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 Akkus

aakkus@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

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

February 19, 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, empirical experimentation is relatively straightforward: given sufficient compute, permutations of architecture width, depth, normalization, and training hyperparameters can be rigorously evaluated, yielding consistent improvements over the years [9, 10]. The dataset side, however, remains far less transparent. Most large-scale training sets are not publicly released, leaving the community to pursue open reproductions [11, 12, 13]. While recent initiatives such as DataPerf, DataComp, and MetaCLIP [14, 15, 16] have begun to bridge this gap by providing consistent evaluation and reproduction frameworks, we argue that dataset design can and should leverage the same principled methodology as model design. In particular, large-scale dataset construction can generally be decomposed into two 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 Vision-Language Models

1.2.2 Computer Use Agents

Computer-use agents aim to enable Large Language Models (LLMs) to operate a computer on behalf of a user. Instead of merely generating text, the model is placed in a perception–action loop: it receives an instruction (e.g., “book the

cheapest flight tomorrow"), observes the current computer state (such as a screenshot or structured interface representation), predicts the next action (e.g., click, type, scroll), executes it, and then receives an updated observation. This process repeats until the task is completed. Formally, the agent learns a policy that maps (instruction , history , current observation) → next action . (instruction,history,current observation)→next action. Actions are typically low-level interface operations such as clicking at coordinates, typing text, scrolling, pressing hotkeys, or invoking system commands. Through sequential decision-making, the agent decomposes high-level goals into executable steps, effectively translating natural language into GUI interactions.

Computer Use Agents with Language-only Paradigm Early approaches relied on language-only interaction. The model operated over structured textual representations of interfaces—such as DOM Trees, accessibility trees, or API descriptions—and produced tool calls or code snippets that manipulated the environment (see Figure 1.1a). While effective in constrained settings, these methods depend on privileged system access such as APIs and handcrafted pipelines (for example to filter the prohibitively large DOM tree), and they struggle with visually grounded tasks where layout, icons, or rendered state changes matter.

Multi-modal Computer Use Agents By operating directly on images of the browser or desktop environment, the agent no longer depends on privileged APIs or internal program structures. Instead, it receives the same information a human user would (see Figure 1.1b). This shift fundamentally changes the observation space of the agent. Rather than conditioning on pre-filtered structured text describing interface elements, the model now conditions on high-dimensional visual input. The current state at each time step is an image, which must be interpreted to extract semantic meaning, identify interactive elements, and infer affordances. The agent must learn to associate visual patterns with functional behavior—for example, recognizing that a rectangular region with specific styling likely corresponds to a button, or that a text field expects typed input.

Planner-Grounder framework In screenshot-based computer use, the agent must solve two conceptually distinct problems at every interaction step: deciding *what* to do next and determining *where* to execute it. This observation has led to the widespread adoption of a planner–grounder framework, in which high-level decision making is separated from low-level spatial execution. The planner operates at the semantic level. Conditioned on the user instruction, the current screenshot, and the trajectory of previous interactions, it proposes the next action in natural language or structured form (e.g., "click the Settings button" or "type the email address into the login field"). The planner therefore reasons over task progress, interface state, and user intent, producing an abstract action

proposal without committing to pixel coordinates. The grounder, in contrast, operates at the perceptual level. Given the current screenshot and the planner's abstract action description, it predicts the precise interaction parameters required for execution—typically screen coordinates for clicks or drag operations. This modularization improves robustness: high-level reasoning is decoupled from pixel-level localization, allowing each component to specialize. Moreover, it enables stronger reasoning models to be used as planners while keeping the grounding module lightweight and optimized for spatial precision.

GUI Grounding GUI grounding refers to the task of mapping an abstract interaction intent to a concrete spatial location in the image. Formally, given a screenshot s s and an action proposal a a that refers to a target UI element, the grounding model predicts coordinates (x , y) (x,y) corresponding to the region of the intended element. Grounding is non-trivial for several reasons. First, GUI layouts are highly structured and information-dense, often containing dozens or hundreds of small interactive elements. Second, interfaces vary significantly across platforms, resolutions, and application domains. Third, visual cues such as color, typography, proximity, and grouping influence how elements are perceived and distinguished. Early approaches typically trained models in a supervised fashion to predict the center of the annotated bounding box of the target element. However, this formulation introduces a misalignment between training objective and task requirement: any coordinate inside the target region constitutes a valid interaction, yet center-point regression penalizes otherwise correct predictions. Recent methods therefore optimize grounding more directly by rewarding clicks that fall within the target region, aligning the learning signal with the actual success criterion of interaction.

1.2.3 GUI Grounding Datasets

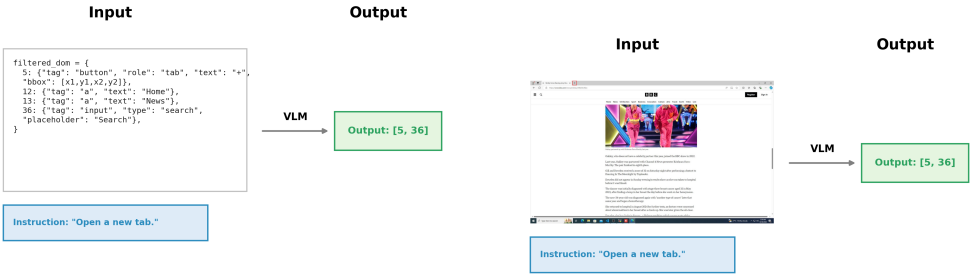
We collected, normalized and filtered multiple open source datasets for GUI grounding tasks.

ShowUI [27] presents a vision-language-action model for GUI visual agents, providing grounding data for both web and desktop interfaces. ShowUI demonstrates the importance of unified representations across different interface types.

AutoGUI [28] scales GUI grounding with automatic functional annotation, introducing methods to automatically generate grounding annotations from interface interactions.

SeeClick [29] harnesses GUI grounding for advanced visual GUI agents, contributing early work on connecting natural language instructions to visual elements.

PixMo Points [30], part of the Molmo project, provides open weights and



(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.

open data for state-of-the-art vision-language models, including pointing and grounding capabilities.

OS-Atlas [31] introduces a foundation action model for generalist GUI agents, providing grounding data across diverse operating system interfaces.

UGround [32] focuses on navigating the digital world as humans do through universal visual grounding for GUI agents.

WaveUI [33] contributes additional web and mobile interface grounding data.

PC-Agent-E [34] demonstrates efficient agent training for computer use by extracting grounding annotations from recorded trajectories.

UI-VISION [35] provides a desktop-centric GUI benchmark for visual perception and interaction, with particular emphasis on professional applications.

1.2.4 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 [36] 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 [**?**] 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 [37] 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 [38] 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 [39] 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

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 data curation, 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 Curation

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 curation pipeline draws from eight complementary sources: ShowUI [27] (covering both web and desktop environments), AutoGUI [28], PC-Agent-E [34], WaveUI [33], OS-Atlas [31], UGround [32], PixMo [30], and SeeClick [29]. Table 2.1 summarizes the scale and composition of these datasets, yielding approximately 9.8 million training samples.

Table 2.1: Dataset Statistics

Dataset	**# of Samples**	**# Text 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**

Normalization To ensure consistency across heterogeneous data sources, we normalize all grounding datasets into a unified format. This normalization process includes partitioning mobile and desktop samples into separate subsets and filtering the data to retain only click action annotations relevant to our grounding task.

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 processing each action's reasoning chain through Claude 3.7 Sonnet. For PixMo Points, we employ Qwen2.5-7B-VL as a filtering mechanism to identify and retain only samples depicting valid computer screen images.

Resizing Both Qwen2.5-VL [40] and Qwen3-VL [41] employ flexible patching mechanisms, where input images are dynamically resized to multiples of 28 pixels before being processed by the Vision Transformer (ViT). Failure to account for this preprocessing step can result in coordinate misalignment between predictions and ground truth annotations. To prevent such artifacts, we standardize all training and evaluation images by resizing them to dimensions that are multiples of 28 prior to model input.

Coordinate System The two model families employ distinct coordinate representation schemes. While Qwen2.5-VL uses absolute pixel coordinates, Qwen3-VL adopts a normalized coordinate system scaled to the range [0, 1000]. They report that this normalization improves robustness to variations in image resolution and aspect ratio across diverse inputs, while also simplifying post-processing and enhancing the usability of predicted coordinates in downstream applications. To maintain consistency with each model's native representation, we adapt our 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.

Data Quality Issues After normalization, we apply additional filtering to eliminate noisy and trivial samples. As illustrated in Figure 2.1, the main quality issues we encounter are: (1) overly simple interactions, such as trivial hyperlink clicks, which are easy to generate since they do not require sophisticated semantic understanding of GUI interfaces and can be bootstrapped from their text content; (2) misaligned instructions, where the instruction text and target region diverge due to artifacts from dataset creation errors; and (3) ambiguous tasks that lack sufficient context for precise grounding. These quality issues are systematically addressed through our filtering pipeline, which we describe in detail in Section 2.2.

2.2 Data Filtering

Hypothesis Testing To enable empirical testing of our data recipe, we require fast iterative evaluation. At this stage we therefore employ offline benchmarks (ScreenSpot-Pro, ScreenSpot v2, and OS-World), which provide signifi-

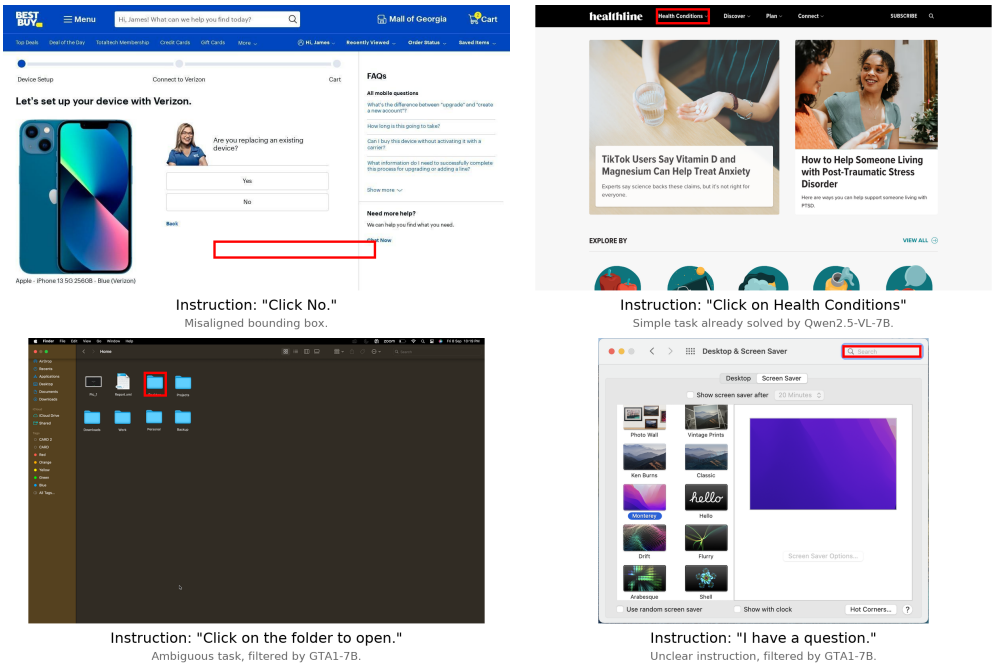


Figure 2.1: Examples of data quality issues encountered in curated datasets and their resolution through our filtering pipeline. Top left: misaligned bounding box; top right: simple task successfully filtered; bottom left: ambiguous task filtered by GTA1-7B; bottom right: unclear instruction filtered by GTA1-7B.

cantly faster evaluation compared to online agentic benchmarks. Furthermore, we use supervised fine-tuning (SFT) on Qwen2.5-VL to accelerate iteration compared to reinforcement learning approaches. We make the assumption that online agentic performance will transfer from our suite of offline benchmarks and that data recipes which work well with SFT will work well with RL.

OmniParser Filtering To address the data quality issues described above, we employ model-based filtering using the detection component of OmniParser [42], a vision-based screen parsing system. OmniParser's detection model is YOLO-based and identifies interactable UI elements by producing bounding boxes without requiring access to underlying code structures or accessibility trees, as illustrated in Figure 2.2. This approach enables parsing of diverse GUI environments—including web browsers, desktop applications, and mobile interfaces—where traditional DOM-based methods are unavailable or unreliable.

We leverage OmniParser's YOLO detector to validate the spatial alignment between instruction targets and actual UI elements. Specifically, we discard training samples where the ground truth click location lies outside all detected UI element bounding boxes, as illustrated in Figure 2.1. This filtering step ef-

fectively removes misaligned annotations that would otherwise introduce noise during training. For element classification (distinguishing text-based from icon-based UI elements), we apply EasyOCR to the detected bounding box regions. By operating purely on visual features, this approach generalizes across application types and platforms, making it particularly valuable for desktop-centric datasets where structured metadata is scarce.

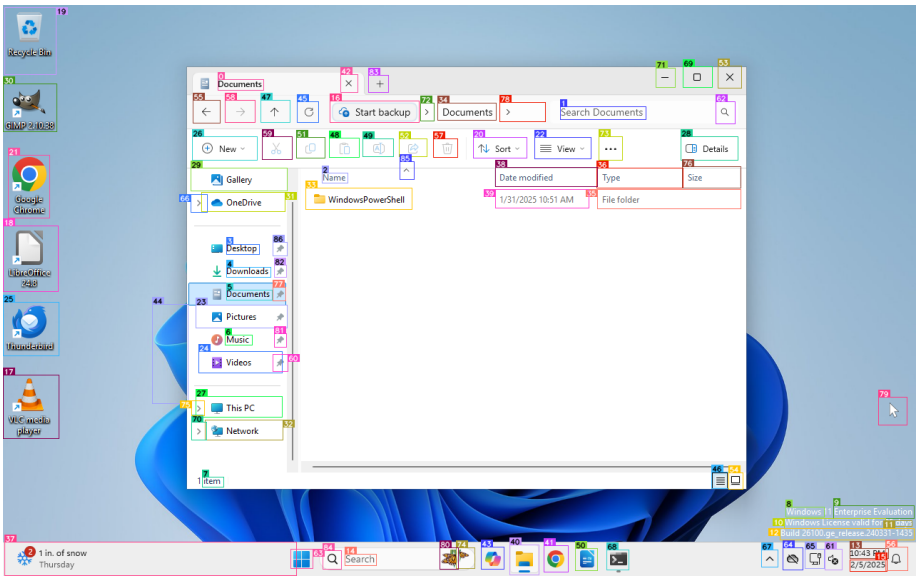
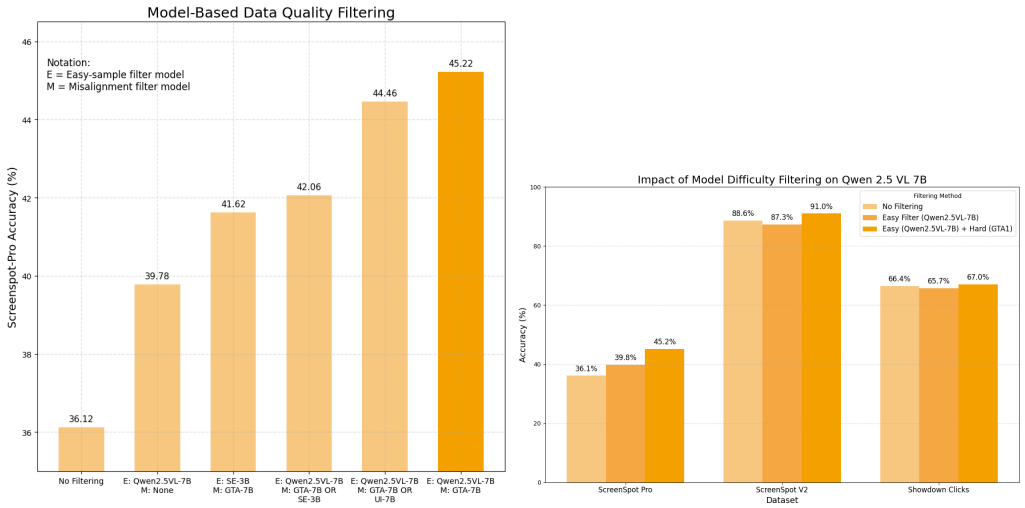


Figure 2.2: OmniParser filtering pipeline. The system detects UI elements and their bounding boxes (shown as colored rectangles), enabling validation of ground truth annotations. Samples where the target click location (red dot) falls outside all detected elements are filtered out as misaligned.

Difficulty Filtering To filter samples by difficulty, we use Qwen2.5VL-7B (for autofiltering) or SE-3B [43] for easy sample filtering, and GTA1-7B [44], UI-7B [45], or SE-3B for hard sample filtering. To validate this approach, we conduct detailed ablations by fine-tuning Qwen2.5-7B-VL on a source-balanced 10k subset and evaluating on ScreenSpot-Pro[3]. We find that using Qwen2.5-7B-VL as the difficulty filter and GTA1-7B-2507 as the alignment filter produces a +9 percentage point accuracy gain over unfiltered data and outperforms other filtering model combinations.



Overrepresentation of Text

Data Source Analysis In the previous filtering pipeline, we trained on a balanced set from all data sources. Our current data pool comprises a mixture of multiple data sources, and we seek to determine which sources have the greatest impact on downstream task performance. 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	Showdown Clicks Accuracy	OS-World G Accuracy
ShowUI-Web	36.43%	86.77%	63.73%	48.24%
AutoGUI	34.54%	87.68%	63.02%	47.06%
PC-Agent-E	34.28%	87.29%	63.73%	47.84%
WaveUI	33.40%	87.16%	63.02%	44.71%
Omniact	33.21%	87.16%	62.12%	44.90%
ShowUI-Desktop	32.01%	85.21%	60.68%	42.16%
UGround	31.82%	85.99%	63.73%	41.76%
Pixmo Points	30.68%	86.64%	61.04%	42.16%
SeeClick	29.22%	84.82%	61.94%	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. On Showdown Clicks, ShowUI-Web, PC-Agent-E, and UGround tie at 63.73%, followed by WaveUI and AutoGUI at 63.02%. 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 and sampled 10k samples using the optimal filtering approach identified in our ablation studies.

Data Pool	SS Pro Accuracy	SS V2 Accuracy	Showdown Clicks Accuracy	OS-World G Accuracy
All Data Sources	45.22%	91.05%	66.97%	N/A
Remove Bottom 3 on SS Pro (SeeClick, Pixmo Points, UGround)	45.03%	90.14%	68.94%	N/A
Remove Worst one on SS Pro (Pixmo Points)	44.78%	90.79%	67.68%	N/A

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

As shown in Table 2.3, removing the bottom three data sources does not yield consistent improvements across benchmarks. While Showdown Clicks accuracy increases slightly (68.94% vs. 66.97%), 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.

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. We implement a two-stage filtering strategy: hard samples are filtered using GTA1-7B, while easy samples are filtered using Qwen2.5-VL-7B. From this filtered pool, we sample data from each source at different scales (10k, 20k, 35k, and 80k samples), exploring both sampling with and without replacement.

As shown in Table 2.4, 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 and Showdown Clicks benchmarks remains relatively stable across different scales, suggesting that model capacity and data quality play complementary roles in determining final performance.

Table 2.4: Impact of Training Data Scale on Model Performance

Configuration	**SS Pro Acc.**	**SS V2 Acc.**	**Showdown Clicks Acc.**	**OS-World G Acc.**
10k (w/o replacement)	45.22%	91.05%	66.97%	N/A
10k + best prompt	46.23%	—	67.50%	N/A
20k (w/ replacement)	45.41%	90.66%	66.61%	N/A
20k (w/o replacement)	46.55%	90.27%	67.14%	N/A
35k (w/o replacement)	47.18%	90.66%	69.12%	N/A
80k + improvements	49.65%	90.79%	68.40%	N/A

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 exisiting instructions using LLM-based approaches (Qwen3 4B) as well as image-aware synthetic prompts (Qwen 2.5 VL 7B) to improve the quality of the instructions. Detailed results and ablations for these experiments are provided in the Appendix (see Tables A.3, A.2, A.4, and A.5).

2.3 Supplemental Data

After collecting data (Section 2.1) and unifying it into a pool and filtering this pool (Section 2.2), we evaluate performance and look for systematic weaknesses of the models to identify underrepresented areas of the training data. We specifically focus on the ScreenSpot Pro benchmark, because performance of even closed source models is poor on this benchmark. - Screenspot Pro is heavy on professional applications. This is a gap in many GUI models because data is much harder to collect due to missing APIs and domain knowledge for annotation. Furthermore it contains a lot of very large images, screens with multiple windows and complex layouts.

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 categories of challenging samples. First, single-screen samples at low resolutions (2-3MP) and aspect ratios around 1.8 prove most difficult, achieving only 15.79% accuracy. Second, dual-screen samples at high resolutions (6-8MP) and wide aspect ratios (3.6) present substantial challenges, with 43.75% accuracy. Third, ultra-high resolution single-screen samples (>8MP) yield 32.19% accuracy, suggesting that extreme resolutions strain the model's visual processing capabilities.

Image Resolution	SS Pro Accuracy	Sample Count	Coverage
2-3 MP	15.79%	19	1.2%
3-4 MP	54.49%	613	38.8%
4-5 MP	44.92%	236	14.9%
5-6 MP	67.78%	90	5.7%
6-8 MP	43.75%	272	17.2%
>8 MP	32.19%	351	22.2%
Overall	**46.55%**	**1,581**	**100%**

Table 2.5: Model performance stratified by image resolution on ScreenSpot Pro. The model achieves 736 correct predictions out of 1,581 total samples.

Aspect Ratio	SS Pro Accuracy	Sample Count	Coverage
1.5	56.17%	308	19.5%
1.6	57.14%	147	9.3%
1.8	43.33%	884	55.9%
3.6	39.67%	242	15.3%
Overall	**46.55%**	**1,581**	**100%**

Table 2.6: Model performance stratified by aspect ratio on ScreenSpot Pro. Ultra-wide displays (3.6 aspect ratio) present significant challenges.

Figure 2.3 visualizes the relationship between image characteristics and model performance. The heatmap (Figure 2.4) illustrates that performance degradation is particularly pronounced at the extremes: low-resolution standard displays and high-resolution ultra-wide configurations. These findings suggest that expanding training data coverage in these underrepresented regions could yield targeted performance improvements.

- Additionally, using prior models for difficulty filtering or annotation is problematic, because these models do not perform well in this domain either. Filtering

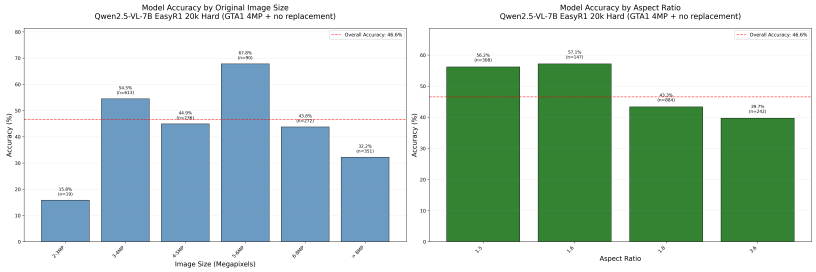


Figure 2.3: Model performance across different image resolutions and aspect ratios on ScreenSpot Pro. Error bars indicate 95% confidence intervals.

would probably lead to discard all the collected data.We therefore employ different strategies to collect data.

UI Vision and JEDI To improve robustness on professional desktop applications, we incorporate UI-Vision [35] 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 is a desktop-centric GUI dataset consisting of densely annotated screenshots collected from real-world applications. It 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. We normalize, resize and convert the UI Vision dataset to the same format as the other datasets. Importantly, we do not apply the entire filtering pipeline like before to these datasets. We obtain 5,733 samples from UI Vision.

We normalize, resize and convert the UI Vision [35] and JEDI [38] datasets to the same format as the other datasets. Importantly, we do not apply the entire filtering pipeline like before to these datasets. We obtain 5,733 samples from UI Vision and 18,032 samples from JEDI. We added UI-Vision and Jedi to the data pool. UI-Vision is a dataset of UI elements that we collected from Youtube and Jedi is a dataset of UI elements that we collected from Jedi.

Supplemental Data Experiments We conducted experiments using DeepSpeed ZeRO Stage 3 on 4 nodes with 4 40GB A100 GPUs at a batch size of 16. Table 2.7 shows the impact of adding UI-Vision and JEDI data to the training set.

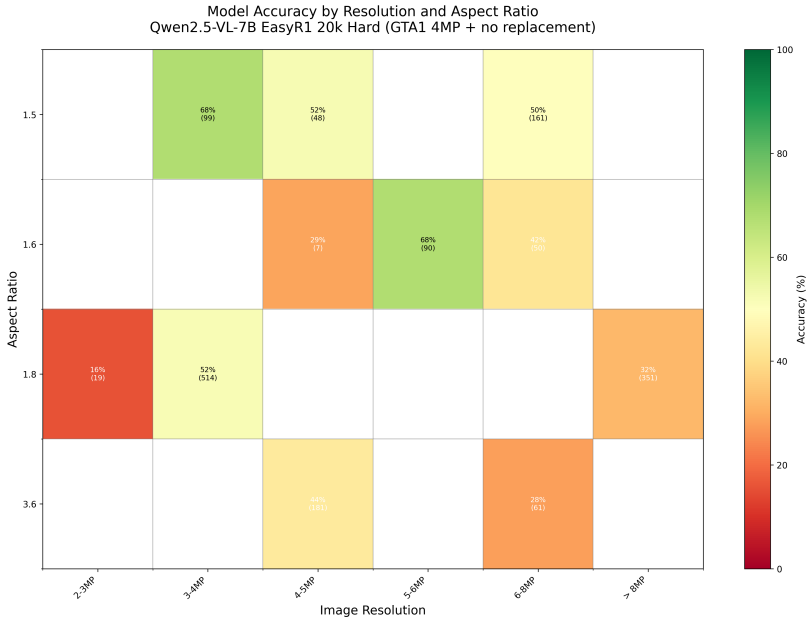


Figure 2.4: Performance heatmap showing accuracy by resolution and aspect ratio combinations. Darker regions indicate lower accuracy, highlighting performance bottlenecks at extreme configurations.

Table 2.7: Impact of Supplemental Data on Model Performance

Model Configuration	**LR**	**SS Pro**	**OS-World-G**
SFT-7B (38k)	1e-6	49.3%	57.4%
SFT-7B (38k) - 2 epochs	1e-6	50.16%	56.0%
SFT-7B (44k) + UI-Vision + YT icons	1e-6	47.6%	58.6%
SFT-7B (49k) + UI-Vision + YT + 5k JEDI	1e-6	47.9%	60.8%
SFT-7B (63k) + UI-Vision + YT + all JEDI	1e-6	50.09%	60.1%
SFT-7B (63k) + all data (BS 8, 2 nodes)	1e-6	48.26%	60.4%

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

In the first stage, we manually curated 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, covering tutorials in both English and Chinese to increase linguistic and visual diversity.

In the second stage, we extract unique screen-like frames from each down-

loaded 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 employs keyword-based application matching to handle naming variations, allocates frames in a balanced manner across applications, partitions by tutorial language, and 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. Each training sample consists of a single screenshot paired with two instructions corresponding to distinct UI segments. To mitigate redundancy, we deduplicate samples within each video using token-level overlap thresholds.

We added the in-house professional app data we collected from Youtube to the training data, added it to the model difficulty filtered data and sampled 10k samples total. We find that the performance slightly improves from 45.22

Table 2.8: Impact of In-house Professional Application Data

Data Configuration	**SS Pro**	**SS V2**	**Showdown**	**OS-World-G**
10k baseline	45.22%	91.05%	66.97%	N/A
10k + in-house prof. app data	46.11%	90.27%	67.50%	N/A

DOM Tree Methods For VSCode, Cursor and For Word, PowerPoint, Excel

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 by concatenating pairs of randomly selected frames from different data sources, comprising 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 2.9, this naive composition strategy leads to a substantial degradation in ScreenSpot-Pro accuracy, decreasing from 45.22% to 36.94%.

Table 2.9: 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 [46], 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 2.10, 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%. Performance on Showdown Clicks remains approximately unchanged.

Table 2.10: Effect of random image upscaling on benchmark accuracy.

Configuration	**SS-Pro Accuracy**	**SS V2 Accuracy**	**Showdown Clicks Accuracy**	**OS-World G Accuracy**
Baseline	45.22%	91.05%	66.97%	N/A
Random resize up to 4MP	44.14%	88.45%	67.14%	N/A

These results suggest that naive augmentation strategies for high-resolution inputs—whether through frame composition or random upscaling—do not transfer effectively to our training setup and may introduce noise that degrades grounding performance.

2.4 Training

Stabilizing RL Training Existing RL algorithms suffer from a gradient-decreasing problem when some prompts have accuracy equal to 1. For example, 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 results in zero policy gradients, shrinking the 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,

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

The expected effective batch size is

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

The variance is

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

This formulation reveals two problems: (1) a priori imbalance in sample difficulties, and (2) the number of samples with accuracy equal to 1 continues to increase during training, further exacerbating the problem. While DAPO focuses on online over-sampling and filtering to address this issue, we also employ a priori filtering. Without a priori filtering, we would need to oversample more aggressively and may exhaust the oversampling budget, resulting in increased variance in the effective batch size.

To this end, we propose filtering out prompts with accuracy equal to 1 and 0 before training, thereby decreasing the probability p and maintaining more stable RL training throughout the optimization process.

Chapter 3

Results

In this chapter, we present comprehensive evaluation results for Gelato-30B-A3B. We evaluate on isolated grounding benchmarks (ScreenSpot-Pro and OS-World-G) to measure grounding accuracy, and on the full OS-World agent benchmark to assess end-to-end agent performance. We also compare against prior state-of-the-art models and conduct human evaluation to measure the true performance beyond automated metrics.

3.1 Grounding Benchmark Evaluation

3.1.1 ScreenSpot-Pro

ScreenSpot-Pro [37] is a benchmark for GUI grounding on professional high-resolution computer interfaces. It tests the model's ability to locate UI elements in complex, realistic application interfaces.

Gelato-30B-A3B achieves **63.88% accuracy** on ScreenSpot-Pro at step 84 of training. This represents a +3.38 percentage point improvement over the Qwen3-VL-30B-A3B-Instruct base model, demonstrating that our RL training successfully specializes the model for grounding tasks.

3.1.2 OS-World-G

OS-World-G [38] is a grounding benchmark that tests models on diverse operating system interfaces. The benchmark includes a refined version that corrects annotation errors in the original benchmark.

Gelato-30B-A3B achieves:

- **67.19% accuracy on OS-World-G** (at step 84)
- **73.40% accuracy on OS-World-G (Refined)** (at step 84)

These results represent a +6.19 percentage point improvement on OS-World-G over the base model, demonstrating particularly strong gains on this diverse benchmark.

3.1.3 Performance with Refusal

As discussed in Chapter 2, eliciting refusal behavior by prompting the model to decline grounding when elements cannot be found further improves performance:

- **69.15% accuracy on OS-World-G** (+1.96 pp over non-refusal)
- **74.65% accuracy on OS-World-G (Refined)** (+1.25 pp over non-refusal)

3.1.4 Comparison with Prior Work

Table 3.1 summarizes Gelato-30B-A3B's performance compared to prior state-of-the-art models:

Model	ScreenSpot-Pro	OS-World-G	OS-World-G (Refined)
GTA1-32B	-	-	-
Qwen3-VL-235B-A22B-Instruct	-	-	-
Gelato-30B-A3B	**63.88%**	**69.15%**	**74.65%**

Table 3.1: Grounding benchmark results. Gelato-30B-A3B surpasses prior specialized computer grounding models like GTA1-32B and much larger VLMs.

Gelato-30B-A3B surpasses:

- **GTA1-32B**: A specialized grounding model with similar parameter count
- **Qwen3-VL-235B-A22B-Instruct**: A much larger (7.8×) general-purpose vision-language model

These comparisons demonstrate that our focused approach to data curation and RL training enables a 30B parameter model to outperform both specialized and general-purpose models that are larger or similarly sized.

3.2 OS-World Agent Evaluation

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 as the planning model.

3.2.1 Agent Harness

We evaluate Gelato-30B-A3B as a computer-use agent on OS-World using the GTA1.5 agent framework. The agent architecture combines:

- **Planning model**: GPT-5 generates high-level action plans
- **Grounding model**: Gelato-30B-A3B grounds instructions to specific UI locations
- **Action execution**: The system executes clicks, keyboard input, and other actions

The agent has a maximum of 50 steps per task and waits 3 seconds between actions to allow the interface to update.

3.2.2 Implementation Details

We made minor modifications to the agent code:

- Fixed spreadsheet cell modification tool invocation
- Added delay between trajectory completion and evaluation to ensure the VM state fully updates

Our modified implementation is available in `gelato_agent.py`.

3.2.3 Automated Evaluation Results

We conducted three independent trials for both Gelato-30B-A3B and GTA1-32B in the same agent harness to enable fair comparison. The experiments were conducted in a fixed snapshot of OS-World to ensure reproducibility.

Results:

- **Gelato-30B-A3B**: $58.71 \pm 0.66\%$ success rate
- **GTA1-32B**: $56.97 \pm 1.47\%$ success rate

Gelato-30B-A3B performs on par or above GTA1-32B, demonstrating that improved grounding translates to better end-to-end agent performance.

3.2.4 Variance Across Runs

The standard deviations across three runs indicate meaningful variance in agent performance:

- Gelato-30B-A3B shows relatively low variance ($\sigma = 0.66\%$)
- GTA1-32B shows higher variance ($\sigma = 1.47\%$)

This variance underscores the importance of multiple trial evaluations when comparing agent systems, as discussed further in Chapter 4.

3.2.5 Trajectory Release

We provide all of our agent's OS-World trajectories at `mlfoundations/gelato-osworld-agent-traje` to enable detailed analysis and support reproducibility.

3.3 Human Evaluation

Automated evaluation metrics can underestimate agent performance due to incomplete task specifications and ambiguous evaluation criteria. To measure true performance, we conduct human evaluation on a subset of tasks.

3.3.1 Methodology

We manually identified 20 tasks where the automated evaluation function is incomprehensive or the task specification is ambiguous. For each task, we review all agent trajectories across all runs and determine whether the task was successfully completed according to the intended goal.

Common issues with automated evaluation include:

- Evaluation functions that check only one valid solution path when multiple valid approaches exist
- Timing issues where the evaluation runs before the final state is saved
- Over-specific checks that reject valid alternative solutions

3.3.2 Human Evaluation Results

With human evaluation corrections:

- **Gelato-30B-A3B**: $61.85 \pm 0.79\%$ success rate

- **GTA1-32B**: 59.47 ± 1.27% success rate

Comparing to automated evaluation:

- Gelato-30B-A3B: +3.14 pp gain from human evaluation
- GTA1-32B: +2.50 pp gain from human evaluation

These results show that:

1. Automated evaluation significantly underestimates true agent performance (by 3 percentage points)
2. The relative ranking between models remains consistent
3. Gelato-30B-A3B maintains its advantage over GTA1-32B with human evaluation

3.3.3 Detailed Results

Detailed results for all 20 manually evaluated tasks are available in `evaluation/osworld-human-evals`. This is not a comprehensive manual evaluation—we focused on clearly problematic tasks that were also straightforward to verify. A more extensive human evaluation could reveal additional cases where automated metrics underestimate performance.

3.4 Summary

Our comprehensive evaluation demonstrates that Gelato-30B-A3B achieves state-of-the-art performance across multiple benchmarks:

1. **Isolated grounding**: Surpasses specialized grounding models and much larger VLMs on ScreenSpot-Pro and OS-World-G
2. **End-to-end agent performance**: Outperforms GTA1-32B on OS-World with both automated and human evaluation
3. **Refusal capability**: Successfully elicits refusal behavior without explicit training, improving performance on ambiguous cases
4. **Consistency**: Shows low variance across multiple evaluation runs, indicating robust performance

These results validate our approach to data curation, filtering, and RL training, demonstrating that careful attention to data quality and training methodology yields significant improvements in grounding model capabilities.

Chapter 4

Conclusion and Future Work

In this chapter, we discuss key challenges encountered during the development and evaluation of Gelato, particularly focusing on reproducibility issues in agent benchmarking. We then outline limitations of the current work and promising directions for future research.

4.1 Reproducibility Challenges in Agent Evaluation

During our evaluation of Gelato-30B-A3B on OS-World, we encountered significant reproducibility challenges that affect the broader field of computer-use agent research. These issues, many of which align with concerns raised in public discussions of agent benchmarks, merit careful consideration.

4.1.1 Non-Deterministic Planning Models

Modern language models like GPT-5, used as our planning model, exhibit non-deterministic behavior even with temperature set to zero. This non-determinism, combined with insufficient trial repetitions, makes fair comparison against prior work difficult.

We address this by:

- Running three independent trials for both Gelato-30B-A3B and GTA1-32B
- Using the same agent harness and evaluation environment for both models
- Reporting mean and standard deviation across trials

However, we observe meaningful variance across runs ($\sigma = 0.66\%$ for Gelato, $\sigma = 1.47\%$ for GTA1-32B), indicating that single-trial evaluations can be misleading.

4.1.2 Evaluation Prompt Versioning

We found that evaluation prompts have changed over time without explicit versioning in the OS-World benchmark. This creates challenges when comparing results across different papers or reproducing reported numbers.

Best practices for the community:

- Version all evaluation code and prompts
- Document any changes to evaluation procedures
- Release evaluation snapshots alongside paper results

4.1.3 Incomplete Evaluation Coverage

Our human evaluation revealed that many automated evaluation functions are incomplete or overly specific. Of the 20 tasks we manually reviewed, all showed cases where valid solutions were incorrectly marked as failures.

Common patterns include:

- Checking only one valid solution path when alternatives exist
- Timing issues where evaluation occurs before the final state is saved
- Over-specific assertions that reject reasonable variations

The 3 percentage point gap between automated and human evaluation (58.71% vs. 61.85% for Gelato) demonstrates that automated metrics significantly underestimate true agent performance.

4.1.4 Ambiguous Task Specifications

Many tasks in OS-World have ambiguous specifications that fail to recognize valid alternative solutions. For example:

- Tasks requesting "the latest version" without specifying how to determine "latest"
- Tasks with multiple reasonable interpretations of the end goal
- Tasks where the evaluation checks implementation details rather than outcomes

These ambiguities disproportionately affect more capable agents that may find creative solutions outside the narrow evaluation criteria.

4.1.5 Recommendations for Future Benchmarks

Based on our experience, we recommend that future agent benchmarks:

1. **Require multiple trials**: Report mean and standard deviation across at least 3 runs
2. **Version everything**: Use explicit versioning for evaluation code, prompts, and environments
3. **Conduct human evaluation**: Include manual verification on a representative sample
4. **Test evaluation functions**: Validate that evaluation functions accept all valid solutions
5. **Clarify specifications**: Write unambiguous task descriptions that focus on outcomes rather than implementation
6. **Release artifacts**: Provide trajectories, evaluation snapshots, and detailed results to enable verification

4.2 Limitations

While Gelato-30B-A3B achieves strong performance, several limitations merit discussion:

4.2.1 Dataset Limitations

Source diversity: Despite our efforts to incorporate professional application data, the dataset still has imbalanced coverage across application types. Spreadsheets and text editors are well-represented, while specialized domains (CAD software, scientific tools, etc.) remain underrepresented.

Language coverage: Click-100k primarily contains English instructions. Multilingual grounding remains an important area for future work.

Temporal dynamics: Static screenshots cannot capture all UI interactions. Time-dependent behaviors (animations, loading states) are not well-represented in the current dataset.

4.2.2 Model Limitations

Refusal calibration: While our model can elicit refusal behavior, it requires explicit prompting. Ideally, refusal should be the default behavior when elements cannot be found, without requiring special prompts.

Resolution constraints: The model is trained on specific input resolutions. Performance on very high-resolution displays or unusual aspect ratios may degrade.

Latency: At 30B parameters, Gelato-30B-A3B requires significant computational resources for inference. For real-time interactive applications, smaller models or distillation may be necessary.

4.2.3 Evaluation Limitations

Benchmark diversity: Current benchmarks focus primarily on desktop applications. Mobile interfaces, web applications, and other platforms need dedicated evaluation.

Long-horizon tasks: OS-World tasks typically complete in under 50 steps. Evaluating performance on longer-horizon tasks requiring sustained grounding across many interactions remains an open challenge.

Error recovery: Current evaluation does not measure how well models recover from grounding errors or handle unexpected interface states.

4.3 Future Directions

4.3.1 Scaling and Efficiency

Model distillation: Distilling Gelato-30B-A3B into smaller, faster models while retaining performance is an important direction for practical deployment.

Multimodal fusion: Incorporating additional modalities (e.g., accessibility tree information, OCR outputs) could improve grounding accuracy and efficiency.

Sparse architectures: Exploring mixture-of-experts or other sparse architectures could enable larger capacity with lower inference cost.

4.3.2 Dataset Expansion

Broader application coverage: Systematically expanding professional application coverage, particularly for specialized domains, would improve generalization.

Multilingual data: Creating multilingual versions of Click-100k would enable grounding across language boundaries.

Temporal grounding: Incorporating video data to handle time-dependent UI interactions and animations.

Synthetic data generation: Exploring methods to synthesize high-quality grounding data at scale, potentially using rendered interfaces or procedural generation.

4.3.3 Training Methodology

Multi-task learning: Joint training on grounding alongside related tasks (action prediction, outcome prediction, error detection) could improve overall agent capabilities.

Iterative refinement: Enabling models to make multiple attempts at grounding with feedback could improve robustness.

Reward shaping: More sophisticated reward functions beyond binary success/failure could accelerate learning and improve fine-grained control.

4.3.4 Agent Architecture

Unified models: Exploring architectures that combine planning and grounding in a single model rather than separate modules.

Memory and context: Incorporating longer-term memory of past interactions and UI states could improve grounding in complex multi-step tasks.

Self-verification: Enabling agents to verify their own grounding predictions before execution could reduce errors.

4.3.5 Evaluation and Analysis

Diagnostic benchmarks: Creating targeted benchmarks that test specific grounding capabilities (occlusion handling, similarity discrimination, spatial reasoning, etc.).

Failure analysis: Systematic categorization of grounding failures to identify common error modes and guide improvements.

Human-agent comparison: Detailed comparison of how humans and models approach grounding tasks could reveal fundamental limitations or opportunities.

4.4 Conclusion

This thesis presented Gelato-30B-A3B, a state-of-the-art grounding model for GUI computer-use tasks. Through careful data curation, novel filtering ap-

proaches, 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.

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 *et al.*, "Ui-tars: Pioneering automated gui interaction with native agents," 2025. [Online]. Available: https://arxiv.org/abs/2501.12326

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

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

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

[30] 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

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

[32] 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

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

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

[35] 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

[36] 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

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

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

[39] 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

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

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

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

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

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

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

[46] 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

Appendix A

Appendix

A.1 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.2 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 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	Showdown Clicks Accuracy	OS-World G Accuracy
No Rewriting	45.22%	91.05%	66.97%	N/A
Rewritten Prompts using LLM (Qwen3 4B)	42.25%	88.84%	66.24%	N/A
Image-aware Synthetic Prompts (Qwen 2.5 VL 7B)	39.27%	85.86%	67.14%	N/A

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.3 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 and Showdown Clicks, it actually decreases performance on ScreenSpot Pro.

Prompt	SS Pro Accuracy	SS V2 Accuracy	Showdown Clicks Accuracy	OS-W Accu
Qwen Tool Calling Prompt + Image Resolution	37.76%	88.59%	61.94%	N
Default Prompt + Image Resolution	36.12%	88.59%	66.43%	N
Default Prompt	39.03%	87.68%	64.09%	N

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

A.4 Hyperparameter Search and Model Configuration

A.4.1 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.

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.

A.4.2 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 sample filtering.

Learning Rate	ScreenSpot Pro Accuracy	ScreenSpot V2 Accuracy	Showdown Clicks Accuracy
1e-5	32.57%	80.02%	53.68%
5e-6	38.83%	87.80%	63.01%
1e-6	**45.35%**	**90.27%**	**67.50%**
5e-7	38.07%	89.10%	65.17%

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

Listing A.1: 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.2: 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.3: 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.4: 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>
```