Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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:
Supervisor 1, Supervisor 2
Prof. Dr. Roger Wattenhofer

February 16, 2026

Acknowledgements

We thank Florian Brand for sharing details on OS-World task quality issues and Yan Yang for details on GTA1 agent evaluation.

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

Abstract

We present Gelato-30B-A3B, a state-of-the-art grounding model for GUI computer-use tasks. Gelato is trained on our open-sourced Click-100k dataset and achieves 63.88% accuracy on ScreenSpot-Pro and 69.15% / 74.65% on OS-World-G / OS-World-G (Refined), surpassing prior specialized computer grounding models like GTA1-32B and much larger vision-language models including Qwen3-VL-235B-A22B-Instruct. When combined with GPT-5, 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. This thesis describes the complete pipeline from data curation and filtering to reinforcement learning training that enables these results.

Contents

Chapter 1

Introduction

1.1 Motivation

Graphical User Interfaces (GUIs) are central to how indi- viduals engage with the digital world, serving as virtual em- bodied interface for a range of daily activities. Meanwhile, Large Language Models (LLMs) [32] were first primarily developed as chatbots. However, with their ability to comprehend complex language instructions and seamlessly integrate tools, have shown significant potential in performing complex tasks through building Computer-Use agents [1, 13, 16, 56]. This progress inspires the development of intelligent Computer-Use agents that can significantly streamline human workflows based on user instructions.

Early efforts in Computer-Use agents have primarily focused on developing language-only agents [12, 47, 55] that rely on closed-source, API-based LLMs like GPT-4 [32]. These agents leverage text-rich metadata like HTML inputs or accessibility trees to perform navigation and other tasks. However, the text-only approach is limited in real-world applications, because (1) specifially for Computer-Use (as opposed to web use),rich API-based metadata is not always available and (2) users typically interact with user in- terfaces visually—through screenshots—without access to the underlying structural oracle information. This limita- tion underscores the need for developing Computer-Use visual agents that can perceive and interact with UIs as humans do.

The Computer-Use task conditioned on a users instruction can be roughly divided into two parts: (1) planning and (2) grounding. While planning is the process of generating one or a sequence of actions to perform next, grounding is the process of mapping the UI element(s) in questionto coordinates on the screen. This tasks is challenging because it requires rich semantic understanding of UI elements as well as the ability for precise pixel-level localization. Recognizing this gap, researchers have begun training vision-language models to acquire these new abilities. For instance, studies like [11,15,17] uti- lize web screenshot datasets to enhance large multi-modal models' element-grounding capabilities. Despite these advancements, training multi-modal mod- els for GUI visual agents continues to face significant challenges related to modeling and training. (a) Availability of

Training Data: Many models remain closed-source and therefore the availability of training data is limited. Even for open-source models, the training data or exact specification of the training data is often not available. (b) Training Data Composition: similar to the challenge faced by language based Computer-Use agents, the availability of a rich API for the web has lead to a overrepresentation of web data in the training data. Professional app data is difficult to obtain with reliable annotations. (c) If Computer-Use agents are to be deployed in the real world and not on standardized virtual machines, the varying screenshot aspect rations, resolutions and sizes make it necesarry to either resize the screenshots or train a model which can generalize to different resolutions and sizes.

1.2 Related Work

1.2.1 Vision-Language Models

1.2.2 Grounding Models and Computer-use Agents

GTA1 [?] introduces GUI test-time scaling agents and demonstrates the effectiveness of reinforcement learning for grounding tasks. GTA1 establishes strong baselines that our work builds upon and compares against.

UI-TARS 1.5 [?] pioneers automated GUI interaction with native agents, providing base models suitable for further fine-tuning on grounding tasks.

UI-Venus [?] builds high-performance UI agents with rejected sampling fine-tuning (RFT), demonstrating alternative training approaches for grounding models.

SE-GUI [?] enhances visual grounding for GUI agents via self-evolutionary reinforcement learning, introducing data quality filtering methods that inform our approach.

OmniParser [?] provides pure vision-based GUI agent capabilities and UI element detection, which we leverage in our filtering pipeline.

1.2.3 GUI Grounding Datasets

Several datasets have been developed for training GUI grounding models:

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

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

PixMo Points [?], part of the Molmo project, provides open weights and open data for state-of-the-art vision-language models, including pointing and grounding capabilities.

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

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

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

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

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

1.2.4 Evaluation Benchmarks

ScreenSpot-Pro [?] offers GUI grounding evaluation for professional high-resolution computer use, providing a challenging benchmark for measuring grounding accuracy.

OS-World-G [?] scale computer-use grounding via user interface decomposition and synthesis, introducing both isolated grounding benchmarks and end-to-end agent evaluation tasks.

1.2.5 Reinforcement Learning Methods

GRPO [?] (Group Relative Policy Optimization), introduced in the DeepSeek-Math work, provides an efficient reinforcement learning algorithm that we adapt for grounding model training.

DAPO [?] presents an open-source LLM reinforcement learning system at scale, demonstrating simplified RL objectives by removing KL-divergence terms, which we incorporate into our training approach.

1.3 Contributions

1.4 Thesis Structure

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 [**?**] (covering both web and desktop environments), AutoGUI [**?**], PC-Agent-E [**?**], WaveUI [**?**], OS-Atlas [**?**], UGround [**?**], PixMo [**?**], and SeeClick [**?**]. 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 [**?**] and Qwen3-VL [**?**] 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.

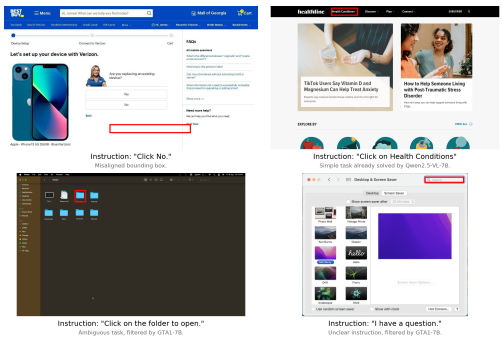
Coordinate System The two model families employ distinct coordinate representation schemes. While Qwen2.5-VL uses absolute pixel coordinates, Qwen3-VL adopts a normalized coordinate system scaled to the range [0, 1000]. This normalization improves robustness to variations in image resolution and aspect 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.

2.2 Data Filtering

Hypothesis Testing To enable empirical testing of our data recipe, we require fast iterative evaluation. We therefore employ offline benchmarks (ScreenSpot-Pro, ScreenSpot v2, and OS-World), which provide significantly 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 offline benchmarks and that data recipes which work well with SFT will work well with RL.

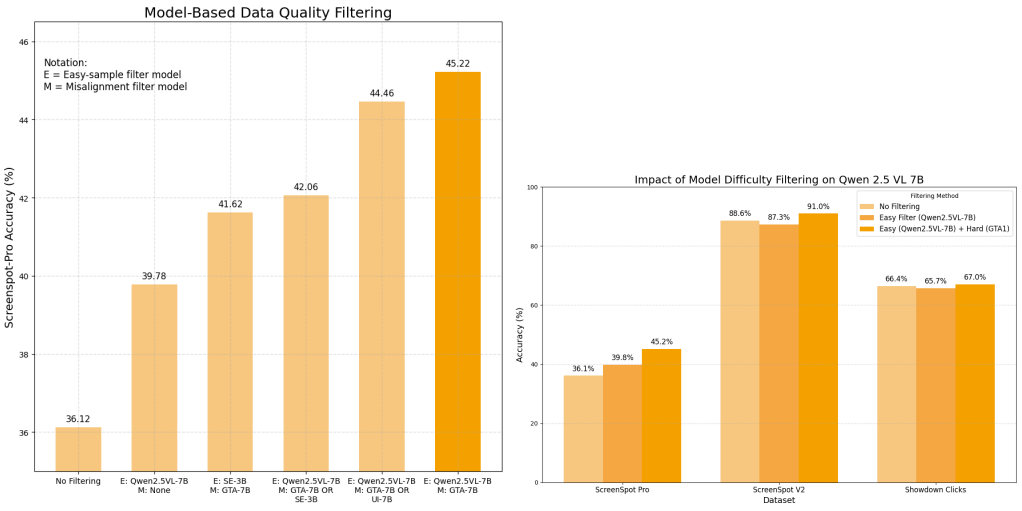
Data Quality Issues After normalization, we apply additional filtering to eliminate noisy and trivial samples. The main quality issues we encounter are: (1) overly simple interactions, such as trivial hyperlink clicks, which are easy to generate since they do not require sophisticated semantic understanding of GUI

interfaces and can be bootstrapped from their text content; and (2) misaligned instructions, where the instruction text and target region diverge due to artifacts from dataset creation errors.



OmniParser Filtering To address the data quality issues described above, we employ OmniParser [**?**] to discard cases where the target click location lies outside detected UI elements, effectively filtering out misaligned samples.

Difficulty Filtering To filter samples by difficulty, we use Qwen2.5VL-7B (for autofiltering) or SE-3B for easy sample filtering, and GTA1-7B, UI-7B, 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

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.

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.

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%)

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.

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.

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.

Additional Experiments We conducted several additional experiments to optimize the training process, including training prompts, instruction rewriting, vi-

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.

sion 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 instructions using LLM-based approaches (Qwen3 4B) and image-aware synthetic prompts (Qwen 2.5 VL 7B). Detailed results and ablations for these experiments are provided in the Appendix (see Tables A.2, A.1, A.3, and A.4).

2.3 Supplemental Data

2.4 Training

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 [**?**] 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 [**?**] 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.

Appendix A

Appendix

A.1 Supplemental Data

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

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