and standardize the existing Qwen3-VL-4B trajectories collected on OSWorld/ZeroGUI tasks. This includes removing corrupted episodes, normalizing metadata, and aligning screenshots, actions, and rewards.
   - Next, we will transform low-level actions (fine-grained mouse movements, micro scrolls, keypress bursts) into semantically meaningful macro-steps. A macro-step will correspond to a human-interpretable decision such as "open the correct file", "submit a search query", or "confirm a dialog". This representation will be derived by grouping contiguous low-level actions that share a common short-term intent (e.g., moving the cursor to a button and clicking it).
   - For each macro-step, we will aggregate the original step-wise LLM rewards (e.g., average, max, or a robust statistic) to produce a macro-level reward label and rationale, which will serve as training targets for the PRM.
2. **Train a Process Reward Model for GUI trajectories.**
   - We will build a compact PRM that takes as input the current screenshot, the task instruction, and a candidate macro-step (represented by its action type and a short textual description) and predicts a scalar reward.
   - Architecturally, we plan to start with a lightweight vision–language encoder (e.g., a frozen or partially frozen LVLM backbone plus a small MLP head) so that the PRM can be efficiently evaluated many times per episode. We will compare variants that condition on:
     (a) only the "before" state and proposed action, and
     (b) both "before" and "after" states to capture actual state changes.
   - For the training objective, we will experiment with:
     – *Direct regression* to calibrated LLM scores, to provide an initial baseline;
     – *Pairwise ranking losses* where the PRM is trained to assign higher scores to empirically better steps (within the same task or local context);
     – *Variance-aware weighting* that reduces the influence of highly ambiguous la-

bels by using annotator disagreement (when multiple teachers are available) as an uncertainty signal.
    - We will evaluate the PRM on held-out tasks and annotators, measuring calibration (e.g., correlation with teacher scores), ranking quality (e.g., fraction of correctly ordered pairs), and robustness to distribution shift across task types.
3. **Integrate the PRM into GUI agent training.**
    - Once we have a reasonably calibrated PRM, we will use it as a dense critic or scoring function to guide policy improvement on top of the Qwen3-VL-4B baseline.
    - In the *off-policy phase*, we will re-weight existing trajectories using PRM scores. High-reward macro-steps will be prioritized for imitation or behavior cloning, while low-reward steps will be downweighted or filtered out. This allows us to extract the most informative supervision signal from already collected data without immediately requiring new environment rollouts.
    - In the *on-policy phase*, we will collect new trajectories where the agent's actions are updated to maximize PRM-predicted reward. We will explore standard RL algorithms (e.g., policy gradient with PRM as a value proxy) and simpler bandit-style updates (e.g., PRM-guided reranking of candidate actions at each step).
    - Throughout training, we will monitor stability indicators such as reward hacking (e.g., the agent finding PRM "loopholes"), distributional collapse of actions, and divergence from the original LVLM behavior, and we will adjust the strength of PRM guidance accordingly.
4. **Evaluate PRM-guided agents on OSWorld and AndroidWorld.**
    - We will compare the PRM-guided agent to the original Qwen3-VL-4B baseline on OSWorld and, time permitting, AndroidWorld. Key metrics will include:
        - overall success rate on held-out tasks,
        - number of steps taken to complete tasks (efficiency),
        - and distributions of error types (e.g., mis-click, navigation loops, premature actions).
    - We will perform ablation studies to understand the contribution of each component: macro-step representation vs. low-level actions, regression vs. ranking objectives, and off-policy vs. on-policy PRM guidance.
    - Finally, we will analyze qualitative examples of trajectories where PRM guidance clearly improves behavior (e.g., avoiding redundant clicks, recovering from early mistakes) and cases where it fails, to inform future iterations.

By the end of the project, we aim to deliver:

- a cleaned and documented trajectory + reward dataset for GUI tasks (including macro-step annotations),
- a trained and evaluated GUI PRM,
- a PRM-guided GUI agent that outperforms the baseline on OSWorld and AndroidWorld, and
- analysis of how inference-time scaling with PRMs and world models affects reliability, failure modes, and generalization.

5.2 Part B – Internal World Model

Our overarching goal for Part B is to develop an internal world model that improves GUI-agent reliability by learning from both successful and failed trajectories, enabling informed and safer decision-making during inference.

Our Q2 objectives include:

1. **Develop a latent contrastive world model.**
   - Train an encoder that maps success and failure trajectories into a latent space where they are explicitly separated.
   - Apply contrastive objectives (e.g., InfoNCE, supervised contrastive loss) to align embeddings with task outcomes.
   - Use the latent space to estimate success likelihood of candidate actions.
2. **Improve dynamic, state-aware retrieval.**
   - Train retrieval embeddings that incorporate the current screenshot and recent history.
   - Replace CLIP-only retrieval with latent embedding similarity.
   - Dynamically adjust retrieval thresholds based on similarity confidence.
3. **Integrate the world model into the agent policy loop.**
   - Use latent signals to rank candidate actions or provide structured hints.
   - Explore hybrid prompting approaches combining latent signals with minimal natural-language guidance.
4. **Evaluate improvements over Q1 baselines.**
   - Measure success rate, navigation efficiency, and reduction in repeated failure modes.
   - Perform ablations: static vs. dynamic retrieval, success-only vs. contrastive memory, text-based vs. latent guidance.

Additional Planned Work.

- **Hierarchical world models.** Build multi-level representations capturing:
  1. screen-level visual patterns,
  2. action-level transitions,
  3. trajectory-level semantics.
- **Uncertainty-aware guidance.** Compute retrieval entropy and similarity variance to modulate guidance strength.
- **Failure signature modeling.** Cluster systematic failure types to build predictive failure warnings.
- **Model-based scored action ranking.** Score candidate actions using latent divergence risk estimation.
- **Expanded evaluation metrics.** Include divergence recovery rate, repeated failure reduction, and long-horizon consistency metrics.

By the end of Q2, we aim to deliver:

- a scalable contrastive trajectory memory system,
- a latent internal world model capable of predicting and avoiding failures,

- an improved GUI agent that outperforms our Q1 baselines,
- and insights into how inference-time scaling via internal world models improves GUI-agent reliability.

References

Bai, Shuai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge et al. 2025. "Qwen3-VL Technical Report." *arXiv preprint arXiv:2511.21631*

He, Hongliang, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. "WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models." In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*. Association for Computational Linguistics. [Link]

Johnson, Jeff, Matthijs Douze, and Hervé Jégou. 2019. "Billion-Scale Similarity Search with GPUs." *IEEE Transactions on Big Data* 7 (3): 535–547

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. "Efficient memory management for large language model serving with pagedattention." In *Proceedings of the 29th symposium on operating systems principles*.

Li, Junnan, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. 2023. "BLIP-2: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation." *arXiv preprint arXiv:2301.12597*

Lightman, Hunter, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. "Let's Verify Step by Step." In *Proceedings of the International Conference on Learning Representations (ICLR 2024)*.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. "Learning Transferable Visual Models From Natural Language Supervision." In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*.

Rawles, Christopher, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala et al. 2024. "Androidworld: A dynamic benchmarking environment for autonomous agents." *arXiv preprint arXiv:2405.14573*

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." [Link]

Wu, Wenyi, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. 2025. "Auto-Scaling Continuous Memory For GUI Agent." *arXiv preprint arXiv:2510.09038*

Xie, Tianbao, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. "OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments." [Link]

Yang, Chenyu, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. 2025. "ZeroGUI: Automating Online GUI Learning at Zero Human Cost." [Link]

Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." [Link]

Zhang, Ziniu, Shulin Tian, Liangyu Chen, and Ziwei Liu. 2024. "MMInA: Benchmarking Multihop Multimodal Internet Agents."

Contributions

Bella: Led the repair and standardization of the PRM dataset pipeline to support end-to-end, training-ready data production. Trajectory generation was re-run with Qwen3-VL to confirm and lock down the exact output schema, followed by the development of an automated conversion workflow that exports validated JSONL files for PRM fine-tuning. In addition, schema checks and diagnostic scripts were implemented to summarize and quantify common formatting errors, enabling systematic tracking of data quality over time. Conducted a training smoke test to verify that the dataloader and training loop execute stably on the cleaned dataset, and an initial format specification was documented to improve reproducibility for collaborators.

Rita: Focused on stabilizing the fine-tuning and training infrastructure required for reproducible PRM and downstream agent training experiments. The LLaMA-Factory fine-tuning pipeline was brought into a working state by resolving dependency and version incompatibilities across CUDA/PyTorch/transformers and related components, and a single-GPU run was validated (5 epochs with checkpoints saved every 150 steps). Integration points across the pipeline were debugged and tightened through standardized scripts and configurations to ensure consistent reruns. In parallel, OSWorld environment compatibility and runtime issues were addressed to enable reliable online execution; when the original training stack integration proved more involved than expected, an existing OSWorld integration setup was used as a reference to complete a working end-to-end pipeline and prepare controlled training runs and ablations.

Shuchang: Focused on the development and stabilization of the contrastive scoring and ranking logic for the internal world model. Included designing the evidence aggregation pipeline over retrieved success and failure trajectories, implementing confidence-aware rules to handle ambiguous or conflicting signals, and analyzing retrieved neighbors to diagnose cases where visual similarity did not align with task relevance. Also investigated improvements to the query representation to better capture task context and improve the reliability of contrastive evidence.

Ziyu: My contribution focused on strengthening the internal world model and improving the reliability of memory-guided reasoning. I extended the original memory system to support contrastive indexing by organizing and maintaining separate stores for successful and failed trajectories, enabling side-by-side retrieval of positive and negative examples. I implemented step-level, state-aware querying so that memory retrieval adapts to the agent's current screen context rather than relying solely on the initial task description. To address instability in retrieved evidence, I developed aggregation and confidence-filtering strategies that reduce the influence of weak or visually similar but semantically irrelevant neighbors. I also conducted targeted analyses of divergence points and repeated failure patterns across domains, helping identify when guidance improved decision quality and when it introduced distraction or bias. Finally, I supported structured ablation experiments comparing success-only retrieval, static contrastive memory, and dynamic contrastive retrieval, ensuring that improvements from the world model were empirically validated rather than qualitatively assumed.