Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Gelato-30B-A3B: Training a State of the Art Model Grounding Model

Master's Thesis

Aylin Güliz Akkus
aakkus@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Prof. Dr. Roger Wattenhofer, Prof. Dr. Ludwig Schmidt

February 20, 2026

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Prof. Dr. Roger Wattenhofer, Prof. Dr. Ludwig Schmidt, and Prof. Dr. Yejin Choi, for their invaluable guidance and support throughout my Master's thesis. Their expertise and insightful feedback have been instrumental in shaping this research.

I am also grateful to Anas Awadalla and Drubha Ghosh for their collaboration on the project. The discussions and support within the team have been a great source of motivation. I furthermore thank Florian Brand for sharing details on OS-World task quality issues and Yan Yang for details on GTA1 agent evaluation.

I gratefully acknowledge computing time granted for the project synthesis by the JARA on the supercomputer JURECA at Jülich Supercomputing Center (JSC), as well as storage resources on JUST granted and operated by JSC and supported by the Helmholtz Data Federation (HDF).

Finally, I am deeply thankful to my life partner, Mert Unsal, for his constant love, patience, and understanding throughout this journey.

Unique Contributions. As most of this work was possible only as a collaborative effort, I would like to outline my unique contributions to this Master's thesis. In particular, I:

- Participated in the search and normalization of the existing data sources.
- Ran experiments with different filtering methods for the difficulty-based filtering, trained models, and evaluated the results on the offline benchmarks.
- Developed methods to utilize APIs (DOM trees and Accessibility Trees) to supplement training data beyond the web, leveraging richer semantic information to generate instructions.
- Studied the effect of dense vs. sparse rewards on model performance during reinforcement learning training.
- Wrote an agentic harness to evaluate the model on the OS-World benchmark and conducted human evaluation to mitigate the limitations of automated evaluators.
- Wrote this thesis in full (all text, figure selection and placement, and experimental narration).

Scope and Non-Contributions.

- I did not participate in the video annotation process for the supplemental data collection.
- I did not participate in running the online OS-World evaluations using containerization and VMs.

Abstract

Recent progress in machine learning has been driven largely by scaling compute, model size, and training data—yet of these three pillars, training data has received the least systematic attention. This gap is especially pronounced for Computer-Use agents, where limited heterogeneous data sources exist and data practices remain closed source.

In this thesis we apply a data-centric methodology to the Graphical User Interface (GUI) grounding problem and introduce **Click-100k**, an open-source dataset assembled through a complete pipeline of data curation, filtering, and quality control across diverse GUI domains. We then train **Gelato-30B-A3B**, a state-of-the-art grounding model for GUI computer-use tasks, on Click-100k using reinforcement learning. Gelato achieves 63.88% accuracy on ScreenSpot-Pro and 69.15 / 74.65% on OS-World-G / OS-World-G (Refined), surpassing prior specialized grounding models such as GTA1-32B and much larger vision-language models including Qwen3-VL-235B-A22B-Instruct. When paired with GPT-5 as the planning backbone, Gelato enables strong agentic performance at 58.71% automated success rate (61.85% with human evaluation) versus 56.97% (59.47% with human evaluation) for GTA1-32B on OS-World. We describe the complete pipeline—from data curation and filtering to reinforcement learning training—that enables these results and open-source both the dataset and the model.

Contents

Chapter 1

Introduction

1.1 Motivation

Recent progress in machine learning has been largely driven by scaling laws [1, 2, 3, 4], as exemplified by models such as GPT-4, CLIP and Stable Diffusion [5, 6, 7] These advances rest on three pillars: Compute, Model size, and Training data. Of these, training data has received the least systematic attention—despite its crucial role, datasets are rarely the subject of dedicated research [8].

On the modeling side, progress is comparatively well-understood: given sufficient compute, systematic sweeps over architecture width, depth, normalization, and training hyperparameters produce reliable, reproducible gains [9, 10]. Data, by contrast, remains a blind spot. The vast majority of large-scale training sets are proprietary, forcing the community to reverse-engineer or independently reconstruct them [11, 12, 13]. Efforts such as DataPerf, DataComp, and MetaCLIP [14, 15, 16] have made important strides toward standardized evaluation and reproducibility, yet dataset engineering as a discipline still lags far behind model engineering. We contend that it demands the same rigorous, principled treatment. In practice, large-scale dataset construction can be decomposed into two distinct phases: uncurated data collection and dataset filtering.

This challenge is especially pronounced in the emerging domain of Computer-Use agents. Graphical User Interfaces (GUIs) are central to how people interact with the digital world, serving as the primary medium for a wide range of daily tasks. Large Language Models (LLMs), originally developed as conversational chatbots [17, 5, 18], have since proven capable of far more: their ability to comprehend complex language instructions and seamlessly integrate external tools has revealed significant potential for automating complex workflows through Computer-Use agents [19, 20]. This progress has motivated the development of intelligent agents that can substantially streamline human–computer interaction based on natural language instructions.

Early work in this area focused on language-only agents [21, 22, 23] built on top of closed-source, API-based LLMs such as GPT-5 [24], leveraging text-

rich metadata like HTML inputs or accessibility trees to perform navigation and other tasks. However, this text-only paradigm faces fundamental limitations in practice: (1) unlike web environments, general Computer-Use scenarios do not always expose rich API-based metadata, and (2) users typically interact with interfaces visually—through screenshots—without access to the underlying structural information. These limitations underscore the need for Computer-Use *visual* agents that can perceive and interact with UIs in the same way humans do.

Despite this progress, training multi-modal models for GUI visual agents continues to face several significant challenges:

(a) **Availability of Unfiltered Training Data:** Whereas LLM and VLM pretraining can draw on vast pools of (albeit low-quality) data such as Common Crawl [25], the Computer-Use domain does not yet benefit from a comparable abundance of raw data. Multiple heterogeneous sources exist, but they must first be normalized and unified before any filtering can even begin. Even among open-source models [26], the training data or its exact specification is often unavailable, making it difficult to reproduce or build upon existing work.

(b) **Training Data Composition:** Mirroring the challenges faced by language-based Computer-Use agents, the availability of rich APIs for the web has led to a pronounced overrepresentation of web-based data in existing training sets. Professional desktop application data, by contrast, is considerably more difficult to obtain with reliable annotations, resulting in a significant domain imbalance.

(c) **Resolution and Aspect Ratio Variability:** If Computer-Use agents are to be deployed in real-world settings rather than on standardized virtual machines, they must contend with widely varying screenshot aspect ratios, resolutions, and sizes. This necessitates either resizing screenshots—at the cost of information loss—or training models that can generalize robustly across heterogeneous display configurations.

1.2 Background

1.2.1 Computer Use Agents

Computer-use agents enable Large Language Models (LLMs) to operate computers autonomously through a perception–action loop. The agent observes the interface state, predicts an action, executes it, and receives feedback until task completion. Formally, the agent learns a policy:

$$(\text{instruction}, \text{history}, \text{current observation}) \rightarrow \text{next action}$$

where actions are low-level operations (clicks, typing, scrolling) that decompose high-level natural language instructions into executable GUI interactions.

Language-based vs. Vision-based Agents Early language-only approaches operated over structured textual representations (DOM trees, accessibility trees) to generate tool calls (Figure 1.1a). While effective in constrained settings, these methods require privileged system access and handcrafted filtering pipelines, limiting generalization to visually-dependent tasks. Vision-based agents using VLMs instead process raw screenshots (Figure 1.1b), eliminating dependency on internal program structures. This paradigm shift changes the observation space from structured text to high-dimensional visual input, requiring models to extract semantic meaning and identify interactive elements directly from pixels.

End-to-End vs. Two-Stage Agents Computer-use agents can be categorized into two architectural paradigms based on how they decompose the task execution pipeline.

End-to-end agents process the complete perception-action cycle within a single model [27, 28]. These systems learns to jointly handle perception, planning, memory, and action execution in a unified forward pass. While conceptually simple, this approach requires the model to simultaneously master multiple capabilities: understanding visual interfaces, maintaining task context, reasoning about multi-step procedures, and predicting precise interaction coordinates. Examples include Claude Computer Use and recent native computer-use models that output tool calls directly from screenshots. Critically, training these end-to-end models requires large amounts of data trajectory data which is hard to collect and quality maintain.

Two-stage agents [29, 30] decompose the problem into planning and grounding, each handled by specialized modules:

- *Planner*: A high-level reasoning model (typically closed-source models like GPT-4o or Claude 3.7 Sonnet) generates action proposals for each step based on the user task instruction, using real-time UI screenshots and past trajectories as context. The planner outputs natural language descriptions of intended actions (e.g., "click the New Tab button").
- *Grounder*: A separate grounding module maps these natural language action proposals to specific UI elements by predicting precise screen coordinates. This vision-based component localizes semantic descriptions within the current screenshot.

This separation of concerns enables leveraging the reasoning capabilities of large proprietary models for planning while training specialized vision models for the

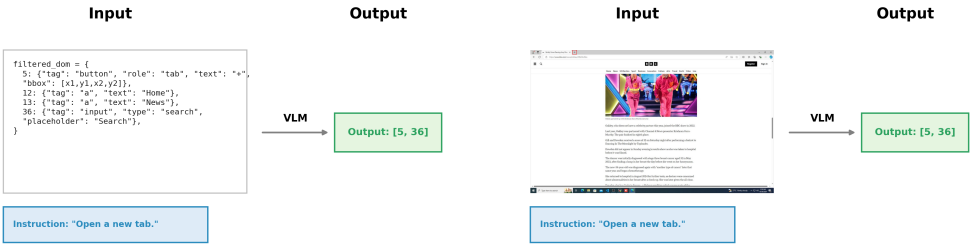
grounding task. The two-stage approach is particularly advantageous when training data for end-to-end computer use is limited, as grounding data (screenshot-instruction-coordinate triples) is more readily available than full task trajectories.

This work adopts the two-stage paradigm, focusing specifically on the grounding component. By training specialized vision-language models for GUI element localization, we enable integration with arbitrary planning models without requiring end-to-end retraining for each new planner configuration.

GUI Grounding GUI grounding maps abstract interaction intents to spatial locations. Given screenshot s and action proposal a, the grounding model predicts target coordinates:

$$f_{\text{ground}} : (s, a) \to (x, y)$$

This task is challenging due to information-dense layouts with numerous small elements, significant cross-platform interface variation, and reliance on visual cues (color, typography, proximity) for element discrimination. Early supervised approaches predicted bounding box centers, creating train-test misalignment: any coordinate within the target region constitutes valid interaction, yet center-point regression penalizes such predictions. Recent methods directly optimize for region-based success, aligning training objectives with actual interaction requirements.



(a) Language-based grounding: The model receives a filtered DOM tree as input and predicts coordinates based on structured text representations of UI elements.

(b) Visual grounding: The model operates directly on screenshots and predicts coordinates from high-dimensional visual input.

Figure 1.1: Comparison of grounding approaches. Both methods receive the instruction "Open a new tab" and predict coordinates, but differ in their input modality: (a) language-based grounding uses structured DOM representations, while (b) visual grounding processes raw screenshots.

1.2.2 GUI Grounding Datasets

Our training data comprises multiple open-source GUI grounding datasets, from which we extract desktop samples and pointwise click actions. When datasets provide full computer-use trajectories rather than individual grounding annotations, we extract click actions and generate corresponding instructions using large language models. A detailed overview of all data sources and sample counts is provided in Section 2.1.

General-Purpose Training Data Our primary data pool (Table 2.1 in Section 2.1) combines eight complementary sources. UGround [31] provides web-based samples synthesized from HTML structures and paired with natural language referring expressions. AutoGUI [32] contributes functionality-focused annotations generated by analyzing state changes before and after interactions. ShowUI [33] spans web, mobile, and desktop environments emphasizing cross-device generalization. OS-Atlas [34] covers multiple platforms (Windows, macOS, Linux, Android) synthesized using cross-platform tooling.

Additional sources include WaveUI [35] for diverse desktop interfaces, PC-Agent-E [36] providing trajectory-based Windows data, PixMo Points [37] offering high-quality visual grounding annotations, and SeeClick [38] contributing web and mobile UI samples. Together, these datasets provide broad coverage of standard consumer interfaces and common web interactions.

Professional Application Data To address underrepresentation of professional desktop software, we incorporate two specialized datasets. UI-Vision [39] provides densely annotated samples from real-world productivity applications, including professional tools typically inaccessible to automated collection methods. JEDI [40] contributes samples emphasizing complex desktop environments, office software, and system applications. These datasets specifically target high-resolution, functionally rich interfaces characteristic of professional workflows that dominate challenging benchmarks like ScreenSpot-Pro.

1.2.3 Evaluation Benchmarks

Offline Benchmarks

We evaluate grounding performance using four offline benchmarks and one online benchmark. Offline benchmarks pair a screenshot with a natural-language instruction and require the model to predict the coordinates of the referenced UI element; performance is measured via spatial containment between the predicted coordinates and annotated bounding boxes. They are fast and inexpensive to run.

ScreenSpot. ScreenSpot [41] was the first realistic GUI grounding benchmark spanning mobile, desktop, and web environments, comprising over 600 screenshots and more than 1,200 instructions. However, recent models have largely saturated ScreenSpot, achieving approximately 90% accuracy, which limits its discriminative power.

ScreenSpot-V2. ScreenSpot-V2 [42] addresses annotation quality issues in the original ScreenSpot benchmark by identifying and re-annotating 11.32% of incorrect samples, providing a more reliable evaluation baseline. Despite being nearly saturated, ScreenSpot-V2 was still used as a benchmark for evaluating grounding performance.

ScreenSpot-Pro. ScreenSpot-Pro [43] targets complex, high-resolution professional desktop environments. It contains 1,581 authentic full-screen images from 23 professional applications across five industries and three operating systems. Grounding targets are extremely small—on average only 0.07% of the screen area—and embedded in dense, multi-window workflows, making ScreenSpot-Pro substantially more challenging than its predecessors.

OS-World-G. OS-World-G [40] is a fine-grained grounding benchmark built on screenshots from the OS-World online environment. It comprises 564 manually annotated examples covering text matching, element recognition, layout understanding, fine-grained manipulation, and infeasible instruction handling. A corrected variant, OS-World-G (Refined), fixes annotation errors in the original set.

Online Benchmarks

Online benchmarks instead provide a high-level task instruction and grant the model access to a live virtual machine, where it must interact through mouse and keyboard actions; they are more realistic but substantially harder to evaluate.

OS-World. OS-World [44] is an online benchmark providing a scalable virtual-machine infrastructure across operating systems. It includes 369 open-ended tasks derived from real-world use cases—spanning web browsing, office software, file operations, coding, and multi-application workflows—each equipped with an initial state configuration and an execution-based evaluation script. Agents interact through raw mouse and keyboard actions, making OS-World the most realistic evaluation setting used in this thesis.

1.3 Related Work

Closed-Source Computer-Use Models Several commercial systems incorporate GUI grounding and computer-use capabilities, including GPT-4o with vision, Operator by OpenAI, Claude 4.5 Sonnet by Anthropic, and Gemini 3 Pro by Google. These systems remain entirely closed-source: training procedures, datasets, model architectures, and implementation details are undisclosed, precluding detailed comparison or reproducibility.

General-Purpose Vision-Language Models Recent open-weights vision-language models such as Qwen2.5-VL and Qwen3-VL [45, 46] incorporate domain-specific data targeting GUI interaction and computer-use scenarios alongside standard multimodal pretraining corpora. In addition to image-text, OCR, and general grounding data, these models include screenshot-based supervision, multi-step agent trajectories, and function-calling data, with evaluations reported on agentic and GUI-centric benchmarks. Despite this increasing focus on computer-use capabilities, these models remain general-purpose multimodal foundation models rather than task-specialized agents, and the precise training configuration for domain-specific components is described only at a high level.

End-to-End GUI Agents Building on Qwen-based vision-language backbones, the UI-TARS series [26, 27, 28] shifts toward explicitly training end-to-end GUI agents. UI-TARS applies continual training from Qwen2-VL 7B and 72B on approximately 50B tokens, incorporating large-scale GUI screenshot perception data, grounding annotations, action traces, and 6M filtered GUI tutorials to support System-2 reasoning and multi-step execution. UI-TARS-2 further scales this paradigm through a data flywheel combining continual pretraining, supervised fine-tuning, and multi-turn reinforcement learning in a unified GUI sandbox environment, enabling large-scale interactive rollouts across thousands of VMs. Empirically, UI-TARS-2 reports strong performance on GUI-centric benchmarks, achieving 94.2% on ScreenSpot-V2, 61.6% on ScreenSpot-Pro, and 47.5% on OS-World. While these works demonstrate substantial progress toward end-to-end computer-use agents, the large-scale trajectory data, annotation pipelines, and reinforcement learning infrastructure underlying these results remain closed-source, limiting full reproducibility and controlled comparison.

Specialized GUI Grounding Models Several recent works share our focus on specialized GUI grounding through data-centric approaches. JEDI [40] and SeeClick [38] investigate data collection and filtering methods and open-source their datasets. However, these works rely on continual pretraining through supervised fine-tuning alone, whereas our work incorporates reinforcement learning to directly optimize for grounding accuracy.

Two works are particularly closely related to our approach in both methodology and philosophy: SE-GUI [30] and GTA1 [29]. SE-GUI trains a GUI grounding agent using Group Relative Policy Optimization (GRPO) with a dense spatial reward that encourages predicted click points to lie inside the ground-truth bounding box. Like our work, SE-GUI emphasizes data quality and difficulty filtering, curating a high-quality dataset by removing noisy annotations and overly simple samples that the base model already solves reliably. Their data filtering strategy directly inspired aspects of our own filtering pipeline.

GTA1 [29] presents a two-stage GUI agent that significantly advanced grounding performance. Similar to our approach, the authors adopt a reinforcement learning framework based on GRPO, directly rewarding successful clicks when predicted coordinates fall inside the ground-truth bounding box. They introduce a data-cleaning strategy that filters noisy annotations via an IoU-based consistency check using an external UI element detector, thereby improving supervision quality. This explicit attention to data quality through model-based filtering informed our own use of OmniParser for annotation validation. Both SE-GUI and GTA1 demonstrate that combining reinforcement learning with careful data curation yields substantial improvements over pure supervised fine-tuning, a finding that motivates our SFT-then-RL training approach and systematic filtering pipeline.

1.4 Contributions

This thesis investigates data-centric and training strategies for GUI grounding tasks. We unify multiple heterogeneous datasets into a single, standardized collection and conduct detailed ablations over different filtering methods. We furthermore explore ways of enriching the resulting dataset in underrepresented domains such as professional desktop applications. Finally, we study training dynamics ranging from supervised fine-tuning to reinforcement learning. The main contributions are as follows:

1. **Filtering pipeline.** We develop a comprehensive filtering pipeline that leverages a variety of pre-existing models to systematically curate a large, multi-source data pool into a high-quality training set.

2. **Supplemental data collection.** We introduce a pipeline for collecting and annotating screen frames extracted from GUI tutorial videos, enabling the acquisition of training data for underrepresented domains that lack access to rich APIs typically available on the web.

3. **Reinforcement learning training recipe.** We develop a reinforcement learning training recipe tailored to GUI grounding tasks and study the effects of task difficulty and reward function design on model performance.

4. **State-of-the-art results.** We demonstrate that the resulting model, Gelato-30B-A3B, achieves 63.88% accuracy on ScreenSpot-Pro and 69.15% / 74.65% on OS-World-G / OS-World-G (Refined), surpassing prior specialized grounding models such as GTA1-32B and much larger vision-language models including Qwen3-VL-235B-A22B-Instruct.

5. **Open source.** We publicly release the Gelato-30B-A3B model, the Click-100k dataset, and the accompanying code to foster further research and ensure reproducibility.

1.5 Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2 – Methodology Presents the data pool, followed by a data filtering pipeline, the collection of supplemental data to address gaps in the dataset, and the training of Gelato-30B-A3B.

Chapter 3 – Results Provides benchmarking performance on offline grounding benchmarks, as well as agentic performance on the OS-World benchmark.

Chapter 4 – Conclusion & Future Work Summarizes the main contributions, discusses limitations of the current approach, and outlines promising future directions.

Chapter 2

Methodology

2.1 Data Pool

2.1.1 Data Sources

We curate our training data from existing datasets for web and desktop GUI grounding. Each dataset sample contains a screenshot image, a natural language instruction describing the desired interaction, and ground truth bounding box coordinates for the target UI element. Our data pool draws from eight complementary sources: ShowUI [33], AutoGUI [32], PC-Agent-E [36], WaveUI [35], OS-Atlas [34], UGround [31], PixMo [37], and SeeClick [38].

Normalization Since the collected datasets differ substantially in schema and annotation procedure, we apply a normalization step to consolidate them into a unified format. First, we partition samples by platform, discarding mobile samples. Second, because the source datasets span a spectrum from full interaction trajectories to isolated point annotations and encompass diverse action types (e.g., clicks, drags, text input), we extract and retain only click-action annotations pertinent to our grounding objective, discarding all other action types. The resulting annotation schema thus reduces to a triple of (image, instruction, bounding box). Table 2.1 summarizes the resulting scale and composition of the normalized data pool, which comprises approximately 9.8 million training samples. Representative examples from six sources are shown in Figure A.1 in the Appendix.

Table 2.1: Dataset Statistics

Dataset	Number of Samples	Number of Tokens (M)
AutoGUI	701,861	52.22
PC-Agent-E	27,782	42.09
WaveUI	24,977	1.74
SeeClick	27,193	1.80
OS-ATLAS	61,534	4.12
UGround	8,290,455	618.48
ShowUI-Web	598,856	40.48
ShowUI-Desktop	7,496	0.48
PixMo Points	92,477	5.81
Total	**9,832,631**	**767.22**

2.1.2 Additional Processing

Two datasets require additional preprocessing to extract suitable training samples. For PC-Agent-E, we extract individual click actions from recorded computer-use trajectories and generate corresponding natural language instructions by

summarizing each action's reasoning chain with Claude 3.7 Sonnet. Since PixMo Points is a general-purpose grounding dataset not specifically designed for computer-use tasks, we employ Qwen2.5-7B-VL as a classifier to identify and retain only samples depicting valid computer screen images.

Resizing Since our work, as well as previous efforts, builds on top of the Qwen2.5-VL [45] and Qwen3-VL [46] model families, careful attention to architecture-specific preprocessing is required. Both models employ flexible patching mechanisms in which input images are dynamically resized to dimensions that are multiples of 28 pixels before being processed by the Vision Transformer (ViT). Failing to account for this resizing step can introduce coordinate misalignment between model predictions and ground truth annotations. To prevent such artifacts, we standardize all training and evaluation images by resizing them to the nearest valid dimensions prior to model input.

Resolution To determine an appropriate image resolution for training, we conduct a preliminary study on the sensitivity of grounding performance to input resolution. Specifically, we evaluate GTA1 [29] on two offline benchmarks—ScreenSpot Pro and ScreenSpot V2—while varying the maximum pixel budget from 1 MP to 12 MP. As shown in Figure 2.1, the two benchmarks exhibit markedly different resolution dependencies. ScreenSpot Pro, which targets fine-grained elements in professional applications, benefits substantially from higher resolution: accuracy nearly doubles from 25.2% at 1 MP to a peak of 49.7% at 4 MP, before exhibiting diminishing returns at larger budgets. In contrast, ScreenSpot V2 already achieves strong performance at 1 MP (89.5%) and peaks at 2 MP (91.8%), with marginal degradation at higher resolutions. Based on these findings, we set the maximum resolution to 4 MP for all training data, as this provides the best trade-off between grounding accuracy and computational cost.

Coordinate System A further architectural distinction concerns coordinate representation. Qwen2.5-VL operates with absolute pixel coordinates, whereas Qwen3-VL adopts a normalized coordinate system scaled to the range $[0, 1000]$, which is reported to improve robustness to variations in image resolution and aspect ratio while simplifying post-processing. To maintain consistency with each model's native representation, we adapt the data preparation accordingly: absolute coordinates for Qwen2.5-VL experiments—facilitating direct comparison with other models built on the same backbone—and normalized coordinates for Qwen3-VL experiments.

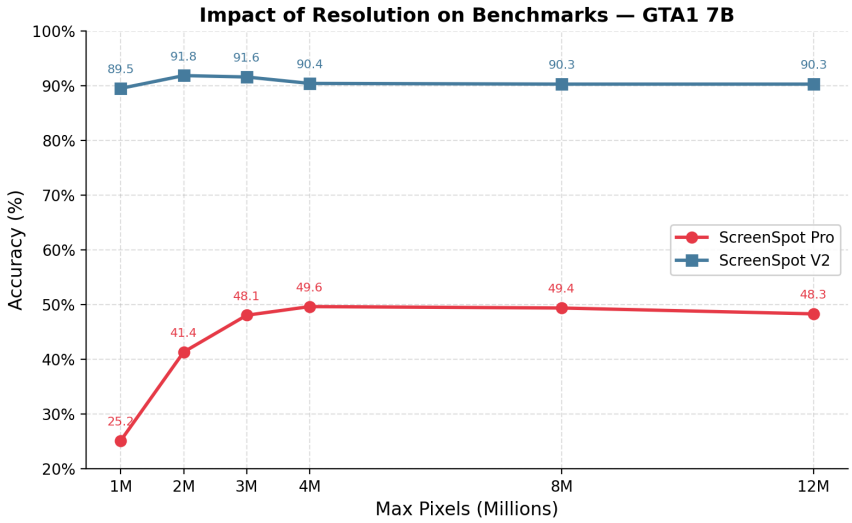


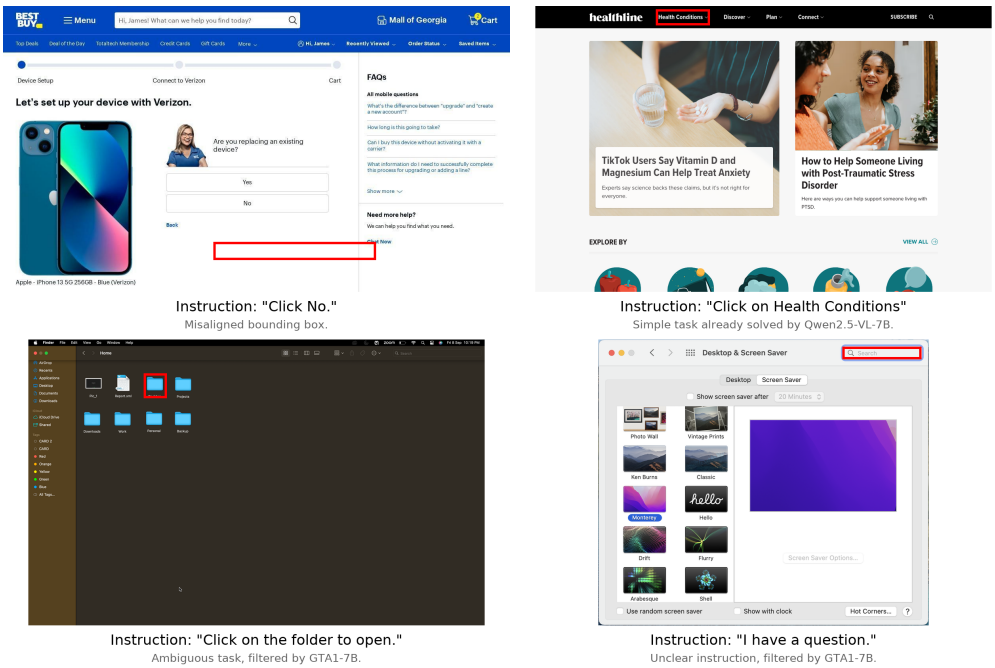
Figure 2.1: Impact of maximum pixel budget on grounding accuracy for GTA1-7B. ScreenSpot Pro shows a strong positive correlation with resolution up to 4 MP, while ScreenSpot V2 is largely resolution-invariant.

2.1.3 Data Quality Issues

After assembling the data pool, we conduct a manual quality assessment by inspecting samples alongside their instructions and ground truth bounding boxes. As illustrated in Figure 2.2, we identify three prevalent quality issues:

- **Overly simple interactions**, such as "Click on Health Conditions", which can be trivially resolved through optical character recognition alone without requiring deeper semantic understanding of the GUI layout.
- **Misaligned annotations**, where the instruction text and target region diverge due to annotation errors in the source datasets.
- **Ambiguous tasks** that lack sufficient context for precise grounding or have multiple possible target regions.

We address these quality issues through a systematic filtering pipeline, described in Section 2.2.



Instruction: "Click No."
Misaligned bounding box.

Instruction: "Click on Health Conditions"
Simple task already solved by Qwen2.5-VL-7B.

Instruction: "Click on the folder to open."
Ambiguous task, filtered by GTA1-7B.

Instruction: "I have a question."
Unclear instruction, filtered by GTA1-7B.

Figure 2.2: Examples of data quality issues encountered in curated datasets and their resolution through our filtering pipeline. Top left: misaligned bounding box; top right: simple task successfully filtered; bottom left: ambiguous task filtered by GTA1-7B; bottom right: unclear instruction filtered by GTA1-7B.

2.2 Data Filtering

2.2.1 Iterative Evaluation

Developing an effective data recipe requires rapid empirical iteration. To enable this, we make two design choices that prioritize evaluation speed. First, we employ offline benchmarks—ScreenSpot-Pro, ScreenSpot V2, and OS-World—which can be evaluated in minutes, in contrast to online agentic benchmarks that require provisioning virtual machines and executing multi-step trajectories. Second, we use supervised fine-tuning (SFT) on Qwen2.5-VL rather than reinforcement learning, as SFT provides substantially faster training cycles. We operate under the assumption that data recipes that improve offline benchmark performance under SFT will transfer to online agentic settings and to RL-based training.

2.2.2 Construction of Filtering Pipeline

Balanced Weighting of Data Sources To ensure that the filtering pipeline is not biased towards any particular data source, we balance the data pool by sampling up to 50k samples from each source (or the maximum available if fewer than 50k exist).

OmniParser Filtering To address the misaligned bounding box annotations identified in Section 2.1, we adopt a vision-based validation approach inspired by the IoU overlap strategy of [29]. Specifically, we employ OmniParser [47], a screen parsing system built on a YOLO-based detection model that identifies interactable UI elements by producing bounding boxes directly from screenshot pixels, without requiring access to underlying code structures or accessibility trees (Figure 2.3). This purely visual approach enables robust parsing across diverse GUI environments—including web browsers and desktop applications—where traditional DOM-based methods are unavailable or unreliable.

We use OmniParser's detected bounding boxes to validate the spatial alignment between ground truth annotations and actual UI elements. Concretely, we discard any training sample whose ground truth click location falls outside all detected element bounding boxes, as illustrated in Figure 2.2. This filtering step removes misaligned annotations that would otherwise introduce noise during training. Because the method operates purely on visual features, it generalizes across application types and platforms, making it particularly valuable for desktop-centric datasets where structured metadata is scarce.

Difficulty Filtering Our difficulty filtering is inspired by [30], who filter out overly easy samples. However, their approach does not address overly difficult

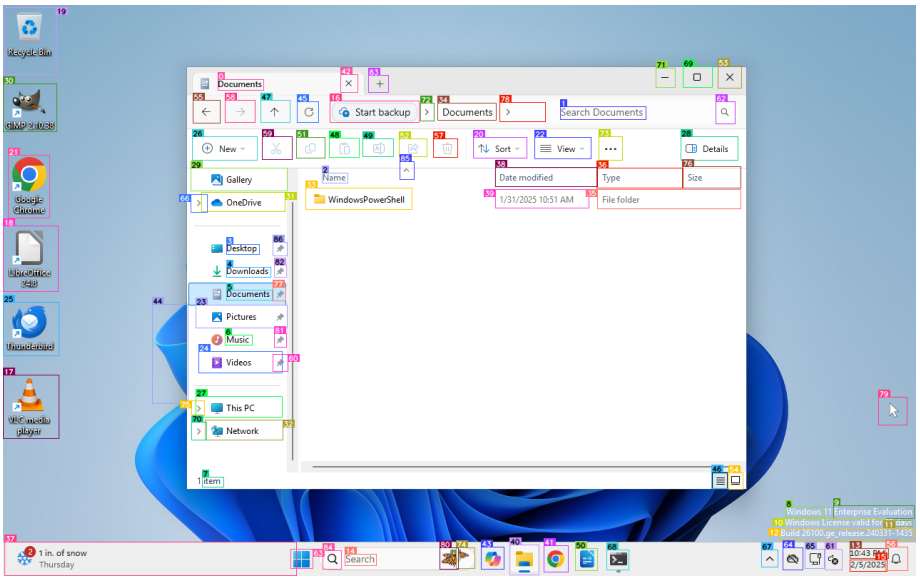


Figure 2.3: OmniParser filtering pipeline. The system detects UI elements and their bounding boxes (shown as colored rectangles), enabling validation of ground truth annotations. Samples where the target click location falls outside all detected elements are filtered out as misaligned.

samples. We observe that ambiguous and unclear instructions—as identified among the data quality issues in Section 2.1—are not captured by OmniParser alignment or easy-sample filtering alone, motivating an additional hard-sample filtering stage.

To filter samples by difficulty, we use either Qwen2.5-VL-7B or SE-3B [30] for easy sample filtering, and GTA1-7B [29], UI-7B [48], or SE-3B for hard sample filtering. Together with the OmniParser bounding box alignment step, these components form the complete filtering pipeline illustrated in Figure 2.4. To validate this approach, we conduct detailed ablations by fine-tuning Qwen2.5-VL-7B on a source-balanced 10k subset and evaluating on ScreenSpot-Pro. We find that using Qwen2.5-VL-7B as the easy-sample filter and GTA1-7B as the hard-sample filter produces a +9 percentage point accuracy gain over unfiltered data and outperforms all other filtering model combinations. Furthermore, we find strong empirical evidence that hard-sample filtering is beneficial: adding GTA1-7B filtering on top of Qwen2.5-VL-7B filtering alone yields an additional +5.5 percentage point gain.

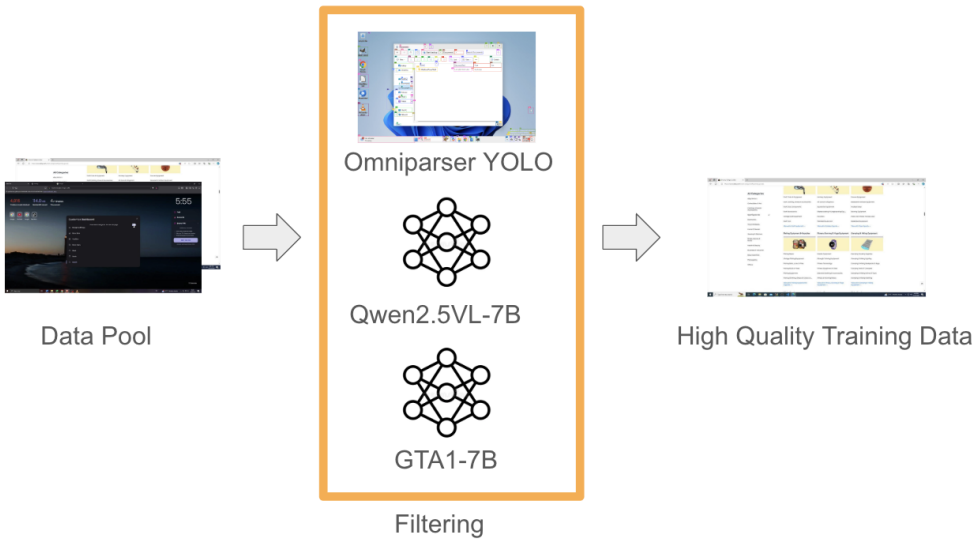
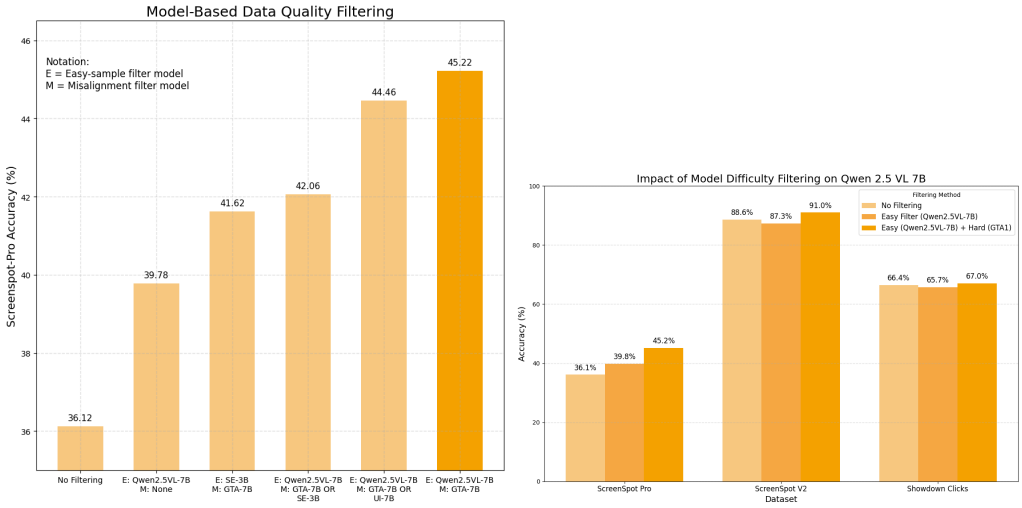


Figure 2.4: Overview of the complete data filtering pipeline. The raw data pool is processed through three filtering stages: OmniParser YOLO for bbox alignment filtering (removing samples where the ground truth does not overlap with any detected UI element), Qwen2.5VL-7B for easy sample filtering, and GTA1-7B for hard sample filtering. Only samples that pass all three stages are retained as high-quality training data.



While it might seem that filtering out hard samples with a stronger model like GTA1-7B would prevent the trained model from surpassing it, this is—counterintuitively—not the case. Previous work has shown that filtering a data pool using a weaker CLIP model can produce a stronger CLIP model when trained on the filtered data [49]. One plausible explanation is that removing noisy or ambiguous samples enables the model to learn cleaner representations.

2.2.3 Data Source Analysis

In the previous filtering pipeline, we trained on a balanced set from all data sources. However, we are interested in identifying the performance of individual data sources. To investigate this, we trained Qwen 2.5 VL 7B on each data source separately, capping the number of samples at 4.9k to control for the confounding effect of data repetition.

Data Source	SS Pro Accuracy	SS V2 Accuracy	OS-World G Accuracy
ShowUI-Web	36.43%	86.77%	48.24%
AutoGUI	34.54%	87.68%	47.06%
PC-Agent-E	34.28%	87.29%	47.84%
WaveUI	33.40%	87.16%	44.71%
Omniact	33.21%	87.16%	44.90%
ShowUI-Desktop	32.01%	85.21%	42.16%
UGround	31.82%	85.99%	41.76%
Pixmo Points	30.68%	86.64%	42.16%
SeeClick	29.22%	84.82%	43.53%

Table 2.2: Performance of individual data sources on downstream benchmarks. Each data source was trained separately on Qwen 2.5 VL 7B with 4.9k samples.

Table 2.2 reveals substantial variation in data source quality across benchmarks. The top three performing data sources for ScreenSpot Pro are ShowUI-Web (36.43%), AutoGUI (34.54%), and PC-Agent-E (34.28%). For ScreenSpot V2, AutoGUI (87.68%), PC-Agent-E (87.29%), and Omniact/WaveUI (87.16%) achieve the highest accuracies. For OS-World G, ShowUI-Web (48.24%), PC-Agent-E (47.84%), and AutoGUI (47.06%) demonstrate superior performance.

Overall, ShowUI-Web exhibits consistently strong performance across all benchmarks, particularly excelling on ScreenSpot Pro and OS-World G. PC-Agent-E demonstrates robust and balanced performance across all evaluation metrics, while AutoGUI achieves the highest ScreenSpot V2 accuracy alongside strong ScreenSpot Pro performance.

Conversely, SeeClick and Pixmo Points represent the poorest performing data sources, achieving the lowest (29.22%) and second-lowest (30.68%) ScreenSpot Pro accuracies, respectively.

Based on these data source experiments, we investigated the effect of removing the poorest performing data sources from the training pool. Specifically, we removed the bottom three data sources ranked by ScreenSpot Pro performance: SeeClick, Pixmo Points, and UGround. Subsequently, we trained the model on the remaining data sources in our filtered data pool sampling 10k samples.

As shown in Table 2.3, removing the bottom three data sources at this stage

Data Pool	SS Pro Accuracy	SS V2 Accuracy
All Data Sources	45.22%	91.05%
Remove Bottom 3 on SS Pro (SeeClick, Pixmo Points, UGround)	45.03%	90.14%
Remove Worst one on SS Pro (Pixmo Points)	44.78%	90.79%

Table 2.3: Impact of removing poorest performing data sources on downstream task performance at equal data scale.

does not yield consistent improvements across benchmarks. ScreenSpot Pro and ScreenSpot V2 accuracies decrease marginally. Given these mixed results, we conclude that it is beneficial to retain all data sources in the training pool to maximize overall performance. However, during RL training we find that these data sources contain problematic ambiguous annotations and ultimately exclude them from the RL data pool (Section 2.4.3).

2.2.4 Data Sampling and Scale

To investigate the impact of training set size on model performance, we conduct a series of ablation studies with varying data scales. From our filtered pool, we sample data from each source at different scales (10k, 20k, 35k, and 80k samples). As shown in Figure 2.5, we observe consistent performance improvements with increased data scale on ScreenSpot Pro, with accuracy improving from 45.22% at 10k samples to 49.65% at 80k samples when combined with enhanced prompting and additional in-house data. Performance on ScreenSpot V2 remains relatively stable across different scales, suggesting that model capacity and data quality play complementary roles in determining final performance.

2.2.5 Additional Experiments

We conducted several additional experiments to optimize the training process, including training prompts, instruction rewriting, vision tower configuration, and learning rate tuning. Our hyperparameter search revealed that a learning rate of 1e-6 yields optimal performance, and that full fine-tuning (including the vision tower) significantly outperforms freezing the vision tower. We also found that the default training prompt is sufficient and that verbose tool-calling prompts do not improve performance. We experimented with rewriting existing instructions using LLM-based approaches (Qwen3-4B) as well as image-aware synthetic prompts (Qwen2.5-VL-7B) to improve instruction quality; however, this did not improve performance. Detailed results and ablations for these experiments are provided in the Appendix (see Tables A.3, A.2, A.4, and A.5).

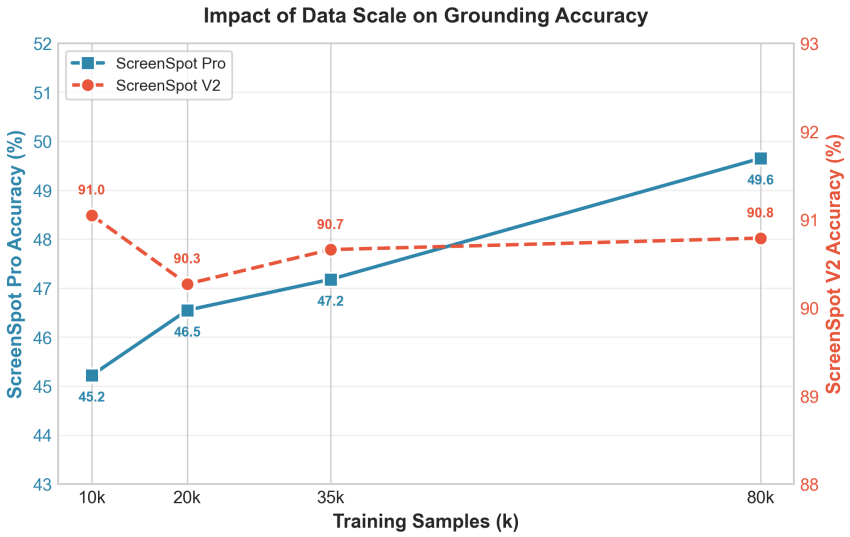


Figure 2.5: Impact of training data scale on grounding accuracy. ScreenSpot Pro (left axis) shows consistent improvement with increasing data, while ScreenSpot V2 (right axis) remains relatively stable across scales.

2.3 Supplemental Data

After assembling the data pool (Section 2.1) and filtering it (Section 2.2), we evaluate performance and identify systematic weaknesses to locate underrepresented areas in the training data. We specifically focus on the ScreenSpot Pro benchmark, on which even closed-source models achieve poor performance. ScreenSpot Pro is dominated by professional applications, a domain that poses a significant gap for many GUI models because data collection is substantially more difficult due to the absence of structured APIs and the domain expertise required for annotation. Furthermore, the benchmark contains many high-resolution images, multi-window layouts, and complex screen configurations.

2.3.1 Performance Analysis by Image Resolution and Aspect Ratio

To better understand the characteristics of challenging samples in our evaluation set, we analyze model performance across different image resolutions and aspect ratios. We evaluate the 20k sample (without replacement) trained model on ScreenSpot Pro, stratifying performance by megapixel count and aspect ratio.

The analysis reveals three distinct observations:

- The most common aspect ratio is 16:9 (approximately 1.78), and model performance decreases as we move toward ultra-wide aspect ratios (3.6).
- Performance declines with increasing image size, which we attribute to the resizing of images to a maximum of 4 MP to control prompt length—this downsampling may remove fine-grained visual details necessary for accurate grounding.
- Surprisingly, model performance is also weak at low resolutions (2–3 MP).

To mitigate these weaknesses, we explore data augmentation strategies.

2.3.2 Data Augmentation

We investigate two data augmentation strategies aimed at improving model performance on high-resolution screenshots, as evaluated on the ScreenSpot-Pro benchmark.

Composing High-Resolution Frames. To expose the model to high-resolution inputs during training, we experiment with synthetically composing dual-screen and large desktop montages. Specifically, we construct dual-screen samples with

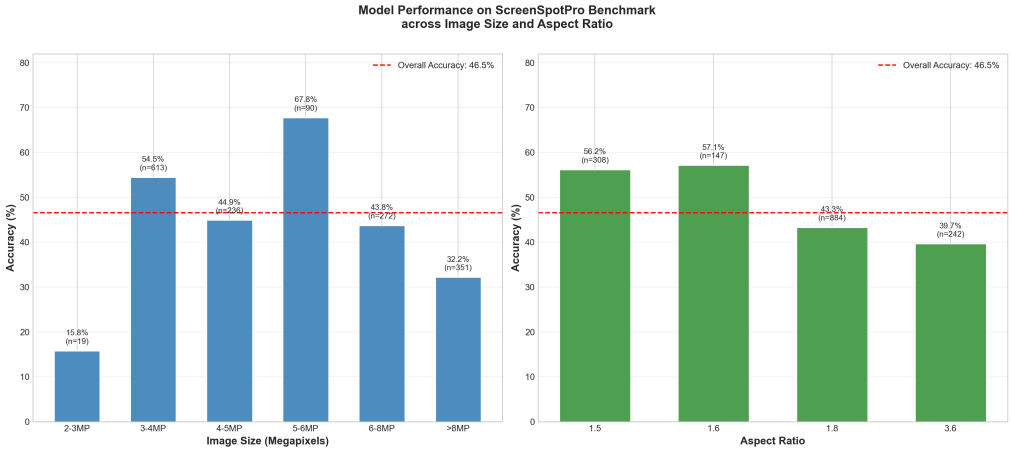


Figure 2.6: Model performance on the ScreenSpot Pro benchmark stratified by image resolution (left) and aspect ratio (right). The dashed red line indicates overall accuracy (46.5%). Bar labels show per-category accuracy and sample count (n). The model achieves 736 correct predictions out of 1,581 total samples.

high aspect ratio by concatenating pairs of randomly selected frames from different data sources, comprising around 20% of the augmented dataset. Additionally, we overlay random frames onto large desktop background images to simulate multi-window desktop environments. In this initial experiment, no verification is performed to ensure that instructions from different constituent frames do not spatially collide. As shown in Table 2.4, this naive composition strategy leads to a substantial degradation in ScreenSpot-Pro accuracy, decreasing from 45.22% to 36.94%.

Table 2.4: Effect of composing high-resolution frames on ScreenSpot-Pro accuracy.

Configuration	**SS-Pro Accuracy**
Baseline	45.22%
Composed High-Resolution Frames	36.94%

Random Image Upscaling. Motivated by findings from Phi-Ground [50], which reported an 8 percentage point improvement on ScreenSpot-Pro through random image upscaling during training, we evaluate a similar strategy. We randomly resize training images up to a maximum resolution of 4 megapixels. However, as shown in Table 2.5, this approach does not yield improvements in our setting: ScreenSpot-Pro accuracy decreases slightly from 45.22% to 44.14%, while ScreenSpot V2 accuracy drops more notably from 91.05% to 88.45%.

These results suggest that naive augmentation strategies for high-resolution

Table 2.5: Effect of random image upscaling on benchmark accuracy.

Configuration	SS-Pro Accuracy	SS V2 Accuracy
Baseline	45.22%	91.05%
Random resize up to 4MP	44.14%	88.45%

inputs—whether through frame composition or random upscaling—do not transfer effectively to our training setup and may introduce noise that degrades grounding performance.

2.3.3 Professional Application Data

UI Vision and JEDI To improve robustness on professional desktop applications which are underrepresented in comparison to web data, we incorporate UI-Vision [39] as additional training data. We do not use UI-Vision for evaluation, as it is not yet widely adopted as a standard benchmark for reporting final model performance. Instead, we treat it purely as supplementary supervision to increase coverage of desktop UI elements and interaction patterns. UI-Vision provides bounding boxes for UI elements, semantic labels, and interaction-related metadata across a diverse set of professional and productivity software. Compared to web-based GUI datasets, UI-Vision focuses on complex desktop environments where structured APIs are typically unavailable and manual annotation requires domain knowledge. This makes it particularly valuable for improving model exposure to professional application layouts and visual element grounding. The JEDI dataset [40] includes large-scale grounding data sourced not only from web interfaces but also from real-world desktop and professional applications, incorporating screenshots and structured metadata from production software (e.g., office tools, system apps) and agent rollouts in environments like WindowsAgentArena. We normalize, resize, and convert the UI-Vision [39] and JEDI [40] datasets to the same format as the other datasets. Importantly, we do not apply the full filtering pipeline to these datasets, as the prior models used for difficulty filtering do not perform well in this domain either—applying such filtering would likely discard the majority of the collected data. After preprocessing, we obtain 5,733 samples from UI-Vision and 18,032 samples from JEDI.

To evaluate the impact of these supplemental sources, we fine-tune the model by incrementally adding UI-Vision and JEDI samples to the existing 38k-sample training set and compare against the baseline. Results are reported in Table 2.6.

Adding UI-Vision alone initially decreases ScreenSpot Pro accuracy from 49.3% to 47.6%, while improving OS-World-G from 57.4% to 58.6%. As more JEDI data is incorporated, OS-World-G continues to improve, reaching 60.8% with 5k JEDI samples. With the full JEDI set (63k total samples), ScreenSpot

Table 2.6: Impact of supplemental data on model performance. Adding UI-Vision and JEDI progressively improves OS-World-G while recovering ScreenSpot Pro accuracy at sufficient data scale.

Model Configuration	SS Pro	OS-World-G
SFT-7B (38k)	49.3%	57.4%
SFT-7B (38k) - 2 epochs	50.16%	56.0%
SFT-7B (44k) + UI-Vision	47.6%	58.6%
SFT-7B (49k) + UI-Vision + 5k JEDI	47.9%	60.8%
SFT-7B (63k) + UI-Vision + all JEDI	50.09%	60.1%

Pro recovers to 50.09%—on par with the baseline—while OS-World-G remains substantially higher at 60.1%. Notably, simply training the baseline for a second epoch yields only marginal gains on ScreenSpot Pro (50.16%) and a decrease on OS-World-G (56.0%), indicating that the improvements from supplemental data are not attributable to increased training steps alone.

Video Data Collection We additionally construct an automated pipeline for collecting and annotating screen frames from GUI tutorial videos to supplement our training data. The pipeline, illustrated in Figure 2.8, comprises four stages: video acquisition, frame extraction, frame sampling, and instruction generation.

In the first stage, we manually curate a list of approximately 120 tutorial videos spanning over 80 professional applications, including 3ds Max, ANSYS, Adobe Creative Suite, Blender, MATLAB, and Vivado, among others (see Table A.1 in the Appendix for the full list). Videos are downloaded at their highest available quality.

In the second stage, we extract unique screen-like frames from each downloaded video using PySceneDetect. For each detected scene transition, the mid-frame is selected as a representative sample. We then apply a series of heuristic filters to remove non-screen content: face detection via OpenCV cascade classifiers excludes frames containing presenter overlays, while additional filters discard introductory and concluding segments, low-variance or solid-color frames, and grayscale frames. To eliminate near-duplicate content within each video, we perform perceptual deduplication using pHash.

In the third stage, we sample a fixed number of frames (default: 250) across a set of target professional applications from the extracted frame pool. The sampling procedure selects evenly spaced frames within each application to maximize temporal diversity.

In the fourth stage, we construct the final instruction dataset from the sampled frames. Natural language instructions are generated using Claude, producing structured instruction–target pairs for each screenshot (the full prompt is provided in Appendix A.3). Each training sample consists of a single screenshot

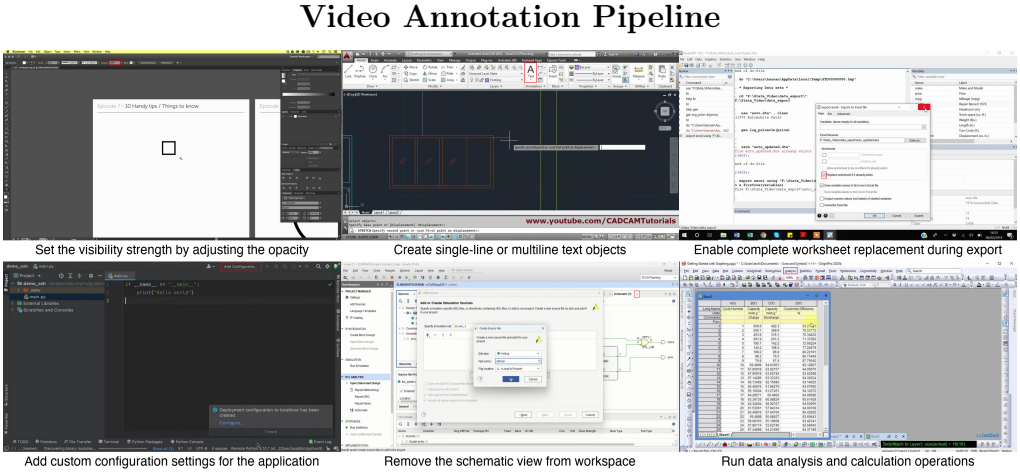


Figure 2.7: Examples of professional application screenshots collected via our video data pipeline, spanning Adobe Illustrator, AutoCAD, Stata, PyCharm, Vivado, and OriginPro. Each frame is paired with a natural language grounding instruction generated by Claude.

paired with two instructions corresponding to distinct UI segments.

We incorporate the in-house professional application data collected from YouTube into the model-difficulty-filtered training set and sample 10k samples in total. As shown in Table 2.7, this yields a modest improvement from 45.22% to 46.11% on ScreenSpot Pro.

Table 2.7: Impact of In-house Professional Application Data

Data Configuration	SS Pro
10k baseline	45.22%
10k + in-house prof. app data	46.11%

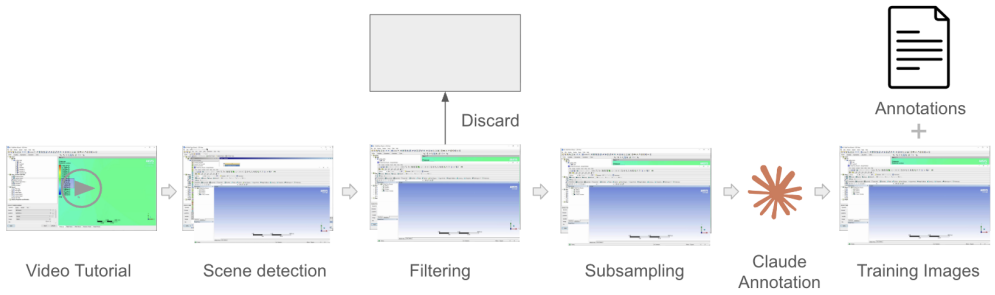


Figure 2.8: Video data collection and annotation pipeline. Tutorial videos are processed through scene detection to extract representative frames, which are then filtered to remove non-screen content (e.g. presenter overlays, intro/outro segments, solid-color frames). A fixed number of frames per application is subsampled for temporal diversity, and finally Claude generates natural language grounding instructions paired with each screen frame.

2.4 Training

After establishing our data recipe through supervised fine-tuning experiments, we turn to reinforcement learning (RL) to push model performance beyond the SFT plateau, similar to [29, 30]. We adopt Group Relative Policy Optimization (GRPO) [51] as our base RL algorithm and progressively refine the training pipeline through a series of experiments, ultimately incorporating key ideas from DAPO [52], leading to our final best-performing Gelato-30B-A3B model. Both GRPO and DAPO are implemented with the EasyR1 framework [53]. This section describes the training methodology; all experimental results are presented in Chapter 3.

2.4.1 Initial GRPO Setup

We optimize the policy using the Group Relative Policy Optimization (GRPO) objective. For each group of G rollouts, we normalize rewards to compute per-token advantages:

$$\hat{A}_{i,t} = \frac{R_i - \bar{R}}{\sigma_R}, \qquad \bar{R} = \frac{1}{G}\sum_{j=1}^{G} R_j, \qquad \sigma_R = \mathrm{std}\big(\{R_j\}_{j=1}^{G}\big). \tag{2.1}$$

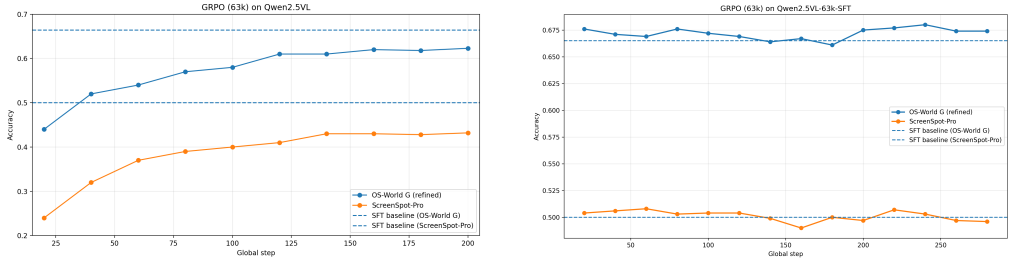
The GRPO objective is then:

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},\, \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}(\cdot|q)} \Bigg[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \min\Big(r_{i,t}(\theta)\,\hat{A}_{i,t},\ \mathrm{clip}\big(r_{i,t}(\theta),\, 1-\varepsilon_\ell,\, 1+\varepsilon_h\big)\,\hat{A}_{i,t}\Big) \Bigg], \tag{2.2}$$

where the per-token probability ratio is:

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid q,\, o_{i,<t})}{\pi_{\theta_{\mathrm{old}}}(o_{i,t} \mid q,\, o_{i,<t})}. \tag{2.3}$$

RL vs SFT Our first RL experiment applied GRPO to both the Qwen2.5-VL-7B-Instruct base model and a model already trained with one round of SFT on the training data to see if we could improve performance beyond the SFT baseline. We used 8 rollouts per prompt at temperature 1.1 with KL divergence disabled, training on a pool of approximately 63k samples. As shown in Figure 2.9, RL did not boost performance beyond the SFT baseline in either setting: ScreenSpot-Pro accuracy remained flat around the SFT level, while OS-World-G (refined) showed only marginal fluctuation. We later addressed this issue by implementing DAPO [52]-style dynamic sampling and asymmetric clipping.



(a) GRPO on base Qwen2.5-VL-7B-Instruct (RL only, no SFT).

(b) GRPO on Qwen2.5-VL-7B after 63k SFT (SFT + RL).

Figure 2.9: Initial GRPO experiments. Validation accuracy on ScreenSpot-Pro (orange) and OS-World-G refined (blue) over training steps. Dashed lines indicate the SFT baseline. In both settings, RL training fails to improve over the SFT checkpoint, motivating the subsequent improvements to the training pipeline.

2.4.2 Reward Function

Unlike mathematical reasoning tasks where reward is binary (correct/incorrect), GUI grounding allows for a natural continuous reward signal based on spatial proximity. Both our reward functions are based on the Euclidean distance $d = \|(x, y) - (c_x, c_y)\|$ between the predicted point (x, y) and the center (c_x, c_y) of the ground truth region R; they differ only in the normalization constant and in whether a signal is provided outside R.

We implement a *sparse* reward function that assigns non-zero reward only when the predicted coordinate falls inside the ground truth region:

$$r_{\text{sparse}}(x, y) = \begin{cases} 1 - \dfrac{d}{d_R} & \text{if } (x, y) \in R, \\ 0 & \text{otherwise,} \end{cases}$$

where d_R is the maximum distance from the center to any corner of R (i.e. the half-diagonal of the bounding box). This reward is maximal (1.0) when the prediction coincides with the region center and decreases smoothly toward zero at the corners. We do not include a format reward, as the model's output format (a coordinate pair) is simple and reliably learned during SFT.

Sparse vs. Dense Reward Function. Inspired by [30], who use a dense reward function, we ablate our sparse reward against a dense variant. The *dense* reward provides a distance-based signal over the entire viewport, rather than only inside the ground truth region. The motivation is that for hard samples with small bounding boxes, the sparse reward assigns zero reward to predictions

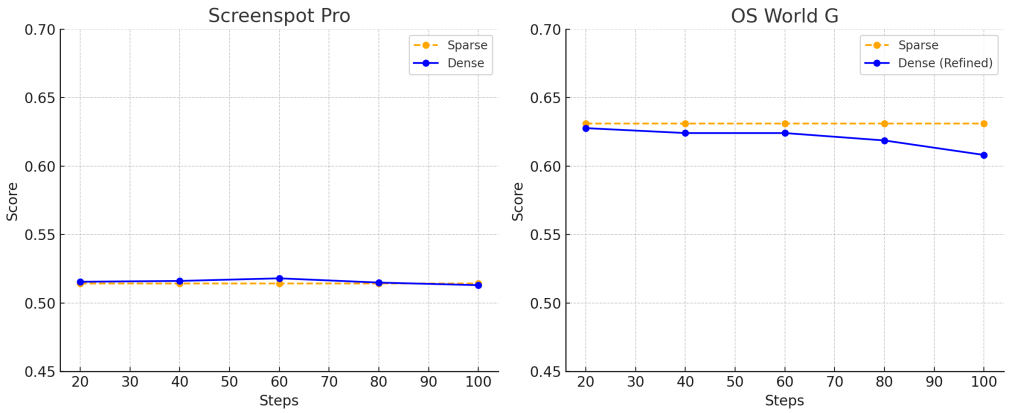


Figure 2.10: Comparison of sparse and dense reward functions in GRPO training. **Left:** ScreenSpot-Pro accuracy. **Right:** OS-World-G accuracy. The sparse reward (orange, dashed) performs comparably or slightly better than the dense variant (blue, solid), despite providing no signal outside the ground truth region.

that are close but just outside the region boundary, providing no gradient signal to improve.

Using the same distance d, the dense reward is defined as:

$$r_{\text{dense}}(x, y) = \begin{cases} 1.0 + \left(1 - \dfrac{d}{D_v}\right)^2 & \text{if } (x, y) \in R, \\ \left(1 - \dfrac{d}{D_v}\right)^2 & \text{otherwise,} \end{cases}$$

where $D_v = \sqrt{W^2 + H^2}$ is the viewport diagonal. Predictions inside R receive a base reward of 1.0 plus a smooth bonus that peaks at the box center, while predictions outside R still receive a continuous signal that decays smoothly with distance. This ensures that near-misses receive positive reward rather than zero, providing useful gradient signal even for hard samples. In contrast, the sparse reward provides no learning signal whatsoever for predictions outside R, regardless of proximity.

We compare the two reward functions under identical training conditions, starting from Qwen2.5-VL-7B-Instruct after one round of SFT and performing RL on the same training data with the GRPO algorithm (Figure 2.10). Despite the richer gradient signal, the dense reward does not consistently outperform the sparse variant: performance is comparable on ScreenSpot-Pro and slightly worse on OS-World-G. We therefore retain the sparse reward for all subsequent experiments.

2.4.3 DAPO: Dynamic Sampling and Asymmetric Clipping

In GRPO, if all outputs $\{o_i\}_{i=1}^{G}$ of a particular prompt are correct and receive the same reward, the resulting advantage for this group is zero. A zero advantage yields zero policy gradients, shrinking the effective gradient magnitude and increasing the noise sensitivity of the batch gradient, thereby degrading sample efficiency.

Let the intended batch size be B and the effective batch size be B_{eff}. Then

$$B_{\text{eff}} = \sum_{i=1}^{B} \mathbf{1}\{A_i \neq 0\}.$$

Under the i.i.d. assumption,

$$B_{\text{eff}} \sim \text{Binomial}(B, 1-p),$$

where p is the probability that all rollouts for a prompt yield the same reward. The expected effective batch size is

$$\mathbb{E}[B_{\text{eff}}] = B(1-p),$$

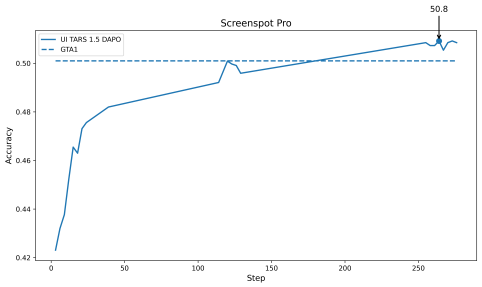
with variance

$$\text{Var}(B_{\text{eff}}) = B(1-p)p.$$

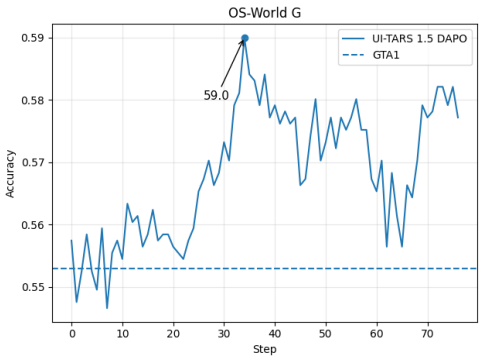
One can interpret the filtering done in Section 2.2 as a way to mitigate this problem before starting RL training by decreasing the probability p. However, as RL progresses and the model gets better, certain prompts are solved with high probability and the probability p increases again. To mitigate this problem during online training, we implement DAPO-style dynamic sampling and asymmetric clipping.

We incorporate key techniques from DAPO [52]: (1) removing the KL-divergence term, (2) clipping the probability ratio asymmetrically, and (3) skipping zero-advantage rollouts via dynamic sampling.

Dynamic Sampling. Rather than relying solely on our a priori filtering pipeline (Section 2.2), DAPO performs online oversampling during training. For each batch, more prompts are sampled than needed, rollouts are generated, and prompts where the mean reward falls below a threshold τ_{low} or above τ_{high} are discarded. We set $\tau_{\text{low}} = 0.01$ and $\tau_{\text{high}} = 0.5$, which ensures that each training batch contains only prompts with non-trivial advantage signal. This approach complements our a priori filtering by additionally removing prompts that become too easy during training.



(a) ScreenSpot-Pro accuracy over training steps.



(b) OS-World-G accuracy over training steps.

Figure 2.11: DAPO training on UI-TARS-1.5-7B with the Click-100k dataset. Solid lines show our DAPO run; dashed lines indicate the GTA1-7B-2507 baseline. On both benchmarks, DAPO training steadily improves accuracy and ultimately surpasses the GTA1 baseline.

Asymmetric Clipping. Standard GRPO clips the probability ratio symmetrically at $[1-\epsilon, 1+\epsilon]$. DAPO introduces an asymmetric upper clip $\epsilon_{\text{high}} > \epsilon_{\text{low}}$, which allows the policy to increase the probability of high-advantage actions more aggressively than it decreases the probability of low-advantage ones. We set $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$ following the DAPO recommendations.

Removing Ambiguous Data Sources in the RL Stage. While SeeClick, PixMo, and UGround were the weakest in earlier SFT experiments, excluding them did not yield consistent improvements. Yet in the RL stage we find that these data points have problematic ambiguous annotations that prohibit the model from solving them consistently. We therefore remove them from the RL training set. The remaining sources form the Click-100k RL training set.

2.4.4 Applying Data and Training Recipe

We apply the DAPO modifications on top of UI-TARS-1.5-7B [27]—a slightly stronger model trained from Qwen2.5-VL-7B-Instruct, and the same base model used to train GTA1-7B-2507 [29]—to enable a direct comparison between our data and training recipe and GTA1's.

Figure 2.12 summarizes the comparison between our data and training recipe and GTA1's. Starting from the same UI-TARS-1.5-7B base model (42.2% ScreenSpot-Pro, 52.8% OS-World-G), the GTA recipe improves performance to 50.1% and 55.3%, respectively. The Gelato recipe pushes further to 50.8% on ScreenSpot-Pro and 59.0% on OS-World-G, demonstrating that our curated data and DAPO

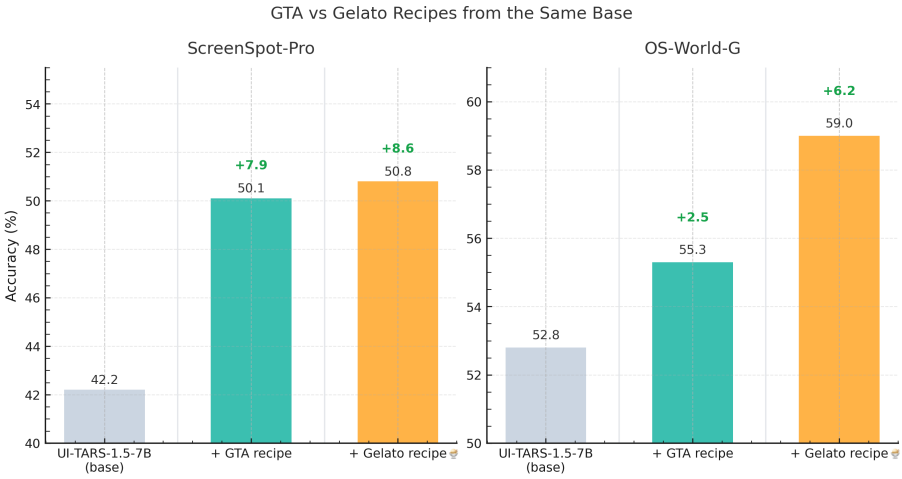


Figure 2.12: Comparison of training recipes applied to the same UI-TARS-1.5-7B base model. The Gelato recipe (DAPO on Click-100k) outperforms the GTA recipe on both ScreenSpot-Pro (+8.6 pp over base, +0.7 pp over GTA) and OS-World-G (+6.2 pp over base, +3.7 pp over GTA).

training pipeline yield consistent gains over GTA1's recipe on the same base model.

2.4.5 Scaling to Gelato-30B-A3B

Having established the Gelato recipe on 7B models, we scale to a larger base model: Qwen3-VL-30B-A3B-Instruct [46], a mixture-of-experts VLM with 30B total parameters and 3B active parameters per token. We initialize Gelato-30B-A3B from Qwen3-VL-30B-A3B-Instruct and train with the full DAPO setup (dynamic sampling, asymmetric clipping, no KL) on the Click-100k dataset for 100 steps on 32×40 GB A100 GPUs. Figure 2.13 shows the training dynamics. Despite starting from a general-purpose VLM—Qwen3-VL-30B-A3B-Instruct, as opposed to a computer-use model like UI-TARS-1.5—and only having 3B active parameters per token, Gelato-30B-A3B achieves strong performance on both ScreenSpot-Pro and OS-World-G, outperforming GTA1-32B and Qwen3-VL-235B-Instruct on both benchmarks.

Figure 3.1 places Gelato-30B-A3B in context against leading models. Despite using only 3B active parameters per token, Gelato-30B-A3B achieves **63.8%** on ScreenSpot-Pro and **69.1%** on OS-World-G, outperforming GTA1-32B, Qwen3-VL-235B-Instruct, and OpenCUA-72B on both benchmarks. Detailed per-category results are presented in Section 3.1.

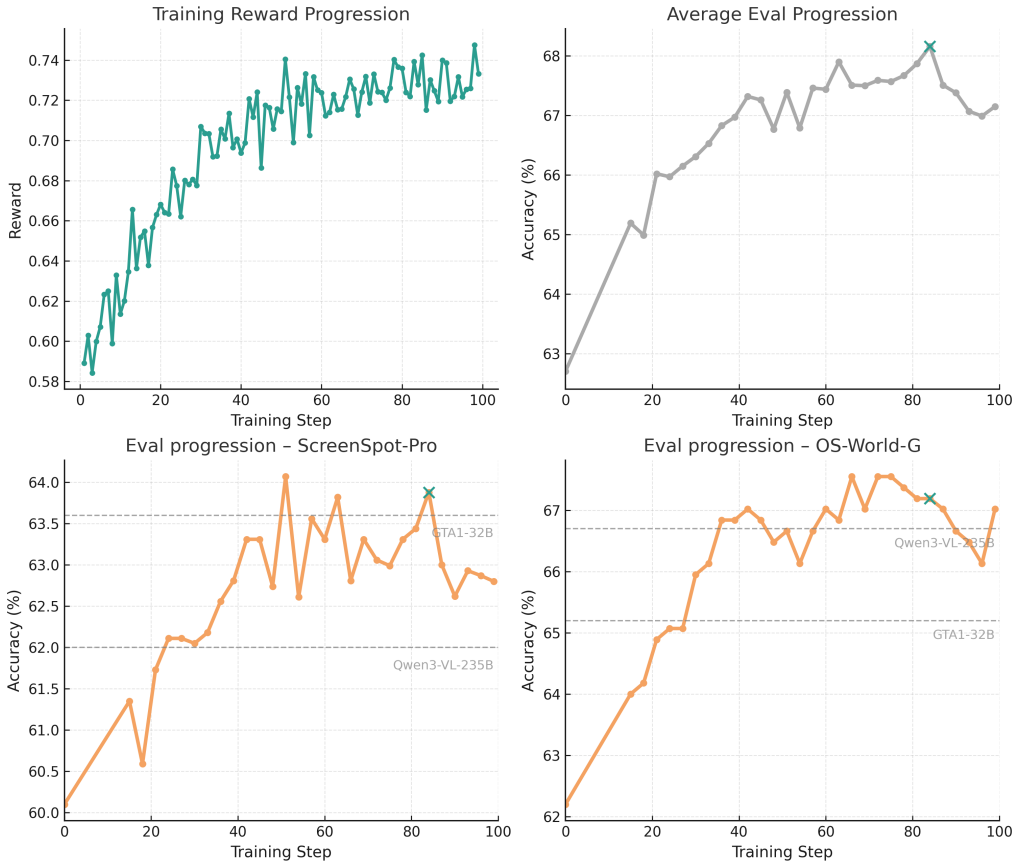


Figure 2.13: Gelato-30B-A3B training progression over 100 DAPO steps. **Top left:** training reward. **Top right:** average evaluation accuracy. **Bottom left:** ScreenSpot-Pro accuracy. **Bottom right:** OS-World-G accuracy. Dashed lines indicate GTA1-32B and Qwen3-VL-235B baselines. The green cross marks the best checkpoint at step 84.

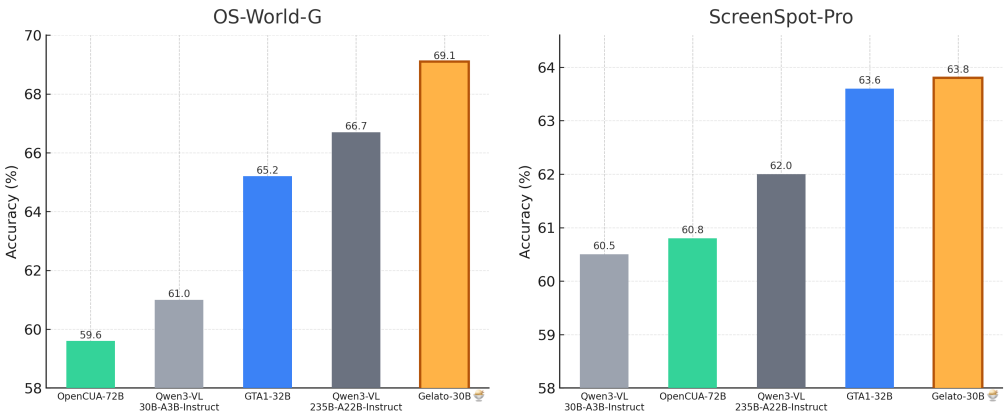


Figure 2.14: Final benchmark comparison of Gelato-30B-A3B against leading models. **Left:** OS-World-G accuracy. **Right:** ScreenSpot-Pro accuracy. Gelato-30B-A3B achieves the highest accuracy on both benchmarks despite having fewer active parameters than all competitors.

Chapter 3

Results

3.1 Benchmarking Performance

To evaluate the overall effectiveness of the Gelato training recipe, we compare it against the leading open-source model, GTA1 [29]. We initialize RL training from UI-TARS-1.5-7B [26] (as done for GTA1-7B-2507) and train on Click-100k until convergence (255 steps). Gelato outperforms GTA1-7B-2507 on both offline benchmarks ScreenSpot-Pro and OS-World-G, with particularly large improvements on OS-World-G.

Figure 3.1 places Gelato-30B-A3B in context against leading models. Despite using only 3B active parameters per token, Gelato-30B-A3B achieves **63.8%** on ScreenSpot-Pro and **69.1%** on OS-World-G, outperforming GTA1-32B, Qwen3-VL-235B-Instruct, and OpenCUA-72B on both benchmarks. Detailed per-category results are presented in Section 3.1.

Performance with Refusal We elicit refusal behavior, the ability to decline grounding when the target element cannot be located, from Gelato-30B-A3B without explicitly training for it. By appending "If you cannot find the element, return refusal" to the instruction prompt and including refusal cases in the evaluation (previously treated as zero-accuracy), we raise overall accuracy on OS-World-G to 69.15% (+1.96 pp) and on OS-World-G (Refined) to 74.65% (+1.25 pp).

3.2 OS-World Agent

To measure end-to-end agent performance, we evaluate Gelato-30B-A3B on the OS-World benchmark as the grounding module together with GPT-5.

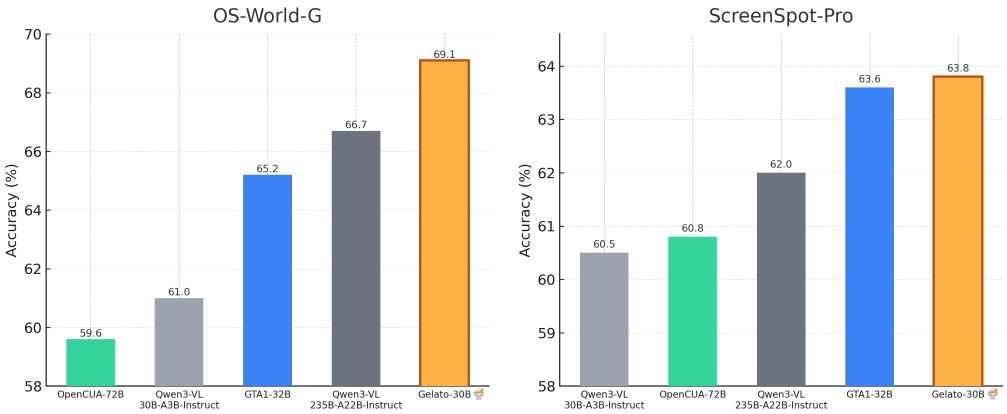


Figure 3.1: Final benchmark comparison of Gelato-30B-A3B against leading models. **Left:** OS-World-G accuracy. **Right:** ScreenSpot-Pro accuracy. Gelato-30B-A3B achieves the highest accuracy on both benchmarks despite having fewer active parameters than all competitors.

3.2.1 Agent Harness

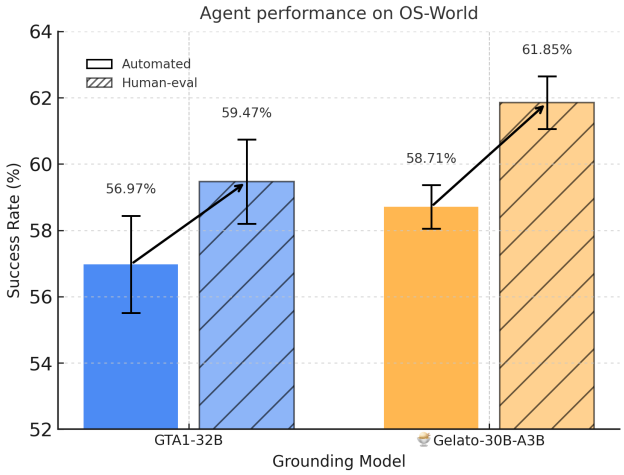
We evaluate Gelato-30B-A3B as a computer-use agent on OS-World using the GTA1.5 agent framework. [54]. The agent uses GPT-5 as a planning model and has a maximum of 50 steps, waiting 3 seconds between actions. We made minor modifications to the agent code, including a change to properly invoke the spreadsheet cell modification tool and an additional delay between trajectory completion and evaluation to ensure the VM state fully updates.

3.2.2 Reproducibility Issues

We found that many of the issues discussed in the EpochAI article critiquing OS-World [55] significantly affect reproducibility. Benchmarking agent performance proved challenging due to:

1. Non-deterministic planner behavior combined with insufficient trial repetitions, making fair comparison against prior work difficult.
2. Changing evaluation prompts without explicit versioning.
3. Incomplete evaluation coverage and ambiguous task specifications that fail to recognize valid alternative solutions.

To enable fair comparison, we ran three trials for both Gelato-30B-A3B and GTA1-32B in the same agent harness. Gelato-30B-A3B achieves **58.71 ± 0.66%** success rate on OS-World automated evaluation, performing on par or above GTA1-32B (**56.97 ± 1.47%** success rate).



Model	R1	R2	R3	Mean (SD)
Gelato-30B-A3B	**58.52 (211.26)**	**58.17 (209.99)**	**59.44 (214.58)**	**58.71 (0.66)**
+ Human eval	62.40 (225.26)	60.94 (219.99)	62.21 (224.58)	61.85 (0.79)
GTA1-32B	**55.31 (199.67)**	**57.53 (207.68)**	**58.08 (209.67)**	**56.97 (1.47)**
+ Human eval	58.08 (209.67)	59.75 (215.68)	60.57 (218.67)	59.47 (1.27)

Figure 3.2: OS-World agent performance across three runs with GPT-5 planner. Automated evaluation underestimates performance due to incomplete task specifications. Human evaluation shows Gelato-30B-A3B achieves **61.85%** success rate vs. GTA1-32B's **59.47%**.

3.2.3 Human Evaluation

We manually identified 20 tasks where the evaluation function is incomprehensive or the task specification is ambiguous. For each task, we review all agent trajectories and determine whether the task was successfully completed (Figure 3.2). With human evaluation corrections, Gelato-30B-A3B achieves **61.85** $\pm$ **0.79%** success rate compared to the automated evaluation result of **58.71** $\pm$ **0.66%**. Similarly, GTA1-32B achieves **59.47** $\pm$ **1.27%** success rate with human evaluation compared to **56.97** $\pm$ **1.47%** on automated evaluation.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This thesis presented Gelato-30B-A3B, a state-of-the-art grounding model for GUI computer-use tasks. Through careful data pool construction, novel filtering approaches, and effective reinforcement learning training, we achieved significant improvements over prior work on multiple benchmarks.

Our key contributions include:

- **Click-100k**: A high-quality, open-source dataset built through principled filtering and enrichment
- **Filtering methodology**: Model-based difficulty and alignment filtering that significantly improves dataset quality
- **Training recipe**: Effective RL training approach building on GRPO with practical simplifications
- **Strong results**: State-of-the-art performance on ScreenSpot-Pro (63.88%) and OS-World-G (69.15% / 74.65%)
- **Agent performance**: Demonstrated end-to-end agent improvements on OS-World (61.85% with human evaluation)

Beyond these technical contributions, our work highlights important challenges in agent evaluation and reproducibility. The 3 percentage point gap between automated and human evaluation, combined with non-deterministic planning models and incomplete evaluation functions, suggests that the field needs more rigorous evaluation practices.

Looking forward, grounding models remain a critical component of computer-use agents. As we push toward more capable and general-purpose agents, continued improvements in grounding accuracy, efficiency, and robustness will be

essential. We hope that our open-source release of Click-100k, trained models, and evaluation artifacts will accelerate progress in this important area.

The path to truly general-purpose computer-use agents is long, but strong grounding models like Gelato represent an important step forward. By combining high-quality data, effective training methods, and rigorous evaluation, we can continue to narrow the gap between human and machine capabilities in navigating digital interfaces.

Chapter 5

Future Work

Bibliography

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020. [Online]. Available: https://arxiv.org/abs/2001.08361

[2] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," 2017. [Online]. Available: https://arxiv.org/abs/1712.00409

[3] A. Aghajanyan, L. Yu, A. Conneau, W.-N. Hsu, K. Hambardzumyan, S. Zhang, S. Roller, N. Goyal, O. Levy, and L. Zettlemoyer, "Scaling laws for generative mixed-modal language models," 2023. [Online]. Available: https://arxiv.org/abs/2301.03728

[4] M. Cherti, R. Beaumont, R. Wightman, M. Wortsman, G. Ilharco, C. Gordon, C. Schuhmann, L. Schmidt, and J. Jitsev, "Reproducible scaling laws for contrastive language-image learning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2023, pp. 2818–2829. [Online]. Available: http://dx.doi.org/10.1109/CVPR52729.2023.00276

[5] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun,

T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[6] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022. [Online]. Available: https://arxiv.org/abs/2112.10752

[7] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[8] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo, ""Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. New York, NY, USA:

Association for Computing Machinery, 2021, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3411764.3445518

[9] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[10] Adept, "Persimmon-8b: A fully permissive 8b parameter language model," Blog post, 2023, accessed: 2026. [Online]. Available: https://www.adept.ai/blog/persimmon-8b/

[11] C. Schuhmann, R. Vencu, R. Beaumont, R. Kaczmarczyk, C. Mullis, A. Katta, T. Coombes, J. Jitsev, and A. Komatsuzaki, "Laion-400m: Open dataset of clip-filtered 400 million image-text pairs," 2021. [Online]. Available: https://arxiv.org/abs/2111.02114

[12] M. Byeon, B. Park, H. Kim, S. Lee, W. Baek, and S. Kim, "Coyo-700m: Image-text pair dataset," https://github.com/kakaobrain/coyo-dataset, 2022. [Online]. Available: https://github.com/kakaobrain/coyo-dataset

[13] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," 2020. [Online]. Available: https://arxiv.org/abs/2101.00027

[14] M. Mazumder, C. Banbury, X. Yao, B. Karlaš, W. G. Rojas, S. Diamos, G. Diamos, L. He, A. Parrish, H. R. Kirk, J. Quaye, C. Rastogi, D. Kiela, D. Jurado, D. Kanter, R. Mosquera, J. Ciro, L. Aroyo, B. Acun, L. Chen, M. S. Raje, M. Bartolo, S. Eyuboglu, A. Ghorbani, E. Goodman, O. Inel, T. Kane, C. R. Kirkpatrick, T.-S. Kuo, J. Mueller, T. Thrush, J. Vanschoren, M. Warren, A. Williams, S. Yeung, N. Ardalani, P. Paritosh, L. Bat-Leah, C. Zhang, J. Zou, C.-J. Wu, C. Coleman, A. Ng, P. Mattson, and V. J. Reddi, "Dataperf: Benchmarks for data-centric ai development," 2023. [Online]. Available: https://arxiv.org/abs/2207.10062

[15] S. Y. Gadre, G. Ilharco, A. Fang, J. Hayase, G. Smyrnis, T. Nguyen, R. Marten, M. Wortsman, D. Ghosh, J. Zhang, E. Orgad, R. Entezari, G. Daras, S. Pratt, V. Ramanujan, Y. Bitton, K. Marathe, S. Mussmann, R. Vencu, M. Cherti, R. Krishna, P. W. Koh, O. Saukh, A. Ratner, S. Song, H. Hajishirzi, A. Farhadi, R. Beaumont, S. Oh, A. Dimakis, J. Jitsev, Y. Carmon, V. Shankar, and L. Schmidt, "Datacomp: In search of the next generation of multimodal datasets," 2023. [Online]. Available: https://arxiv.org/abs/2304.14108

[16] H. Xu, S. Xie, X. E. Tan, P.-Y. Huang, R. Howes, V. Sharma, S.-W. Li, G. Ghosh, L. Zettlemoyer, and C. Feichtenhofer, "Demystifying clip data," 2025. [Online]. Available: https://arxiv.org/abs/2309.16671

[17] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[18] R. Thoppilan, D. D. Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, V. Zhao, Y. Zhou, C.-C. Chang, I. Krivokon, W. Rusch, M. Pickett, P. Srinivasan, L. Man, K. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. Chi, and Q. Le, "Lamda: Language models for dialog applications," 2022. [Online]. Available: https://arxiv.org/abs/2201.08239

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023. [Online]. Available: https://arxiv.org/abs/2210.03629

[20] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," 2023. [Online]. Available: https://arxiv.org/abs/2302.04761

[21] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2web: Towards a generalist agent for the web," 2023. [Online]. Available: https://arxiv.org/abs/2306.06070

[22] Browser Use, "Browser use: Make websites accessible for ai agents," https://github.com/browser-use/browser-use, 2024, gitHub repository, MIT license. [Online]. Available: https://github.com/browser-use/browser-use

[23] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman, "Webgpt: Browser-assisted question-answering with human feedback," 2022. [Online]. Available: https://arxiv.org/abs/2112.09332

[24] A. Singh *et al.*, "Openai gpt-5 system card," 2025. [Online]. Available: https://arxiv.org/abs/2601.03267

[25] Common Crawl, "Common crawl: A free, open repository of web crawl data," https://commoncrawl.org, 2007, 501(c)(3) non-profit organization, founded 2007. [Online]. Available: https://commoncrawl.org

[26] 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," 2025. [Online]. Available: https://arxiv.org/abs/2501.12326

[27] ByteDance Seed, "Ui-tars-1.5: Open-source multimodal agent with reinforcement learning," Hugging Face model card, 2025, built upon UI-TARS with advanced reasoning via reinforcement learning. [Online]. Available: https://huggingface.co/ByteDance-Seed/UI-TARS-1.5-7B

[28] H. Wang, H. Zou, H. Song *et al.*, "Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2509.02544

[29] Z. Yang *et al.*, "Gta1: Gui test-time scaling agent," 2025. [Online]. Available: https://arxiv.org/abs/2507.05791

[30] S. Yuan *et al.*, "Enhancing visual grounding for gui agents via self-evolutionary reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2505.12370

[31] Y. Gou *et al.*, "Navigating the digital world as humans do: Universal visual grounding for gui agents," 2024. [Online]. Available: https://arxiv.org/abs/2410.05243

[32] W. Li *et al.*, "Autogui: Scaling gui grounding with automatic functional annotation," 2025. [Online]. Available: https://arxiv.org/abs/2502.01977

[33] Z. Lin *et al.*, "Showui: One vision-language-action model for gui visual agent," 2024. [Online]. Available: https://arxiv.org/abs/2411.17465

[34] Z. Wu *et al.*, "Os-atlas: A foundation action model for generalist gui agents," 2024. [Online]. Available: https://arxiv.org/abs/2410.23218

[35] AgentSea, "Waveui-25k," Hugging Face dataset, 2024. [Online]. Available: https://huggingface.co/datasets/agentsea/waveui-25k

[36] X. He *et al.*, "Efficient agent training for computer use," 2025. [Online]. Available: https://arxiv.org/abs/2505.13909

[37] M. Deitke *et al.*, "Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models," 2024. [Online]. Available: https://arxiv.org/abs/2409.17146

[38] K. Cheng *et al.*, "Seeclick: Harnessing gui grounding for advanced visual gui agents," 2024. [Online]. Available: https://arxiv.org/abs/2401.10935

[39] A. Nayak *et al.*, "Ui-vision: A desktop-centric gui benchmark for visual perception and interaction," 2025. [Online]. Available: https://arxiv.org/abs/2503.15661

[40] T. Xie *et al.*, "Scaling computer-use grounding via user interface decomposition and synthesis," 2025. [Online]. Available: https://arxiv.org/abs/2505.13227

[41] K. Cheng, Q. Sun, Y. Chu, F. Xu, Y. Li, J. Zhang, and Z. Wu, "Seeclick: Harnessing gui grounding for advanced visual gui agents," 2024. [Online]. Available: https://arxiv.org/abs/2401.10935

[42] Z. Wu, Z. Wu, F. Xu, Y. Wang, Q. Sun, C. Jia, K. Cheng, Z. Ding, L. Chen, P. P. Liang, and Y. Qiao, "Os-atlas: A foundation action model for generalist gui agents," 2024. [Online]. Available: https://arxiv.org/abs/2410.23218

[43] Y. Li *et al.*, "Screenspot-pro: Gui grounding for professional high-resolution computer use," 2025. [Online]. Available: https://arxiv.org/abs/2504.07981

[44] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu, "Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments," 2024. [Online]. Available: https://arxiv.org/abs/2404.07972

[45] Qwen Team, "Qwen2.5-vl technical report," Mar. 2025, arXiv preprint arXiv:2502.13923. [Online]. Available: https://arxiv.org/abs/2502.13923

[46] Qwen Team, "Qwen3-vl technical report," Dec. 2025, arXiv preprint arXiv:2511.21631. [Online]. Available: https://arxiv.org/abs/2511.21631

[47] Y. Lu *et al.*, "Omniparser for pure vision based gui agent," 2024. [Online]. Available: https://arxiv.org/abs/2408.00203

[48] Z. Gu *et al.*, "Ui-venus technical report: Building high-performance ui agents with rft," 2025. [Online]. Available: https://arxiv.org/abs/2508.10833

[49] A. Fang, A. M. Jose, A. Jain, L. Schmidt, A. Toshev, and V. Shankar, "Data filtering networks," 2023. [Online]. Available: https://arxiv.org/abs/2309.17425

[50] M. Zhang, Z. Xu, J. Zhu, Q. Dai, K. Qiu, Y. Yang, C. Luo, T. Chen, J. Wagle, T. Franklin, and B. Guo, "Phi-ground tech report: Advancing perception in gui grounding," 2025. [Online]. Available: https://arxiv.org/abs/2507.23779

[51] Z. Shao *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," 2024. [Online]. Available: https://arxiv.org/abs/2402.03300

[52] T. Yu *et al.*, "Dapo: An open-source llm reinforcement learning system at scale," 2025. [Online]. Available: https://arxiv.org/abs/2503.14476

[53] Y. Zheng, J. Lu, S. Wang, Z. Feng, D. Kuang, and Y. Xiong, "Easyr1: An efficient, scalable, multi-modality rl training framework," https://github.com/hiyouga/EasyR1, 2025, gitHub repository. [Online]. Available: https://github.com/hiyouga/EasyR1

[54] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, ..., and T. Yu, "gta15_agent.py (OSWorld Module)," GitHub repository, 2024, commit ddb8372a6cbb51a29583cc1c0fe8c090e61219b7. [Online]. Available: https://github.com/xlang-ai/OSWorld/blob/ddb8372a6cbb51a29583cc1c0fe8c090e61219b7/mm_agents/gta1/gta15_agent.py

[55] F. Brand and G. Burnham, "What does osworld tell us about ai's ability to use computers?" Epoch AI Blog, 2025, https://epoch.ai/blog/what-does-osworld-tell-us-about-ais-ability-to-use-computers.

Appendix A

Appendix

A.1 Data Source Examples

Figure A.1 shows representative examples from six of the data sources that comprise the Click-100k training set. Each sample consists of a screenshot paired with a natural language grounding instruction. The sources span web pages (ShowUI-Web, WaveUI), desktop applications across multiple operating systems (PC-Agent-E, OmniAct, ShowUI-Desktop), and mixed environments (OS-Atlas), illustrating the diversity of GUI contexts in the data pool.

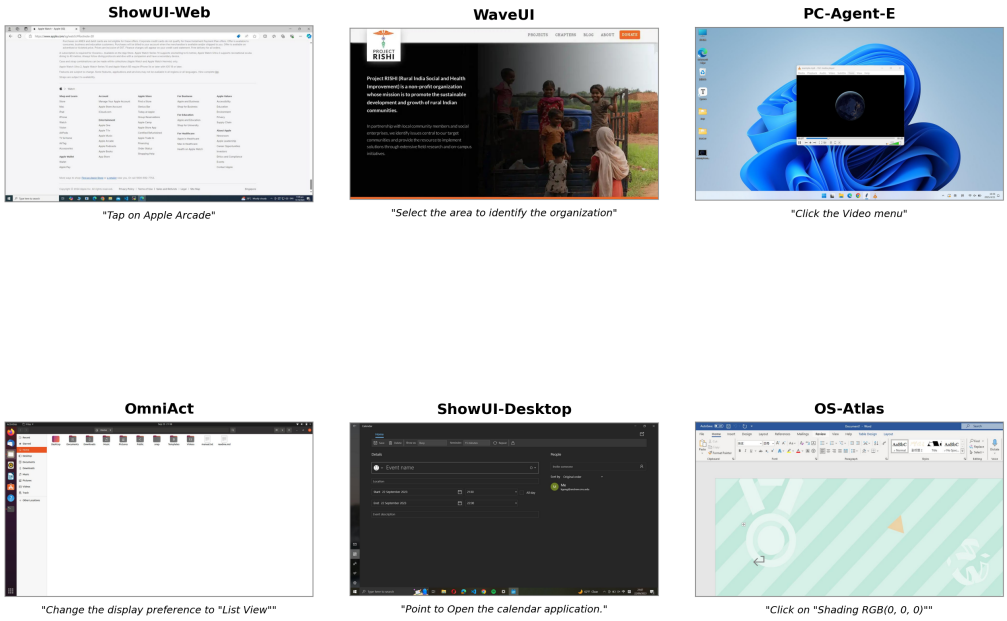


Figure A.1: Example training samples from six data sources in Click-100k. Each panel shows a screenshot with its associated grounding instruction (italic text below). The data pool covers web browsing, desktop applications, and professional software across Windows, macOS, and Linux environments.

A.2 Supplemental Data

This section contains supplemental information and data that supports the main thesis content.

Table A.1: Complete list of professional applications covered by the YouTube tutorial data collection pipeline.

3ds Max	Adobe Acrobat	Adobe After Effects	Adobe Dreamweaver
Adobe Illustrator	Adobe InDesign	Adobe Lightroom	Adobe Photoshop
Adobe Premiere	Airmail	Altium Designer	Android Studio
ANSYS	Apple Mail	Asana	Atom
AutoCAD	Autodesk Eagle	Autodesk Inventor	Autodesk Maya
Autodesk Revit	Avid Media Composer	Axure	Abaqus
Balsamiq	Blender	Brave	Burp Suite
Cadence Virtuoso	Catia	Cinema4D	COMSOL
Confluence	CryEngine	Cubase	DaVinci Resolve
DBeaver	Eviews	Figma	Final Cut Pro
FL Studio	Framer	Fusion 360	GameMaker
GIMP	IBM SPSS	Intel Quartus Prime	IntelliJ IDEA
Jupyter	KiCad	LabVIEW	LibreOffice Base
LibreOffice Calc	LibreOffice Draw	LibreOffice Impress	LibreOffice Math
LibreOffice Writer	Logic Pro X	Looker	Mailbird
MATLAB	Microsoft Edge	ModelSim	Mozilla Firefox
Notion	OBS	Obsidian	OneNote
OriginLab	Outlook	Power BI	PyCharm
Quartus	RStudio	Simulink	SolidWorks
Stata	Tableau	Thunderbird	Unity
Unreal Engine	VLC Media Player	VMWare	Vivado
Xcode			

A.3 Claude Prompt for Video Frame Annotation

The following prompt was used with Claude to generate structured grounding instructions from professional application screenshots collected via the video data pipeline (Section 2.3.3).

A.4 Instruction Relabeling Experiments

Instruction Relabeling We want to understand how scaling the data impacts the SFT process. We trained Qwen 2.5 VL 7B on 10k samples with the hard

Listing A.1: Prompt template used for Claude-based annotation of professional application screenshots.

```
Analyze this user interface image and provide helpful
instructions for using this application.

First, identify the main segments/regions of the interface
(e.g., "top toolbar", "left sidebar", "main canvas",
"color palette popup", "file menu dropdown", etc.).

Then, generate 20-30 diverse, specific instructions
organized by these segments. Focus on:

1. Common tasks a user might want to accomplish
2. How to navigate and use the visible features
3. Workflow suggestions and tips
4. Specific actions related to the tools and functions shown

CRITICAL INSTRUCTION REQUIREMENTS:
- All instructions must be written in English (US), even
  if the interface uses another language
- ALL instructions must be LEFT CLICK based actions only
  (no right click, typing, keyboard shortcuts, drag/drop)
- Each instruction must be VERY exact and require only a
  SINGLE LEFT CLICK to accomplish
- Instructions must be concise: 1-15 words maximum
  (1 word if applicable)
- Be VERY specific in your instructions
- Reference specific UI elements and their purposes
- Provide actionable guidance that users can follow by
  left clicking
- Include both beginner and intermediate level left click
  actions
- Vary instruction structure: use diverse phrasings like
  "Open...", "Access...", "View...", "Save...",
  "Navigate to...", "Switch to...", "Add...", "Remove...",
  "Select...", etc.
- Avoid starting every instruction with "Click" - use
  natural action verbs that imply left clicking
- Focus on what's actually visible and left clickable in
  the interface
- Pay special attention to dynamic elements: open tabs,
  dropdowns, pop-ups, context menus, modal dialogs, and
  expandable sections if they exist
- Avoid overly verbose descriptions

Return your response as a JSON object with this exact
format:
{
  "interface_description": "Brief description of what
    type of interface this is and its main purpose",
  "segments": {
    "segment_name_1": {
      "description": "Brief description of this screen
        segment",
      "instructions": [
        "Specific instruction 1",
        "Specific instruction 2",
        ...
      ]
    },
    "segment_name_2": {
      "description": "Brief description of this screen
        segment",
      "instructions": [
        "Specific instruction 1",
        "Specific instruction 2",
        ...
      ]
    }
  }
}

DO NOT hypothesize features not visible in the image.
Only provide instructions based on what you can clearly
see.

REMEMBER: ALL instructions must be accomplished with a
SINGLE LEFT CLICK only. No right click, typing, keyboard
shortcuts, or multi-step actions.
Use diverse instruction phrasings - examples: "Open File
menu", "Save document", "View project structure",
"Access settings", "Navigate to homepage", "Switch tabs",
"Add new item", "Close dialog", "Expand dropdown",
"Select tab".
```

samples filtered using GTA1 7B and easy samples filtered using Qwen 2.5 VL 7B. We also tried to improve the data by rewriting the prompts using LLM (Qwen3 4B) to remove noisy artifacts and by using image aware synthetic prompts (Qwen 2.5 VL 7B) to rewrite the prompts to include action intent.

Rewriting Strategy	SS Pro Accuracy	SS V2 Accuracy
No Rewriting	45.22%	91.05%
Rewritten Prompts using LLM (Qwen3 4B)	42.25%	88.84%
Image-aware Synthetic Prompts (Qwen 2.5 VL 7B)	39.27%	85.86%

Table A.2: Impact of prompt rewriting strategies on downstream task performance.

The following prompt template was used with Qwen3 4B to clean up noisy UI instructions:

A.5 Training Prompt Ablation

Three system prompts were compared in this ablation. Their full text is shown below.

We trained Qwen 2.5 VL 7B on 10k samples with different training prompts to investigate two questions: (1) whether the base Qwen model's verbose tool-calling computer-use prompt is necessary for good performance, and (2) the impact of including image resolution in the prompt.

We find that the default prompt is sufficient for the model to perform well, achieving competitive results across benchmarks. The impact of including image resolution in the prompt is mixed—while it slightly improves performance on ScreenSpot V2, it actually decreases performance on ScreenSpot Pro.

Prompt	SS Pro Accuracy	SS V2 Accuracy
Qwen Tool Calling Prompt + Image Resolution	37.76%	88.59%
Default Prompt + Image Resolution	36.12%	88.59%
Default Prompt	39.03%	87.68%

Table A.3: Impact of training prompt on downstream task performance for Qwen 2.5 VL 7B trained on 10k samples.

A.6 Cold-Start SFT Budget

Since [27] report a progressive training pipeline with continual SFT followed by RL, we conducted an experiment to understand the interaction between SFT and RL. We conducted cold-start experiments in which Qwen2.5-VL-7B-Instruct models are first fine-tuned on varying amounts of SFT data (1k, 3.3k, 10k, and 63k samples, each for a single epoch), and then trained with GRPO on the full 63k data pool for one epoch (63k was the size of our dataset at that time of the project). This experiment isolates how much the SFT initialization matters for subsequent RL gains.

As shown in Figure A.2, RL performance scales monotonically with SFT data budget, confirming that RL benefits from a stronger initialization and motivating our choice of the 63k SFT model as the starting point for subsequent experiments.

A.7 RL Hyperparameter Ablations

We ablate key RL hyperparameters—sampling temperature, learning rate, and KL penalty weight—to establish robust defaults for all subsequent experiments.

Temperature. Using the 10k SFT model, we compare temperatures of 0.65, 0.85, and 1.0. Temperature controls the diversity of rollouts and thus the exploration–exploitation trade-off during RL training.

Learning Rate and KL Weight. We compare our default hyperparameters (LR $= 1 \times 10^{-6}$, KL coef $= 1 \times 10^{-2}$) with parameters inspired by UI-Venus [48] (LR $= 4 \times 10^{-7}$, KL coef $= 4 \times 10^{-3}$) on the 63k SFT model. Results (Section **??**) show robustness to moderate variations in these hyperparameters.

A.8 Hyperparameter Search and Model Configuration

A.8.1 Vision Tower Fine-tuning

We investigated whether to freeze or fine-tune the vision tower during training. Results show that full fine-tuning significantly outperforms freezing the vision tower.

A.8.2 Learning Rate Search

We performed a learning rate sweep for Qwen 2.5 VL 7B on the 10k model-filtered dataset using Qwen2.5VL-7B for easy sample filtering and GTA1-7B for incorrect

Model Configuration	ScreenSpot Pro Accuracy
Full fine-tune	45.22%
Freeze vision tower	32.95%

Table A.4: Comparison of full fine-tuning versus freezing the vision tower for Qwen 2.5 VL 7B.

sample filtering.

Learning Rate	ScreenSpot Pro Accuracy	ScreenSpot V2 Accuracy
1e-5	32.57%	80.02%
5e-6	38.83%	87.80%
1e-6	**45.35%**	**90.27%**
5e-7	38.07%	89.10%

Table A.5: Learning rate sweep results. The optimal learning rate of 1e-6 (bold) achieves the best performance across all benchmarks.

Listing A.2: Prompt template used for instruction cleaning.

```
You are a text editor that cleans up UI instructions.
Your task is to fix formatting, style, and noise issues
while preserving the exact intent and meaning.

Rules:
1. Fix grammatical errors and awkward phrasing
2. Remove overly verbose descriptions - keep instructions
    concise
3. Clean up technical noise (HTML tags, underscores, etc
   .)
4. NEVER change the core action or target element
5. Keep the output as a single, clear instruction

Examples:
Input:  "Choose Located in the top right corner."
Output: "Choose the element in the top right corner."

Input:  "Based on my descriptions, find the locations of
         the mentioned element in this webpage screenshot
         (with point). This element provides access to
            the
         main WhatsApp page, allowing users to navigate
            to
         the primary WhatsApp interface."
Output: "Find the element that provides access to the
   main
         WhatsApp page."

Input:  "click on new_tab"
Output: "Click on new tab."

Input:  "Click on the button labeled 'Submit' which is
   used
         to submit the form"
Output: "Click on the Submit button."

Now clean up this instruction:
{instruction}

Cleaned instruction:
```

Listing A.3: Default Prompt (no resolution).

```
You are an expert UI element locator. Given a GUI
image and a user's element description, provide the
coordinates of the specified element as a single
(x,y) point. For elements with area, return the
center point.
Output the coordinate pair exactly:
(x,y)
```

Listing A.4: Default Prompt + Image Resolution.

```
You are an expert UI element locator. Given a GUI
image and a user's element description, provide the
coordinates of the specified element as a single
(x,y) point. The image resolution is height {height}
and width {width}. For elements with area, return
the center point.
Output the coordinate pair exactly:
(x,y)
```

Listing A.5: Qwen Tool Calling Prompt + Image Resolution (abbreviated).

```
You are a helpful assistant.

# Tools
You may call one or more functions to assist with the
user query. You are provided with function signatures
within <tools></tools> XML tags:

<tools>
{"type": "function", "function":
  {"name": "computer_use",
   "description": "Use a mouse and keyboard to interact
     with a computer, and take screenshots.
     * This is an interface to a desktop GUI. You do not
       have access to a terminal or applications menu.
     * Some applications may take time to start or process
       actions, so you may need to wait and take successive
       screenshots to see the results of your actions.
     * The screen's resolution is {width}x{height}.
     * Whenever you intend to move the cursor to click on
       an element, consult a screenshot to determine the
       coordinates of the element before moving the cursor.
     * Make sure to click any buttons, links, icons, etc
       with the cursor tip in the center of the element.",
   "parameters": {"properties":
     {"action": {"description": "The action to perform.",
       "enum": ["key", "type", "mouse_move", "left_click",
         "left_click_drag", "right_click", "middle_click",
         "double_click", "scroll", "wait", "terminate"],
       "type": "string"}, ...},
     "required": ["action"], "type": "object"}}}
</tools>

For each function call, return a json object with
function name and arguments within <tool_call></tool_call>
XML tags:
<tool_call>{"name": <function-name>,
  "arguments": <args-json-object>}</tool_call>
```

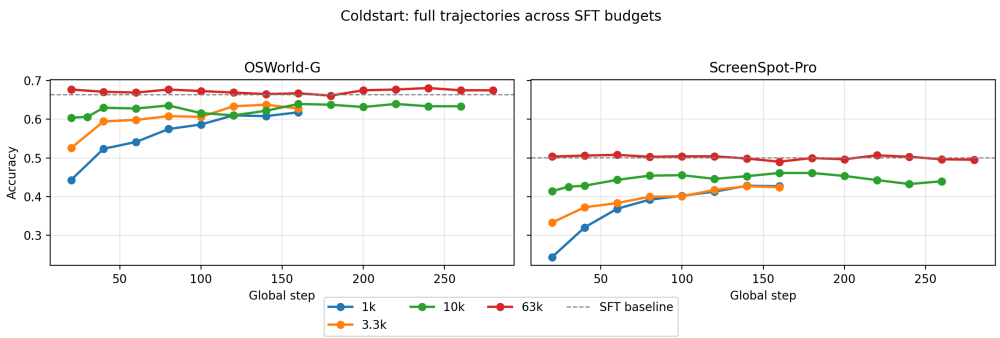


Figure A.2: Cold-start SFT budget experiment. Models fine-tuned on varying amounts of SFT data (1k, 3.3k, 10k, 63k) are subsequently trained with GRPO. RL performance scales monotonically with SFT data budget, confirming that RL benefits from a stronger initialization.