DESKFORGE: DENSE SUPERVISION FROM DESKTOP ENVIRONMENTS FOR COMPUTER-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents need to reliably ground action targets in complex desktop scenes, where multiple applications, overlapping windows, and visually similar controls compete for attention. Existing training data rarely pair such scenes with dense annotations or vary them in a controlled way. We introduce **DeskForge**, a controllable desktop environment that composes and explores real applications to generate large-scale supervision for computer-use agents. It varies application states, content, window layout, appearance, and resolution, and fuses screenshots, accessibility trees, and window geometry into dense element annotations while recording the outcome of each executed action. Using this environment, we construct **DeskForge-1M**, a corpus of 1.2M annotated desktop observations containing 159.7M element instances. We fine-tune four vision-language models on 200K grounding examples drawn from DeskForge-1M. All four improve across held-out desktop conditions and on all five external GUI grounding benchmarks; for Qwen3.5-4B, accuracy increases by 11.51 percentage points on ScreenSpot-Pro and 10.11 points on OSWorld-G. The gains also translate to long-horizon task completion: under a fixed planner, the fine-tuned action models solve more WebArena-Infinity and OpenApps tasks, with Qwen3.5-4B increasing from 31 to 50 of 119 tasks and from 3 to 15 of 100 tasks, respectively. These results show that controllable composition of real desktop environments provides a scalable source of supervision for improving both GUI grounding and long-horizon computer use. We will release the framework, dataset, and trained models.

1 INTRODUCTION

Desktop interfaces are compositional: application windows, dialogs, and system controls coexist on the same screen. Computer-use agents must ground actions within this dense visual context, distinguishing relevant controls from similar elements and cross-application distractors. The same application can present different content or occupy a different position within a multi-window layout. This motivates training resources that capture complete desktop scenes, retain detailed information about their structure, and support systematic variation across desktop configurations.

Recent work has expanded GUI supervision through structured screenshot corpora, large-scale grounding datasets, synthetic interfaces, and human demonstrations (Deka et al., 2017; Wu et al., 2023; Gurbuz et al., 2026; Gou et al., 2025; Wu et al., 2025; Xie et al., 2025; Feizi et al., 2026), while interactive environments make it possible to study agents and tasks under controlled application conditions (Ullrich et al., 2026; Wang et al., 2026a; Zhou, 2026). Each resource provides some of these properties, but none combines them: static corpora cannot vary the scenes they capture, most annotate isolated applications or single targets, and executable environments are designed for task evaluation rather than dense screen annotation (Table 1).

Our key idea is to generate the desktop itself rather than record it: when real applications are composed in controlled scenes, every screen is captured together with its interface structure and window layout, every executed action with its outcome, and the factors that make desktop grounding hard can be varied deliberately. We implement this idea in DeskForge, a Linux desktop environment (Figure 1). Real applications run in isolated sessions, with scene specifications controlling application selection and initial states, content, window layout and stacking, appearance, and resolution. Obtaining this supervision requires more than arranging application windows: accessibility APIs expose application-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

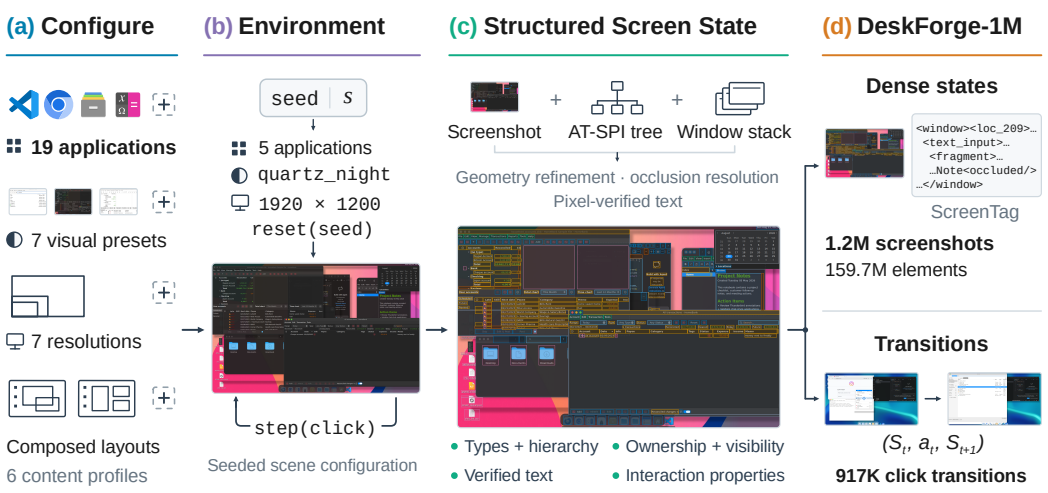


Figure 1: **Overview of DeskForge and DeskForge-1M.** (a) A scene specification configures the desktop. (b) Real applications are launched and explored. (c) Screenshots, accessibility trees, and window stacks yield dense annotations. (d) Annotated states and transitions form DeskForge-1M.

dependent and often incomplete interface structure, while rendered appearance, window relationships, and action outcomes require separate observations. The annotation pipeline therefore reconciles accessibility information, screenshots, and window geometry to construct dense annotations of element geometry, text, visibility, and interaction properties. It also retains interface hierarchy, application and window ownership, and stacking relationships, placing individual elements within the structure of the composed desktop. Executed clicks are linked to successor observations, recording how the interface changes through interaction.

Using DeskForge, we construct DeskForge-1M, a corpus of 1.2M annotated desktop observations containing 159.7M element instances and 917K recorded click transitions. It spans 19 applications, seven appearance presets, and seven display resolutions. Scene-level splits keep related captures together and test generalization to new scenes, held-out applications, a reserved appearance preset, and a reserved resolution. Natural-language instructions derived from recorded interactions provide a grounding view of the corpus, pairing each before screenshot and instruction with a single target.

Fine-tuning four vision-language models on DeskForge-1M grounding examples improves accuracy in every held-out condition and raises mean accuracy across five external GUI grounding benchmarks by 1.5 to 24.9 points. For Qwen, mean held-out accuracy rises from 76.26% to 87.54%, and the gain grows as more applications share the screen and as targets become partly occluded, the conditions that make desktop grounding hard. Externally, Qwen gains 11.51 percentage points on ScreenSpot-Pro and 10.11 on OSWorld-G. The dense annotations also train a whole-screen element detector: an RT-DETRv4-L model fine-tuned on DeskForge-1M outperforms the OmniParser v2 detector and ScreenParse YOLO11L on GroundCUA across three localization metrics.

To assess downstream utility, we fix a Qwen3.6-27B planner (Qwen Team, 2026b) and change only the action-model weights within each base/fine-tuned comparison. All four fine-tuned action models solve more tasks on a custom 119-task WebArena-Infinity panel and the 100-task OpenApps longer-horizon set. For Qwen, tasks solved increase from 31 to 50 and from 3 to 15, respectively. These comparisons connect single-target grounding supervision to improved multistep task completion within the tested planner-based agent. Our contributions are:

1. **DeskForge, a controllable desktop environment.** DeskForge composes and explores real applications under configurable desktop conditions, collecting dense structured observations and recorded interactions at scale.
2. **DeskForge-1M, a large-scale densely annotated desktop corpus.** The corpus contains 1.2M annotated observations, 159.7M element instances, and 917K recorded click transitions, combining dense per-screen annotations with action-linked before/after observations.

3. **Grounding gains and downstream task-completion benefits.** Fine-tuning four vision-language models improves held-out and external grounding and increases task completion under a fixed planner.

2 Related Work

Structured screen supervision. GUI supervision ranges from locating individual targets to recovering structured descriptions of a screen. Rico and WebUI pair screenshots with mobile view hierarchies and webpage metadata, respectively (Deka et al., 2017; Wu et al., 2023). Screenshot-to-structure learning builds on this information: Pix2Struct predicts simplified HTML from masked screenshots, while ScreenParse provides dense web-screen annotations and introduces the ScreenTag representation adopted here (Lee et al., 2023; Gurbuz et al., 2026; Nassar et al., 2025). GroundCUA brings dense, human-verified annotation to expert desktop workflows (Feizi et al., 2026). These resources motivate retaining element attributes and relationships beyond an action's target, but annotate individual web pages, mobile screens, or applications rather than composed multi-application desktops.

Automated collection and interface synthesis. Automated pipelines expand interface coverage (Cheng et al., 2024; Wu et al., 2025; Liu et al., 2026b), while UGround synthesizes referring expressions from webpage structure and visual content (Gou et al., 2025). Other pipelines vary the interfaces themselves: Jedi combines UI decomposition with synthesis and augmentation, while MolmoPoint-GUISyn renders generated HTML interfaces (Xie et al., 2025; AI2, 2026). GUIrilla explores running applications through native accessibility APIs to build state-action graphs, but processes each application individually (Garkot et al., 2025). DeskForge instead configures and explores multiple applications within shared desktop scenes, varying their joint layout and surrounding context.

Controllable environments and desktop composition. Executable environments connect interface actions to task-level outcomes, as in WebArena and OSWorld (Zhou et al., 2024; Xie et al., 2024). Environment control also supports different forms of variation: OpenApps changes application appearance and content to study reliability (Ullrich et al., 2026), while CUA-Gym and WebArena-Infinity construct environment states or applications together with tasks and verifiers (Wang et al., 2026a; Zhou, 2026). WinDeskGround composes captured window images to vary layout, occlusion, and distraction for grounding evaluation, but its static compositions cannot execute further actions (Zhao et al., 2026). DeskForge performs composition and interaction within a running desktop, collecting dense annotations together with observed click outcomes.

Interaction data and transition supervision. Interaction records connect screen observations to task-directed behavior. Mind2Web and AgentNet provide human demonstrations, while VideoCUA within CUA-Suite preserves continuous expert interaction (Deng et al., 2023; Wang et al., 2025b; Jian et al., 2026). GUI-360 collects trajectories through automated task execution and retains screenshots and accessibility metadata (Mu et al., 2025). An alternative reverses the order of interaction and task specification: OS-Genesis explores first and retrospectively derives tasks (Sun et al., 2025). DeskForge uses this interaction-first approach to construct single-step grounding instructions from recorded clicks and before/after observations. Recorded transitions can also support joint inverse and forward dynamics learning (Liu et al., 2026a); our corpus retains them for such objectives.

Table 1 summarizes the comparison: among the listed resources, only DeskForge combines dense labels, multi-application scenes, a configurable environment, visibility geometry, and recorded transitions.

3 The DeskForge Environment

DeskForge combines controllable desktop execution with structured observation to generate dense screen annotations and recorded interactions (Figure 1). Applications render their own interfaces within configured desktop scenes; the annotation pipeline then relates the rendered content to interface structure and records the outcomes of executed actions.

Table 1: **Scale and capabilities of GUI resources.** The DeskForge row combines its environment capabilities with the dataset counts of DeskForge-1M.

Resource	Scale		Capabilities				
	Images	Elements	Dense labels	Multi-app scenes	Configurable Environment	Visibility geometry	Recorded transitions
UGround	1.3M	10M	×	×	×	×	×
WinDeskGround	585†	1,356‡	×	✓	×	✓	×
MolmoPoint-GUISyn	36K	2M	✓	×	×	×	×
AgentNet	421K	421K	×	✓	×	×	✓
GroundCUA	55.6K	3.56M	✓	×	×	×	×
ScreenParse	771K	21M	✓	×	×	×	×
OS-Atlas	2.3M	13M	×	✓	×	×	×
ScaleCUA	1.6M	17.1M	✓	×	×	×	✓
DeskForge	**1.21M**	**159.7M**	✓	✓	✓	✓	✓

✓ supported; × not supported; †source window images; ‡target annotations. Definitions: Appendix F.

3.1 Scene composition and visual diversity

Each desktop scene is defined by a specification that selects applications and their initial states, prepares their content, and sets window layout and stacking, appearance, and display resolution. Sampling weights control the prevalence of these settings, while layout constraints define feasible multi-window arrangements. This places the same application in different contexts, with competing controls, labels, and partially overlapping windows. Application content is also configurable, so variation extends beyond window placement to the information presented within the interface. Each axis is extensible: appearance presets and resolutions are configuration entries, and an application is added through a manifest that specifies its launch command, staged content, and scripted initial states, provided its accessibility tree exposes the main interface content (Appendix B.1).

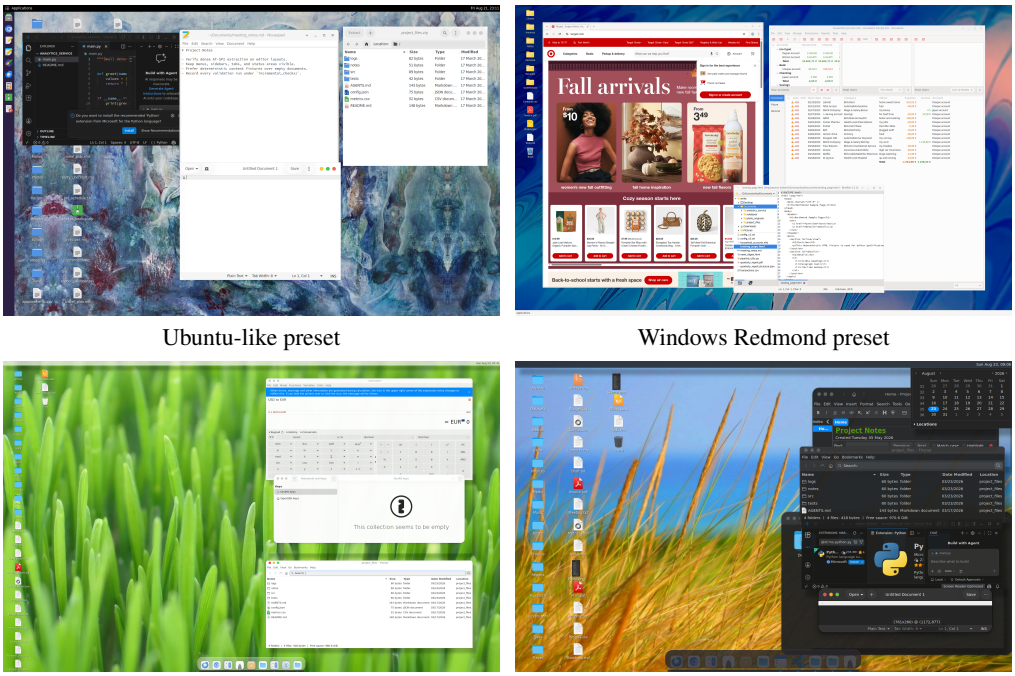
To broaden visual coverage beyond a single Linux desktop style, we integrate community-developed application themes, icon sets, and window decorations with configurable panels, docks, and backgrounds. Seven appearance presets span classic Linux, Ubuntu-like, Windows-inspired, and macOS-inspired styles, including light and dark variants. These assets configure the running desktop and application toolkits, varying widget appearance, iconography, and desktop layout rather than only the background image. Figure 2 illustrates the resulting visual variation. All presets share a Linux execution backend; they reproduce selected visual conventions rather than execute native Windows or macOS applications. Preset definitions and asset sources are given in Appendix B.2.

3.2 Dense structured annotations

The annotation pipeline combines screenshots, accessibility observations, and measured window relationships. Its output describes element types, text, geometry, and interaction properties together with parent–child structure, reading order, and application and window ownership. Accessibility-reported actions and control states, such as whether an element is enabled, selected, or expanded, supplement the visual annotations with information about the captured interface.

Accessibility structure alone does not determine what is visible. Geometry refinement and clipping account for element boundaries, ancestor containers, viewport limits, overlapping windows, and transient overlays. For partially covered elements, visible support is represented as a union of rectangles, preserving the exposed portions of a control rather than treating its entire extent as a valid click region. Accessibility names are kept separate from displayed text: character geometry and pixel checks assess which text is exposed, so an icon's semantic name is not automatically treated as text rendered on screen. Selected desktop controls omitted from accessibility trees are recovered from window geometry.

A visible, redundancy-reduced annotation view is the basis for screen parsing and ScreenTag serialization (Gurbuz et al., 2026), retaining structural nodes where needed. The environment separately preserves captured hidden elements and their visibility states, allowing the retained interface representation to extend beyond the currently exposed content. Automated audits exclude the 4.7% of



Ubuntu-like preset | Windows Redmond preset

macOS Tahoe-like preset | Quartz Night preset

Figure 2: **Appearance variation within DeskForge.** Representative captured desktops illustrate differences in application styling, iconography, window decorations, and panel or dock layout.

captures with inconsistent or implausibly sparse annotations, and pixel-level checks for missed and spurious annotations guided the pipeline's development. A human audit further confirms annotation quality, finding 99.8% of sampled element annotations correct. Annotation views, refinement, and quality assessment are detailed in Appendices A, B, and D.

3.3 Interaction Recording and Instruction Construction

DeskForge explores scenes by sampling actionable elements and executing clicks within their estimated visible regions. Exploration is driven by element selection rather than predefined task goals, and records both changed and unchanged outcomes. Each transition (S_t, a_t, S_{t+1}) links before and after screen observations with the executed action, target metadata, and an aggregate effect summary. The associated element records also allow changes in content, geometry, visibility, and control state to be derived separately.

To construct grounding examples, we prompt a vision-language model with the before and after observations and the target context to write a single-step instruction. Instructions are refined and filtered to describe a goal whose target can be identified from the before screenshot, without referring to coordinates or annotation markers. The resulting grounding example pairs that screenshot and instruction with the recorded target. The successor observation informs instruction construction but is not supplied to the grounding model. This yields instruction-grounding supervision while retaining the underlying transitions for other learning objectives.

Acquisition details and instruction-generation prompts are provided in Appendices B.1 and G.

4 The DeskForge-1M Dataset

4.1 Scale and Composition

DeskForge-1M contains 1.21M annotated desktop observations and 159.7M element instances, spanning 19 applications, seven appearance presets, and seven display resolutions. The applications include browsers, editors, file managers, and desktop utilities, while resolutions range from 1366×768

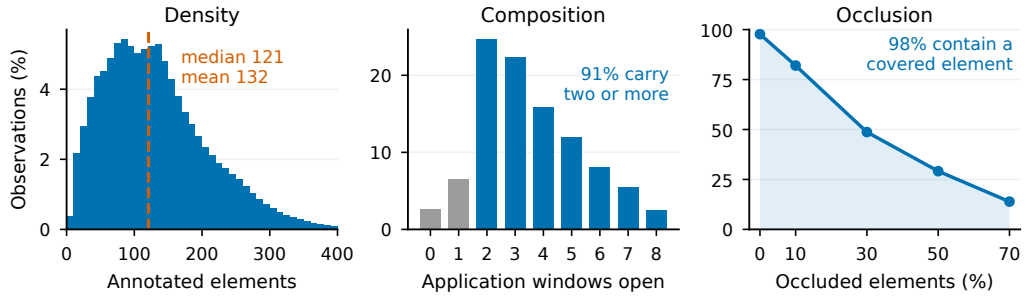


Figure 3: **Corpus composition of DeskForge-1M.** Over all 1.21M observations: annotated elements per observation (truncated at 400; maximum 1,161), number of applications, and share of observations with more than $x\%$ of elements occluded.

to 3840×2160. Observations are grouped into approximately 324K scenes and include individual captures and interaction episodes, with 917K recorded click transitions. Element counts are taken from the visible annotation view, after removing redundant entries, and count each element once per observation. The total is about 7.6 times that of the largest resource in Table 1.

As Figure 3 shows, observations are dense and cluttered: they contain 132 annotated elements on average (median 121), 90.9% record at least two applications, and 97.7% contain occluded elements, with more than 30% of elements affected in nearly half. These statistics describe the desktop context rather than the visibility of the targets selected for grounding.

4.2 Annotation content and supervision views

Each observation pairs a screenshot with dense element records and a ScreenTag representation. The element records give each element's geometry, text, type, window ownership, and interaction properties, and store both its full extent and its visible region, the fragments left exposed by overlapping windows; ScreenTag serializes the same screen as compact markup that preserves the element hierarchy (Section 3). Transition records add the executed click, its target, the successor observation, and a summary of what changed.

To introduce linguistic variation into the grounding supervision, we use Qwen3.6-27B to synthesize natural-language instructions from programmatically recorded clicks, target context, and before/after screenshots. The model is prompted to express the same single-step user goal in concise, standard, and more contextual formulations, varying wording and detail while preserving the intended target. Full prompts are provided in Appendix G.

The grounding experiments use a fixed 200K-example training view, with each example pairing a before screenshot and one synthesized instruction with the recorded click target. Element detection (Section 5.4) instead uses the dense annotations of its training screens.

In a human audit, 97.8% of uniformly sampled instructions were judged sound (Appendix D).

4.3 Evaluation splits

Training, validation, and test partitions are assigned at the scene level, keeping interaction frames and recaptures of the same scene together. Four test conditions distinguish new scenes with represented desktop attributes (New Scenes) from scenes containing held-out applications (App), a reserved appearance preset (Theme), or a reserved display resolution (Resolution).

The application split holds out GNOME System Monitor, Pluma, and Xarchiver. Assignment considers every application recorded across a scene, rather than only the application containing a selected target, so a held-out application cannot enter training through another frame. The appearance split reserves the Quartz Night Nord preset, and the resolution split reserves 2880×1800. These partitions evaluate generalization to desktop configurations not represented in training. The full application inventory, corpus distributions, and split specifications are provided in Appendix A.

Table 2: **Grounding accuracy on the four held-out conditions of DeskForge-1M (%).** Public checkpoints provide reference performance. The lower block directly compares each backbone before and after fine-tuning on DeskForge-1M; fine-tuned results are shown in bold.

Model	Variant	Held-out condition				Mean
		New Scenes	Theme	App	Resolution	
Public VLMs and CUAs						
OS-Atlas-Pro-7B		47.61	47.27	45.58	32.10	43.14
EvoCUA-8B		50.49	45.59	46.65	44.93	46.91
UI-TARS-1.5-7B		67.68	64.96	66.68	61.67	65.25
GUI-Owl-1.5-32B		70.87	62.19	66.68	64.65	66.10
UGround-V1-7B		69.08	64.61	67.18	63.82	66.17
ScaleCUA-7B		72.74	68.28	70.75	63.62	68.85
UI-MOPD-8B		73.04	67.12	70.49	68.07	69.68
GroundNext-7B		73.57	67.67	72.10	68.87	70.55
Gemma4-31B		75.26	71.00	73.57	63.13	70.74
UI-Venus-2-9B		81.45	77.44	79.40	77.40	78.92
Fine-tuning across VLMs						
Gemma4-E4B	Base	16.27	12.10	14.15	6.56	12.27
	+ DeskForge-1M	**89.01**	**85.36**	**84.50**	**81.62**	**85.12**
UI-R1-3B	Base	41.34	40.24	35.56	34.90	38.01
	+ DeskForge-1M	**88.25**	**84.28**	**83.77**	**82.07**	**84.59**
InternVL3.5-8B	Base	65.00	59.28	61.50	47.62	58.35
	+ DeskForge-1M	**87.37**	**83.43**	**83.77**	**71.10**	**81.42**
Qwen3.5-4B	Base	78.56	74.03	76.11	76.34	76.26
	+ DeskForge-1M	**90.14**	**87.36**	**87.50**	**85.15**	**87.54**

5 EXPERIMENTS

We fine-tune vision-language models on DeskForge-1M and ask three questions: does grounding improve on held-out desktop configurations, do the gains transfer to external GUI benchmarks, and does long-horizon task completion increase when only the action model changes within a fixed planner agent? We then test whether the dense annotations also support whole-screen element detection. These comparisons assess the utility of the generated data beyond the scenes and individual interactions used for fine-tuning.

Experimental setup. We fine-tune Qwen3.5-4B (Qwen Team, 2026a), Gemma4-E4B IT (Gemma Team, 2026), InternVL3.5-8B (Wang et al., 2025a), and UI-R1-3B (Lu et al., 2026) on the 200K single-target grounding view of DeskForge-1M using LoRA (Hu et al., 2022). The first three are general-purpose VLMs; UI-R1 provides an additional test starting from a GUI-specialized checkpoint. Within each base/fine-tuned pair, prompts, image preprocessing, decoding, and scoring are held fixed, while model-specific input and output conventions are retained.

Grounding accuracy measures whether a predicted point falls within the annotated target region, with invalid outputs counted as incorrect. DeskForge uses the annotated visible regions; external evaluation uses the corresponding benchmark annotations and scoring settings. Unless otherwise specified, fine-tuning uses eight NVIDIA H100 GPUs for one epoch, with an effective global batch size of 64. Full training settings and model-specific exceptions are provided in Appendix C.

5.1 GROUNDING ACROSS HELD-OUT DESKTOP CONDITIONS

We first evaluate the four scene-level conditions defined in Section 4: New Scenes, Theme, App, and Resolution. Table 2 reports each condition separately and the mean over the four conditions. The matched base/fine-tuned pairs measure the benefit of fine-tuning, while public checkpoints (Wu et al., 2025; Xue et al., 2026; Qin et al., 2025; Xu et al., 2026; Gou et al., 2025; Liu et al., 2026b;

Table 3: **GUI grounding accuracy on five external benchmarks (%).** Rows marked **+ DeskForge-1M** denote fine-tuning on our data.

Model	Training	SS-Pro	SS-v2	OSWorld-G	UI-Vision	MMB-GUI
Gemma4-E4B	Base	1.83	46.86	10.64	3.28	31.39
	+ DeskForge-1M	**21.95**	**79.56**	**44.15**	**13.87**	**58.90**
UI-R1-3B	Base	14.48	81.92	34.04	12.09	56.84
	+ DeskForge-1M	**27.20**	**83.96**	**43.97**	**15.15**	**65.41**
InternVL3.5-8B	Base	18.72	83.33	44.15	17.50	66.58
	+ DeskForge-1M	**23.78**	**84.98**	**44.33**	**17.55**	**67.20**
Qwen3.5-4B	Base	30.17	80.11	51.24	17.24	61.83
	+ DeskForge-1M	**41.68**	**90.33**	**61.35**	**28.77**	**76.82**

Lian et al., 2026; Feizi et al., 2026; Gemma Team, 2026; Venus Team et al., 2026) provide reference performance under model-specific, rather than equal-compute, inference protocols (Appendix C.2).

All four models improve in every held-out condition (Table 2). Qwen's mean accuracy rises from 76.26% to 87.54%, while UI-R1 improves from 38.01% to 84.59%. The fine-tuned Gemma and InternVL models reach 85.12% and 81.42%, respectively.

The gains hold for held-out applications and the reserved appearance and resolution, so the supervision transfers beyond the desktop configurations seen in training. They are also largest where desktop scenes are hardest (Appendices E.2 and E.3). As the number of applications on screen grows from one to four or more, fine-tuned Qwen loses only a few points, whereas its base model and UI-Venus-2-9B, the strongest reference model, degrade steadily; the gap to the base widens from 9.7 to 14.3 points. The gap likewise grows from 10.9 to 13.7 points as the target's visible-area loss rises to 35%.

5.2 Transfer to external grounding benchmarks

To test whether the benefit extends beyond DeskForge-1M, we evaluate on ScreenSpot-Pro, ScreenSpot-v2, OSWorld-G, UI-Vision, and MMBench-GUI (Li et al., 2025; Wu et al., 2025; Xie et al., 2025; Nayak et al., 2025; Wang et al., 2026b). None is used for fine-tuning, and all differ from DeskForge-1M in platform coverage, application inventory, and annotation source: they are human-annotated captures of real usage, spanning Windows, macOS, Linux, mobile, and web screens, and include professional applications such as CAD and creative tools that DeskForge does not run.

All four models obtain higher accuracy on all five external benchmarks (Table 3). Qwen gains 11.51 percentage points on ScreenSpot-Pro and 10.11 points on OSWorld-G, while Gemma's mean over the five benchmarks increases by 24.9 points. UI-R1 improves from 14.48% to 27.20% on ScreenSpot-Pro and from 39.87% to 47.14% in mean accuracy over the five benchmarks. InternVL also improves across all five benchmarks, with a more modest mean gain of 1.51 points. The ScreenSpot-v2 breakdown (Appendix E.1, Table 11) shows where transfer occurs: Qwen and UI-R1 gain most on desktop interfaces (+14.37 and +5.69 points), yet also improve on mobile (+10.77 and +0.60) and web (+6.41 and +0.91), although DeskForge-1M contains no mobile screens.

These results show that supervision generated through controlled desktop execution transfers to GUI collections beyond the DeskForge environment. The gains for UI-R1 are particularly informative: DeskForge-1M benefits not only general-purpose VLMs, but also a model already specialized for GUI interaction. This demonstrates its value both for developing grounding capabilities and for further improving an existing computer-use model.

5.3 Long-horizon task completion under a fixed planner

We next test whether single-target grounding supervision improves task completion during extended interaction. A fixed Qwen3.6-27B planner (Qwen Team, 2026b) selects action types, text or keys, and coordinate-free target descriptions; the action model supplies the locations needed to execute those decisions. Only the action-model weights change within each base/fine-tuned comparison. We

Table 4: **Long-horizon task completion with a fixed Qwen3.6-27B planner.** Number of solved tasks on the custom 119-task WebArena-Infinity panel and the 100-task OpenApps longer-horizon set. **+DeskForge-1M** denotes fine-tuning of the model on our data.

Model	WebArena-Infinity (119)		OpenApps (100)	
	Base	+DeskForge-1M	Base	+DeskForge-1M
Gemma4-E4B	5	**40**	2	**12**
UI-R1-3B	30	**37**	7	**12**
InternVL3.5-8B	37	**44**	4	**11**
Qwen3.5-4B	31	**50**	3	**15**

evaluate a custom 119-task WebArena-Infinity panel and the 100-task OpenApps longer-horizon set (Zhou, 2026; Ullrich et al., 2026). Task success is determined by the environments' state-based checks; task selection, action budgets, and agent protocols are specified in Appendix C.4.

All four fine-tuned action models solve more tasks in both environments (Table 4). Qwen improves from 31 to 50 tasks on WebArena-Infinity and from 3 to 15 on OpenApps. Gemma shows the largest increase on the Infinity panel, from 5 to 40 tasks. InternVL solves seven additional tasks in each environment, while the GUI-specialized UI-R1 solves seven additional Infinity tasks and five additional OpenApps tasks.

Although fine-tuning uses single grounding examples rather than task trajectories, the fine-tuned action models execute the fixed planner's decisions more successfully over successive observations (Appendix E.5), even in browser environments unlike our composed desktops. Improving grounding alone thus raises task completion without planner training, pointing to a practical use of DeskForge-1M: strengthening the visual action-execution component of a computer-use agent.

5.4 Dense element detection

The experiments above use one annotated target per screenshot. Dense annotations additionally enable learning to parse the entire screen, localizing every visible element rather than a single instructed target. We fine-tune an RT-DETRv4-L detector on these annotations and evaluate cross-dataset transfer on GroundCUA. Our detector outperforms the OmniParser v2 detector and ScreenParse YOLO11L on all three localization metrics, reaching 62.17% group F1 compared with 58.55% and 57.59%, respectively. These results show that detectors trained on dense screen annotations from DeskForge localize elements on screens beyond DeskForge, a core step of screen parsing that complements instruction-conditioned grounding. The comparison and metric definitions are provided in Appendix E.4.

6 Conclusion and Future Work

DeskForge makes the composition and exploration of running desktop applications a controllable source of structured supervision. The resulting DeskForge-1M contains 1.2M annotated observations and 159.7M element instances, together with recorded click transitions. Fine-tuning improves grounding across held-out desktop conditions and five external benchmarks for both general-purpose and GUI-specialized models, and increases long-horizon task completion under a fixed planner. Dense element detection demonstrates a complementary use of the same corpus. These results indicate the value of controllably generated desktop data for learning and applying visual grounding.

Limitations and future directions. Current acquisition uses a Linux backend and relies on applications with suitable accessibility support. Broader application integration, native-platform backends, and complementary visual annotation would extend coverage to interfaces that expose less structure. Our VLM training focuses on single-target grounding, leaving joint learning from interface structure and recorded state changes as a natural next step. In particular, the retained transitions could support forward and inverse dynamics objectives for learning about action outcomes. Developing goal-conditioned multi-application tasks with state-based verification would further extend DeskForge toward online training of long-horizon computer-use agents.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data (Qwen3.6-27B writes and refines the grounding instructions) and to implement parts of the pipeline and evaluation code. We have not used them to develop theoretical models, formulate or prove mathematical claims, propose hypotheses, or design experiments; translation and qualitative data analysis are not applicable. Additionally, we used generative AI tools to edit code and improve readability. We have reviewed all AI-assisted work: code was tested before use, and every reported number is checked automatically against its evaluation output. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

DeskForge-1M is meant to improve visual grounding for computer-use agents. More reliable grounding can also make unsanctioned automation cheaper, and our sandboxed, fixed-planner results do not show that agents operate safely without supervision. Screenshots come from isolated virtual desktops with staged content.

REPRODUCIBILITY STATEMENT

Sections 3-4 and Appendices A-B describe the environment, annotation pipeline, seeded scene planning, and splits. Appendix C gives training hyperparameters, per-model inference settings, and the evaluation and task-completion protocols. Appendix E.4 defines the detection metrics and operating points, and Appendix G reproduces all prompts. We will release the code, scene configurations, DeskForge-1M with its splits, the fine-tuned models, and the evaluation scripts.

REFERENCES

AI2. MolmoPoint-GUISyn dataset, 2026. URL https://huggingface.co/datasets/allenai/MolmoPoint-GUISyn. Allen Institute for AI. Official artifact; accessed 2026-09-07.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In *ACL*, 2024. URL https://github.com/njucckevin/SeeClick.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *UIST*, 2017. URL https://www.interactionmining.org/archive/rico.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a Generalist Agent for the Web. In *NeurIPS Datasets and Benchmarks*, 2023. URL https://papers.nips.cc/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf.

Aarash Feizi, Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Kaixin Li, Rabiul Awal, Xing Han Lù, Johan Obando-Ceron, Juan A. Rodriguez, Nicolas Chapados, David Vazquez, Adriana Romero-Soriano, Reihaneh Rabbany, Perouz Taslakian, Christopher Pal, Spandana Gella, and Sai Rajeswar. Grounding Computer Use Agents on Human Demonstrations. In *ICLR*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0961fffb7adb81388b9b24ab6a3ff7ab-Abstract-Conference.html.

Sofiya Garkot, Maksym Shamrai, Ivan Synytsia, and Mariya Hirna. GUIrilla: A Scalable Framework for Automated Desktop UI Exploration, 2025. URL https://arxiv.org/abs/2510.16051.

Gemma Team. Gemma 4 Technical Report, 2026. URL https://arxiv.org/abs/2607.02770.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents. In *ICLR*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/4ca0e369689dadb25a5345ba9755ad6f-Abstract-Conference.html.

A. Said Gurbuz, Sunghwan Hong, Ahmed Nassar, Marc Pollefeys, and Peter Staar. ScreenParse: Moving Beyond Sparse Grounding with Complete Screen Parsing Supervision. In *ICML*, 2026. URL https://github.com/Saidgurbuz/screenparse.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Xiangru Jian, Shravan Nayak, Kevin Qinghong Lin, Aarash Feizi, Kaixin Li, Patrice Bechard, Spandana Gella, and Sai Rajeswar. CUA-Suite: Massive Human-annotated Video Demonstrations for Computer-Use Agents, 2026. URL https://arxiv.org/abs/2603.24440.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP*, 2023. URL https://arxiv.org/abs/2309.06180.

Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding. In *ICML*, 2023. URL https://proceedings.mlr.press/v202/lee23g.html.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use. In *ACM Multimedia*, pp. 8778–8786, 2025. URL https://doi.org/10.1145/3746027.3755688.

Niu Lian, Tongbo Chen, Zhehao Yu, Chengzhen Duan, Fazhan Liu, Hui Liu, Pei Fu, Jian Luan, Heng Qu, Shu-Tao Xia, and Jinpeng Wang. UI-MOPD: Multi-Platform On-Policy Distillation for Unified GUI Agents, 2026. URL https://arxiv.org/abs/2607.04425.

Zijun Liao, Yian Zhao, Xin Shan, Yu Yan, Chang Liu, Lei Lu, Xiangyang Ji, and Jie Chen. RT-DETRv4: Painlessly Furthering Real-Time Object Detection with Vision Foundation Models. In *ECCV*, 2026. URL https://doi.org/10.1007/978-3-032-37029-7_7.

Xiangyan Liu, Kaixin Li, Haonan Wang, Biao Wu, Meng Fang, Longxu Dou, Chao Du, Michael Qizhe Shieh, and Tianyu Pang. Scaling GUI Agents with Visual State Transitions, 2026a. URL https://arxiv.org/abs/2607.24112.

Zhaoyang Liu, Jingjing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, Shenglong Ye, Qingyun Li, Zeyue Tian, Gen Luo, Xiangyu Yue, Biqing Qi, Kai Chen, Bowen Zhou, Yu Qiao, Qifeng Chen, and Wenhai Wang. ScaleCUA: Scaling Open-Source Computer Use Agents with Cross-Platform Data. In *ICLR*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/666c1861d709bd84e20b6e0e02a2c223-Abstract-Conference.html.

Yadong Lu, Thomas Dhome-Casanova, Jianwei Yang, and Ahmed Awadallah. OmniParser V2: Turning Any LLM into a Computer Use Agent. Microsoft Research, 2025. URL https://www.microsoft.com/en-us/research/articles/omniparser-v2-turning-any-llm-into-a-computer-use-agent/.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Pengxiang Zhao, Guangyi Liu, Guanjing Xiong, and Hongsheng Li. UI-R1: Enhancing Efficient Action Prediction of GUI Agents by Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 17608–17616, 2026. doi: 10.1609/aaai.v40i21.38816. URL https://ojs.aaai.org/index.php/AAAI/article/view/38816.

Jian Mu, Chaoyun Zhang, Chiming Ni, Lu Wang, Bo Qiao, Kartik Mathur, Qianhui Wu, Yuhang Xie, Xiaojun Ma, Mengyu Zhou, Si Qin, Liqun Li, Yu Kang, Minghua Ma, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. GUI-360°: A Comprehensive Dataset and Benchmark for Computer-Using Agents, 2025. URL https://arxiv.org/abs/2511.04307.

Ahmed Nassar, Matteo Omenetti, Maksym Lysak, Nikolaos Livathinos, Christoph Auer, Lucas Morin, Rafael Teixeira de Lima, Yusik Kim, A. Said Gurbuz, Michele Dolfi, and Peter W. J. Staar. SmolDocling: An Ultra-Compact Vision-Language Model for End-to-End Multi-Modal Document Conversion. In *ICCV*, pp. 21972–21983, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Nassar_SmolDocling_An_ultra-compact_vision-language_model_for_end-to-end_multi-modal_document_conversion_ICCV_2025_paper.html.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A. Rodriguez, Montek Kalsi, Nicolas Chapados, M. Tamer Özsu, Aishwarya Agrawal, David Vazquez, Christopher Pal, Perouz Taslakian, Spandana Gella, and Sai Rajeswar. UI-Vision: A Desktop-centric GUI Benchmark for Visual Perception and Interaction. In *ICML*, 2025. URL https://proceedings.mlr.press/v267/nayak25a.html.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering Automated GUI Interaction with Native Agents, 2025. URL https://arxiv.org/abs/2501.12326.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-4B. Official artifact (Qwen3.5-4B); accessed 2026-09-07.

Qwen Team. Qwen3.6-27B: Flagship-Level Coding in a 27B Dense Model, 2026b. URL https://huggingface.co/Qwen/Qwen3.6-27B. Official artifact; accessed 2026-09-07.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. OS-Genesis: Automating GUI Agent Trajectory Construction via Reverse Task Synthesis. In *ACL*, 2025. URL https://aclanthology.org/2025.acl-long.277/.

Karen Ullrich, Jingtong Su, Claudia Shi, Arjun Subramonian, Amir Bar, Ivan Evtimov, Nikolaos Tsilivis, Randall Balestriero, Julia Kempe, and Mark Ibrahim. OpenApps: Simulating Environment Variations to Measure UI-Agent Reliability. In *ICLR*, 2026. URL https://openreview.net/pdf?id=cj1MAx7lKs.

Venus Team, Zhuohan Cai, Haoxing Chen, Jiaxuan Chen, Weizhi Chen, Changlong Gao, Zhangxuan Gu, Yuan Guo, Yusong Hu, Jianrong Jiang, Jianguo Li, Runze Li, Jinzhen Lin, Zhenyu Ma, Changhua Meng, Han Peng, Xinyu Qiu, Shuheng Shen, Zhongyi Shui, Weiqiang Wang, Ming Wen, Zhuoer Xu, Hang Yan, Kaiwen Yang, Ruilin Yao, Nanjun Yu, Zhengwen Zeng, Lianrui Zhang, Yunzhu Zhang, Zhe Zhao, and Beitong Zhou. UI-Venus-2 Technical Report, 2026. URL https://arxiv.org/abs/2609.00028.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. CUA-Gym: Scaling Verifiable Training Environments and Tasks for Computer-Use Agents, 2026a. URL https://arxiv.org/abs/2605.25624.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. InternVL3.5: Advancing Open-Source Multimodal Models in Versatility, Reasoning, and Efficiency, 2025a. URL https://arxiv.org/abs/2508.18265.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin

Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y. Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open Foundations for Computer-Use Agents. In *NeurIPS*, 2025b. URL https://papers.nips.cc/paper_files/paper/2025/hash/cc7ae529e945226b0d52ea4ac478c4f3-Abstract-Conference.html.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, Weiyun Wang, Xiangyu Zhao, Jixuan Chen, Haodong Duan, Tianbao Xie, Chenyu Yang, Shiqian Su, Yue Yu, Yanting Zhang, Xiangyu Yue, Weijie Su, Xizhou Zhu, Wei Shen, Jifeng Dai, and Wenhai Wang. MMBench-GUI: A Unified Hierarchical Evaluation Framework for Multi-Platform GUI Agents. In *CVPR*, pp. 6239–6248, 2026b. URL https://openaccess.thecvf.com/content/CVPR2026/html/Wang_MMBench-GUI_A_Unified_Hierarchical_Evaluation_Framework_for_Multi-Platform_GUI_Agents_CVPR_2026_paper.html.

Christian Wolf and Jean-Michel Jolion. Object Count/Area Graphs for the Evaluation of Object Detection and Segmentation Algorithms. *International Journal on Document Analysis and Recognition*, 8(4):280–296, 2006. URL https://doi.org/10.1007/s10032-006-0014-0.

Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P. Bigham. WebUI: A Dataset for Enhancing Visual UI Understanding with Web Semantics. In *CHI*, 2023. URL https://arxiv.org/abs/2301.13280.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-ATLAS: A Foundation Action Model for Generalist GUI Agents. In *ICLR*, 2025. URL https://openreview.net/pdf?id=n9PDaFNi8t.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *NeurIPS Datasets and Benchmarks*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. Scaling Computer-Use Grounding via User Interface Decomposition and Synthesis. In *NeurIPS Datasets and Benchmarks*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/22c868099177ee278eb7baccec649f35-Abstract-Datasets_and_Benchmarks_Track.html.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. Mobile-Agent-v3.5: Multi-platform Fundamental GUI Agents, 2026. URL https://arxiv.org/abs/2602.16855.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. EvoCUA: Evolving Computer Use Agents via Learning from Scalable Synthetic Experience, 2026. URL https://arxiv.org/abs/2601.15876.

Haoren Zhao, Tianyi Chen, and Zhen Wang. WinDeskGround: A Benchmark for Robust GUI Grounding in Complex Multi-Window Desktop Environments. In *ICML*, 2026. URL https://icml.cc/virtual/2026/poster/66756.

Shuyan Zhou. WebArena-Infinity: Generating Browser Environments with Verifiable Tasks at Scale, 2026. URL https://github.com/web-arena-x/webarena-infinity. Official artifact; accessed 2026-09-07.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *ICLR*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

Table 5: **Information stored for each element, observation, and transition.** Element records and ScreenTag accompany every screenshot; transition fields link consecutive observations of an interaction episode.

Level	Stored information
Element	Type (55 ScreenTag classes) and kind (26 categories, Table 6); accessibility role; accessible name, and visible text with its verification status; visible fragments and, where retained, the source extent; occlusion state (none, partial, hidden); owning application and window, stacking index, parent and child links, and reading order; interaction properties (actionable, enabled, focusable, editable, checkable and checked, expandable and expanded, selected) and a click point inside the visible region; an identifier stable within an episode.
Observation	Screenshot and ScreenTag serialization; resolution, appearance preset, panel variant, desktop profile, window layout, and scene seed; launched applications; window stack with titles, extents, and stacking order; element and occlusion counts; near-duplicate and no-op flags; split and generator provenance.
Transition	Before and after observations; click point; target identifier, role, name, text, and box in the before frame; element-level effects (appeared, disappeared, moved, text or state changed, newly occluded, revealed) with an aggregate change magnitude; training eligibility and exclusion reasons.

Table 6: **Element kinds.** Each element also retains its accessibility role and one of 55 ScreenTag classes. Kinds merge classes that are not visually distinguishable; for example, check boxes, radio buttons, switches, and toggle buttons form *toggle*, whose state is stored separately.

Group	Kinds
Structure	window, titlebar, menubar, toolbar, statusbar, sidebar
Navigation	menu, menuitem, tab, link
Controls	button, toggle, textinput, select, slider, scrollbar, progress
Content	text, heading, image, icon, canvas
Collections	list, listitem, table, cell

A Dataset Specification

A.1 Observations, annotations, and supervision views

DeskForge-1M contains 1,207,368 annotated observations and 159,723,780 retained element instances. The element total counts entries in the visible, redundancy-reduced annotation view separately for each observation, including recurring elements across interaction frames. The mean and median are 132.29 and 121 elements per observation.

The observation payload contains the screenshot, element annotations, ScreenTag serialization, and capture record; Table 5 lists the information stored for each element, observation, and transition, and Table 6 the element kinds. The environment additionally retains broader extraction and amodal records; these are separate from the main observation payload.

Because every screenshot, annotation, and transition is stored, DeskForge-1M is used without executing the environment: training and evaluation require no virtual machines, application installation, or environment resets, and results on the held-out splits are reproducible from the stored files alone. When a study needs states beyond those recorded, DeskForge can generate them.

A.2 Scene-level splits

All observations and recaptures belonging to a scene share one split. Application assignment uses the union of applications recorded throughout the scene. The reserved applications are GNOME System Monitor, Pluma, and Xarchiver; the reserved appearance is Quartz Night Nord; and the reserved resolution is 2880×1800. Scenes matching multiple holdouts are assigned in application, appearance, then resolution priority. The New Scenes condition, stored as `test_id`, holds out scene instances with represented application, appearance, and resolution attributes. Table 7 reports observations, element instances, click transitions, and grounding examples in each partition. Excluding observations

Table 7: **DeskForge-1M by split.** Element instances refer to the visible, redundancy-reduced view. Each grounding example pairs one instruction with its recorded target; the 200,000 training examples are sampled from 551,651 training-split instructions.

Split	Observations	Element instances	Click transitions	Grounding examples
Train	999,494	126,746,711	760,850	200,000
Validation	11,279	1,448,957	8,667	6,281
New Scenes	22,550	2,826,805	17,383	12,589
App	43,206	8,151,157	32,848	22,729
Theme	51,627	7,991,478	38,671	27,175
Resolution	79,212	12,558,672	58,792	43,210
Total	**1,207,368**	**159,723,780**	**917,211**	**311,984**

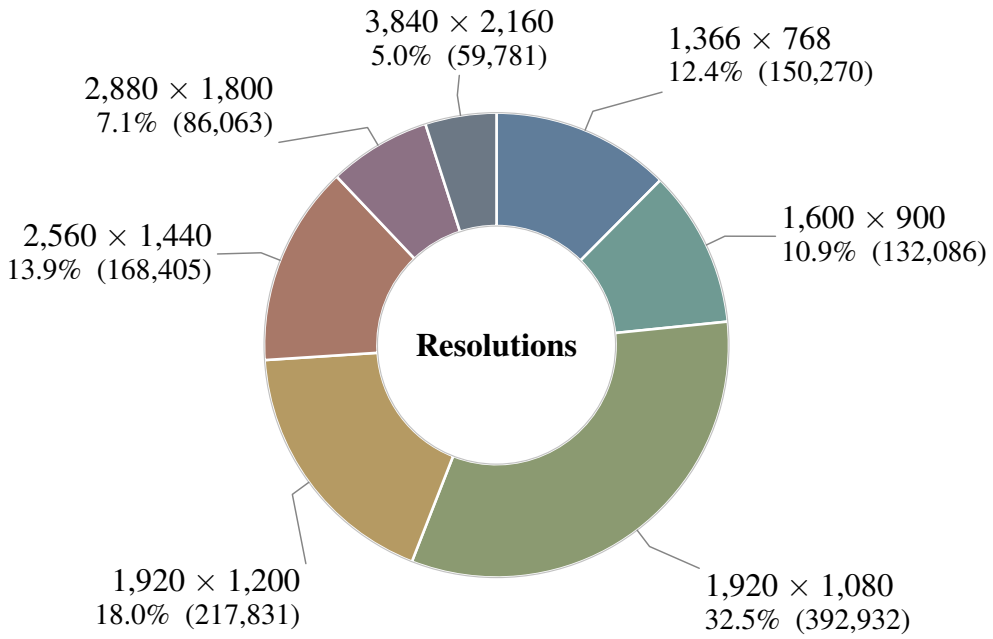


Figure 4: **Display resolutions in DeskForge-1M.** Observation counts and proportions across the seven resolutions. All 1,207,368 observations are included, including successive episode frames.

marked as near-duplicates or no-op frames leaves a filtered state view of 1,067,799 observations and 142,537,053 element instances, from which the human audit samples (Appendix D).

A.3 Application and display composition

Figures 5 and 4 show application co-occurrence and the distribution of display resolutions. Application co-occurrence is measured within individual captures and then averaged at the scene level, so longer interaction episodes do not receive greater weight simply because they contain more frames.

Corpus-level statistics. Figure 3 weights observations equally. Its application count is the size of the recorded launched-application list. Its occlusion proportion is the fraction of elements entering the occlusion pass that are clipped or removed by that pass; the curve reports observations exceeding each threshold. This proportion has a different denominator from the retained-element density and does not measure the visibility of the selected grounding target.

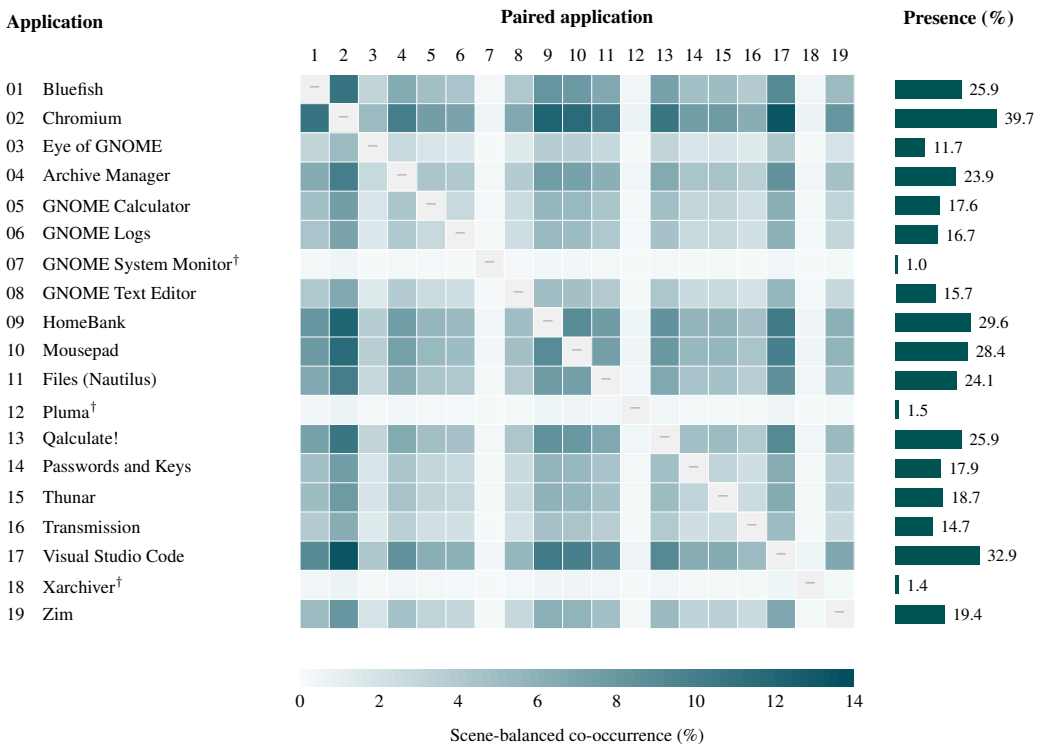


Figure 5: **Application inventory and co-occurrence.** Each matrix entry averages the within-scene fraction of captures recording both applications; bars show the corresponding marginal application presence. The analysis covers 1,175,179 captures from 315,705 scenes with nonempty application records. Application presence is taken from capture metadata, not from visible-window counts. †Reserved applications.

B Environment and Data Generation

B.1 Execution and acquisition

Each acquisition worker runs an isolated Linux desktop session using Xvfb, a private D-Bus session, xfwm4, a MATE panel, and Caja desktop components. Applications are initialized with staged content and arranged according to a scene specification. Application integration requires reliable launch behavior and accessibility coverage of the principal interface content; for example, candidates whose file panes publish no list or table roles, or whose main view is a single drawing surface, were not admitted. Seeded planning reproduces scene specifications for a fixed configuration and application pool.

Desktop acquisition is CPU-only and runs as a batch job array on a shared cluster; sessions share nothing but the file system. The largest campaign ran 300 jobs with 24 CPU cores each (7,200 cores) and five desktop sessions per job, for 1,500 concurrent sessions. Its measured throughput of 84,807 observations per hour corresponds to about 9.4 hours for its 795,725 observations. Because sessions are independent and scenes are planned before capture, acquisition scales out without coordination: each job processes a disjoint slice of the plan, and capacity grows with the number of sessions. Repeated interaction within a scene further amortizes the cost of initializing it. These figures describe desktop acquisition, separately from instruction generation and model training.

B.2 Appearance presets and source assets

The seven presets coordinate application themes, icons, window decorations, fonts, and panel or dock layouts. Classic-Linux and Ubuntu-like presets use Adwaita and Papirus assets with different desktop layouts. The Windows-inspired preset combines Win11 toolkit/window themes and icons.[1] Four macOS-inspired variants combine MacTahoe light, glass, dark, or Nord styling with WhiteSur icons and cursors.[2] These assets style running applications and the surrounding desktop; all presets retain the Linux execution backend.

B.3 Annotation views and refinement

Accessibility-reported actions and states in the element records (Table 5) are distinct from the outcomes recorded after executing a click.

The unfiltered representation preserves the broader captured hierarchy. The visible, redundancy-reduced representation supports parsing and ScreenTag serialization, retaining structural containers where needed. The amodal representation augments visible records with retained fully hidden elements and their visibility states; it is not a duplicate of the entire unfiltered hierarchy.

Geometry refinement handles coordinate offsets, window boundaries, and popup coordinate frames. Ancestor and viewport clipping, higher windows, and detected overlays determine the exposed regions. Selected window controls missing from accessibility trees are recovered from window geometry. Character geometry and pixel checks separate text rendered on screen from accessible names, including names attached to unlabeled icons.

Visible geometry. An element's visible support V_e is a union of axis-aligned rectangles. Where a source extent B_e is retained, it describes the element after initial geometry refinement, and its saved visibility loss is

$$\rho_e = 1 - \frac{\text{area}(V_e)}{\text{area}(B_e)}. \tag{1}$$

This measure includes clipping and occlusion. The visible-fragment union, rather than an enclosing rectangle, defines visible-hit grounding targets. Some synthesized elements lack a source extent; those entries do not define a paired visible/source-extent example.

[1] https://github.com/yeyushengfan258/Win11-gtk-theme; https://github.com/yeyushengfan258/Win11-icon-theme.

[2] https://github.com/vinceliuice/MacTahoe-gtk-theme; https://github.com/vinceliuice/WhiteSur-icon-theme; https://github.com/vinceliuice/WhiteSur-cursors.

ScreenTag uses a 0–500 normalized coordinate grid with nested structure and text/state tokens. Its type vocabulary is distinct from the 26 element kinds (Table 6) used for detection.

Automated quality control. Every capture passes structural audits before release. Captures are rejected when element-to-window ownership is inconsistent, visible fragments are missing or extend beyond their element, or the annotation is implausibly sparse (too few elements, low type diversity, a shallow hierarchy, or low coverage). Of 1,266,471 source captures, 59,103 (4.7%) failed and are excluded from DeskForge-1M. Pipeline revisions were evaluated with pixel-level measurements that count errors in both directions, since reducing one kind by introducing the other is not an improvement. False positives are annotations over blank pixels, text claimed where no glyph ink is drawn (phantom text), and text boxes displaced from their glyphs (position drift); false negatives are drawn pixels inside a window that no annotation covers (uncovered ink). Human assessment is reported separately in Appendix D.

B.4 Recorded interactions and instruction construction

Exploration selects actionable elements rather than following predefined task goals. Role priorities and target size guide sampling, while elements with low observed interaction yield remain selectable. Each executed click links a before observation to a successor observation, with target metadata and an aggregate effect summary. Element-level changes are derived separately from retained state records. In the capture format, the action is stored with the observation it produced; its coordinates refer to the preceding screenshot.

Instruction synthesis and semantic refinement use batched vLLM inference (Kwon et al., 2023). The instruction model described in Section 4 receives the before screenshot, a target-marked copy, a target crop, the successor screenshot, and coordinate-free target context. It is prompted to generate concise, standard, and detailed-contextual formulations of the same single-step goal, or to decline when that goal cannot be inferred. Refinement checks the draft against the visual evidence and requests a goal identifiable from the unmarked before screenshot. Filtering removes malformed outputs and prohibited references to coordinates or annotation markers. The reported training view uses generated, refined, and filtered instructions; round-trip grounding verification was not a selection filter for those examples. Full templates appear in Appendix G.

C Training and Evaluation Protocols

C.1 Grounding fine-tuning

Qwen3.5-4B, Gemma4-E4B IT, InternVL3.5-8B, and UI-R1-3B are fine-tuned with LoRA using the 200K single-target grounding training view. Language-model parameters receive the adapters; the vision encoder and multimodal alignment modules remain frozen. UI-R1 starts from its released GUI-specialized checkpoint and retains its native action-answer convention. Table 8 summarizes the common training settings.

Grounding evaluation uses the final training checkpoint. For Qwen, the held-out and external grounding tables use the same checkpoint; the browser interaction study uses a separate training run of the same 200K-example recipe.

C.2 Inference and paired comparisons

The evaluation pipeline uses batched vLLM inference with model-specific image processors, prompts, and response parsers. For each base/fine-tuned pair, the evaluation examples, image preparation, prompt, decoding settings, and scorer are fixed. Table 9 gives the settings for the four fine-tuned model families. Coordinates are converted to the original screenshot frame before scoring.

The additional public checkpoints in Table 2 retain their model-specific inference protocols. UI-Venus uses a 12,845,056-pixel native resize cap and 256 greedy response tokens; GUI-Owl-32B uses 3,072,000 pixels and 512 greedy tokens. UI-MOPD uses a 7,840,000-pixel initial resize cap and up to 8,192 response tokens with temperature 0.7, top-p 0.8, and top-k 20. Gemma4-31B uses 1,120 native image tokens and up to 8,192 reasoning tokens with temperature 1, top-p 0.95, and top-k

Table 8: **Common VLM fine-tuning settings.** Settings for the 200K grounding recipe; image processing and output conventions are model-specific (Table 9).

Setting	Value
Training examples	200,000 single-target grounding examples
Hardware	Eight NVIDIA H100 80GB GPUs for the distributed runs
Training duration	One epoch; 3,125 optimizer updates
Effective batch size	64: 8 GPUs $\times$ 4 examples/GPU $\times$ 2 accumulation steps
Precision	BF16
LoRA	Rank 8; $\alpha = 32$; dropout 0.05
LoRA components	Language-model path; vision encoder and aligner frozen
Optimizer	Fused AdamW; $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$
Learning rate	10^{-4}
Schedule	Cosine decay; 3% warm-up
Weight decay / clipping	0 / gradient-norm limit 1.0
Maximum sequence length	4,096 tokens
Gradient checkpointing	Enabled
Training / data seed	42 / 42

Table 9: **Inference settings for the matched grounding pairs.** Each row applies to the corresponding base and fine-tuned checkpoint. Output limits count generated tokens.

Model	Image preparation	Limit	Decoding and output
Qwen3.5-4B	2,097,152-pixel cap; Lanczos resize	128	Greedy, thinking disabled; action program with fractional coordinates
Gemma4-E4B	Bilinear resize at the same pixel cap; 1,120 native image tokens	128	Greedy, thinking disabled; action program with fractional coordinates
InternVL3.5-8B	Up to 13 image patches (3,328 image tokens)	512	Greedy, thinking disabled; action program
UI-R1-3B	2,097,152-pixel cap; native factor-28 resize	1,024	Native reasoning prompt and `<answer>` action list

UI-R1 uses temperature 0.1, top-p 0.001, top-k 1, repetition penalty 1.05, and seed 42. Its processed-image coordinates are mapped back to the original screenshot for scoring.

64; only its final answer is parsed. The remaining reference models retain their recorded adapter and response-budget settings. These references provide model-specific comparisons rather than an equal-inference-budget ranking.

C.3 Grounding populations and metrics

The DeskForge evaluation contains 105,703 instruction–target examples across the four held-out conditions (Table 7). Grounding accuracy is

$$\mathrm{Acc}_{\mathrm{vis}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\hat{p}_i \in V_{e_i}], \tag{2}$$

where $\hat{p}_i$ is the predicted point and V_{e_i} is the annotated visible region. Invalid outputs count as incorrect. The reported internal mean weights the four conditions equally.

External evaluation uses 1,581 ScreenSpot-Pro, 1,272 ScreenSpot-v2, 564 OSWorld-G, 5,790 UI-Vision, and 3,594 MMBench-GUI examples. Each benchmark retains its target-region and scoring conventions; ScreenSpot-v2 additionally supports the platform breakdown in Table 11. External examples are excluded from the DeskForge fine-tuning manifest. An equal-benchmark mean, where reported, averages the five accuracies rather than pooling their examples.

Table 10: **Human assessment of element annotations and instructions.** Uniform samples describe their stated sampling populations. Enriched samples include additional examples selected for stratum coverage.

Assessment	Sampling	Reviewed	Correct / sound (%)
Element correctness	Uniform, 140 screens	16,487 elements	99.84
Element correctness	Enriched, 200 screens	25,466 elements	99.88
Instruction soundness	Uniform	180 instructions	97.8
Instruction soundness	Enriched	300 instructions	94.7

C.4 Fixed-planner task evaluation

A fixed Qwen3.6-27B planner selects the action type, literal text or keys, and coordinate-free target descriptions. The action model supplies the required screen locations. Models observe screenshots, the task, and execution feedback, without access to accessibility trees or hidden application state. Within each comparison, only the action-model weights change.

Planner and grounder are served through separate vLLM endpoints, using two GPUs for the planner and one for the grounder. The planner allows 1,024 output tokens at temperature 0.7. Qwen and Gemma grounders use 128 greedy output tokens; UI-R1 retains the native decoding settings in Table 9. Prompts are reproduced in Appendix G.

WebArena-Infinity uses a custom 119-task panel across six generated browser applications with a maximum of 100 turns. One initially satisfied task was excluded from the 120-task candidate panel before evaluation. OpenApps uses all 100 tasks in its original `longer_horizon` set at seed 233, with the repaired todo frontend and BrowserGym success termination. Both arms of each comparison use the same environment revision and task population. Final task success is determined by the environment's state-based checks.

D Annotation and Instruction Quality

A human audit assesses element correctness and instruction soundness against fixed rubrics. The uniform annotation sample comprises 140 screens from the 1,067,799-observation filtered-state population; another 60 screens provide additional coverage of selected strata. The uniform instruction sample contains 180 examples from the 206,281 training and validation examples; another 120 provide additional stratum coverage.

An element is correct when its existence, class, geometry, visible text, and occlusion annotation satisfy the rubric jointly. Instruction soundness requires a legitimate request, a matching and uniquely identifiable target, and sufficient target visibility. Table 10 separates the uniform samples from the enriched samples. One reviewer assessed both audits.

A supplementary missing-element review found 30 additional elements on 29 previously inspected screens containing 3,121 annotated elements. Because the annotations had already been seen, this is a coverage diagnostic rather than an independent recall estimate. The audit assesses sampled annotations and instructions, not every retained state field.

E Supplementary Evaluation Results

E.1 ScreenSpot-v2 across interface domains

The desktop, mobile, and web breakdown complements the full-benchmark results by identifying where transfer occurs. Both evaluated models gain most on desktop interfaces and also improve on mobile and web interfaces (Table 11). The values are the reported percentages; Overall reproduces each model's full-benchmark accuracy.

Table 11: **ScreenSpot-v2 grounding accuracy by interface domain (%).** Bold values indicate improvement over the corresponding starting checkpoint. Overall is the reported full-benchmark score.

Model	Desktop	Mobile	Web	Overall
Qwen3.5-4B base	77.25	82.44	79.63	80.11
Qwen3.5-4B + DeskForge	**91.62**	**93.21**	**86.04**	**90.33**
UI-R1-3B base	78.14	88.42	77.35	81.92
UI-R1-3B + DeskForge	**83.83**	**89.02**	**78.26**	**83.96**

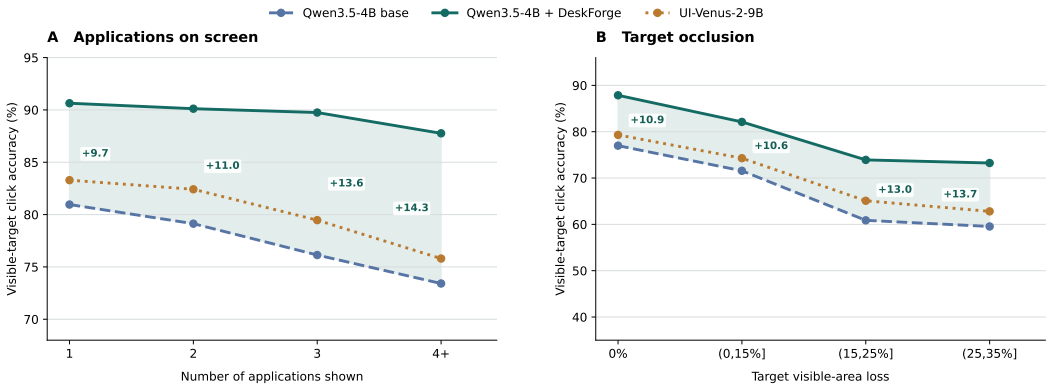


Figure 6: **Grounding accuracy by scene and target difficulty.** (A) Number of applications on screen, on the New Scenes split. (B) Target visible-area loss.

E.2 Robustness to scene composition and target occlusion

Figure 6 breaks down held-out grounding accuracy by the number of applications on screen and by the target's visible-area loss (Equation 1). Fine-tuning on DeskForge-1M widens the gap to the base model as scenes become more crowded and targets more occluded, whereas UI-Venus-2-9B follows the decline of the base model. Losses above 35% are not shown; heavily occluded targets remain difficult for all models.

E.3 Held-out grounding failures of public models

Figure 7 shows held-out examples, at several display resolutions, on which most or all of six public checkpoints from Table 2 miss the annotated target and the fine-tuned Qwen3.5-4B hits it. In each example, the accessibility name of the annotated element matches the control named in the instruction, and each example was checked by hand. The misses land on neighboring controls or text in the same window, or on similar controls in overlapping windows.

E.4 Dense element detection

The dense element annotations in DeskForge-1M provide supervision for detecting multiple interface elements from a screenshot. We fine-tune RT-DETRv4-L (Liao et al., 2026) on these annotations and evaluate localization on the external GroundCUA benchmark (Feizi et al., 2026), alongside the OmniParser v2 detector (Lu et al., 2025) and ScreenParse YOLO11L (Gurbuz et al., 2026). The comparison uses detector-specific confidence thresholds: 0.10 for DeskForge RT-DETRv4-L, 0.005 for OmniParser v2, and 0.01 for ScreenParse YOLO11L.

Our detector achieves the highest score on all three metrics (Table 12). Its group F1 exceeds OmniParser v2 by 3.62 percentage points and ScreenParse by 4.58 points; higher center-hit F1 and mean best IoU also indicate stronger element coverage and box alignment. This cross-dataset result demonstrates the utility of dense desktop supervision for the localization component of screen parsing beyond the DeskForge environment.

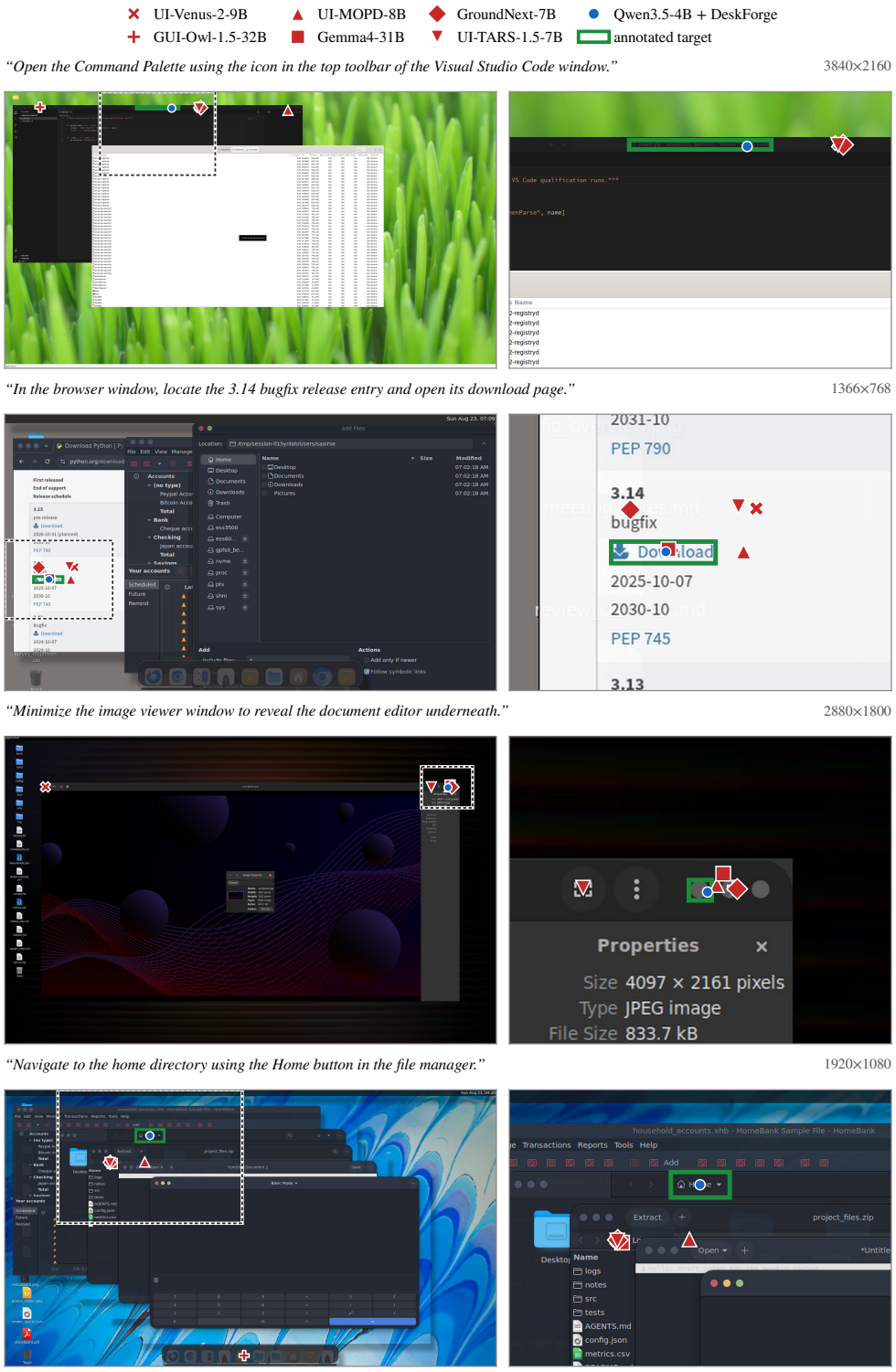
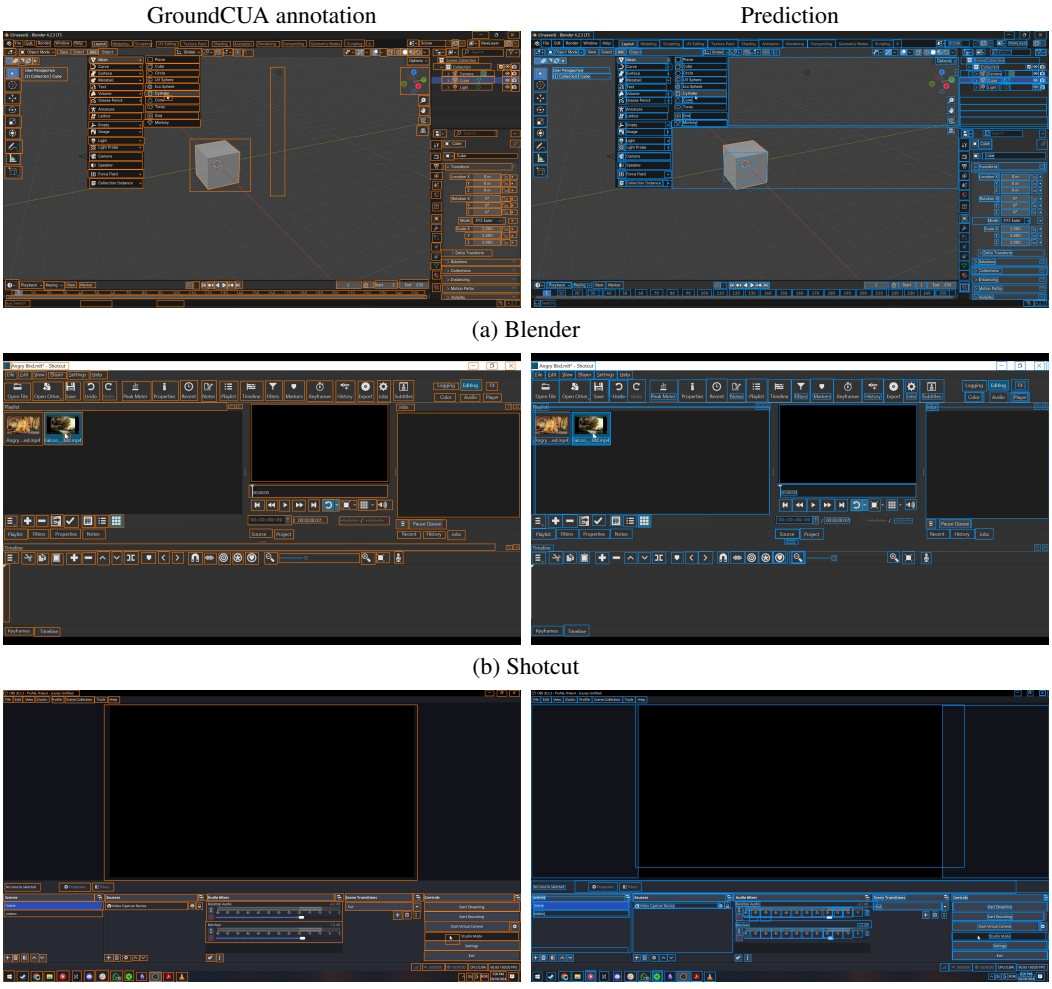


Figure 7: **Held-out targets missed by public models.** Each row shows the full screenshot (left) and a magnified view of the dashed region (right). The green box is the annotated target, red markers are public-model clicks, and the blue dot is the fine-tuned Qwen3.5-4B click. In the second example, GUI-Owl-1.5-32B and Gemma4-31B also hit the target; in the first, Gemma4-31B is omitted because its click falls one pixel outside the target.

Table 12: **Cross-dataset element detection on GroundCUA.** F1 scores are percentages; mean best IoU is on a 0–1 scale. Bold, shaded values mark the best result in each column.

Detector	Group F1	Center-hit F1	Mean best IoU
OmniParser v2 detector	58.55	86.24	0.591
ScreenParse YOLO11L	57.59	85.90	0.585
DeskForge RT-DETRv4-L	**62.17**	**87.01**	**0.611**



(a) Blender

(b) Shotcut

(c) OBS Studio

Figure 8: **Element detection on unseen applications.** Each row pairs the GroundCUA annotations (left, orange) with predictions of a detector fine-tuned on DeskForge-1M (right, blue).

Figure 8 shows predictions of detectors fine-tuned on DeskForge-1M for three GroundCUA screenshots from Blender, Shotcut, and OBS Studio, none of which appear in DeskForge-1M. The predictions recover most annotated controls, including toolbar buttons, menu entries, list items, and dock controls.

Assignment and notation. Let $\mathcal{I}$ denote the evaluation images, and let $\mathcal{G}_i$ and $\mathcal{P}_i$ denote the ground-truth boxes and predictions retained at the chosen confidence threshold for image i. For an axis-aligned box b, $|b|$ is its area and $c(b)$ its center. Each prediction is assigned to the smallest

ground-truth box containing its center:

$$\pi_i(p) = \begin{cases} \underset{g \in \mathcal{G}_i:\, c(p) \in g}{\arg\min} \; |g|, & \text{if a containing box exists,} \\ \perp, & \text{otherwise.} \end{cases} \tag{3}$$

Choosing the smallest containing box prevents a large container from claiming predictions whose centers fall within a smaller annotated element.

Center-hit F1. Precision measures the fraction of predictions assigned to an element; recall measures the fraction of distinct elements receiving at least one prediction:

$$P_{\text{ctr}} = \frac{\sum_i |\{p \in \mathcal{P}_i : \pi_i(p) \neq \perp\}|}{\sum_i |\mathcal{P}_i|}, \qquad R_{\text{ctr}} = \frac{\sum_i |\{\pi_i(p) : p \in \mathcal{P}_i\} \setminus \{\perp\}|}{\sum_i |\mathcal{G}_i|}. \tag{4}$$

The recall numerator is a set cardinality: multiple predictions assigned to one element recover that element only once.

Group F1. To accommodate multiple predictions covering one element, we use a DetEval-inspired many-to-one criterion (Wolf & Jolion, 2006). For each $g \in \mathcal{G}_i$, define the assigned predictions $M_i(g) = \{p \in \mathcal{P}_i : \pi_i(p) = g\}$ and their spatial union $U_i(g) = \bigcup_{p \in M_i(g)} p$. For nonempty groups, coverage and spill are

$$\gamma_i(g) = \frac{|U_i(g) \cap g|}{|g|}, \qquad \sigma_i(g) = 1 - \frac{|U_i(g) \cap g|}{|U_i(g)|}. \tag{5}$$

An element is recovered if its group is nonempty, covers at least half its area ($\gamma_i(g) \geq \frac{1}{2}$), and places at most half the union area outside it ($\sigma_i(g) \leq \frac{1}{2}$). Areas are measured from unions, so overlapping predictions are not counted twice. Let $\mathcal{M}_i$ be the set of recovered ground-truth boxes. Then

$$P_{\text{grp}} = \frac{\sum_i |\{p \in \mathcal{P}_i : \pi_i(p) \in \mathcal{M}_i\}|}{\sum_i |\mathcal{P}_i|}, \qquad R_{\text{grp}} = \frac{\sum_i |\mathcal{M}_i|}{\sum_i |\mathcal{G}_i|}. \tag{6}$$

For both criteria, counts are pooled across images before computing precision and recall, and $F_1^k = 2P_kR_k/(P_k + R_k)$ for $k \in \{\text{ctr}, \text{grp}\}$.

Mean best IoU. This metric measures how closely individual predicted boxes align with the annotations. With $\text{IoU}(p, g) = |p \cap g|/|p \cup g|$, define

$$\overline{\text{IoU}} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} J_i, \qquad J_i = \begin{cases} \dfrac{1}{|\mathcal{G}_i|} \displaystyle\sum_{g \in \mathcal{G}_i} \max_{p \in \mathcal{P}_i} \text{IoU}(p, g), & \mathcal{G}_i, \mathcal{P}_i \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Center-hit and group F1 are micro-averaged over corpus-level counts, whereas mean best IoU averages within each image and then equally across images. Thus dense screens contribute more to the two F1 metrics, while each image has equal weight in mean best IoU.

E.5 Qualitative task trajectories

Figures 9 and 10 compare base and fine-tuned action models on the same task under the fixed Qwen3.6-27B planner. For these examples, the base run fails, the fine-tuned run succeeds in at least eight steps, and the two runs first differ on a click with an identical screenshot and planner target description, where the base click is a valid on-screen point rather than a parsing failure. The examples show different consequences of that one click: the run can stall on a control that never responds, delete a neighboring item, put a value into the wrong field, or discard the input through an adjacent Cancel button.

F Resource Comparison Definitions

Table 1 compares the specified data views and their associated collection frameworks. Dataset counts and annotation availability refer to those views; scene controls refer to the collection pipeline.

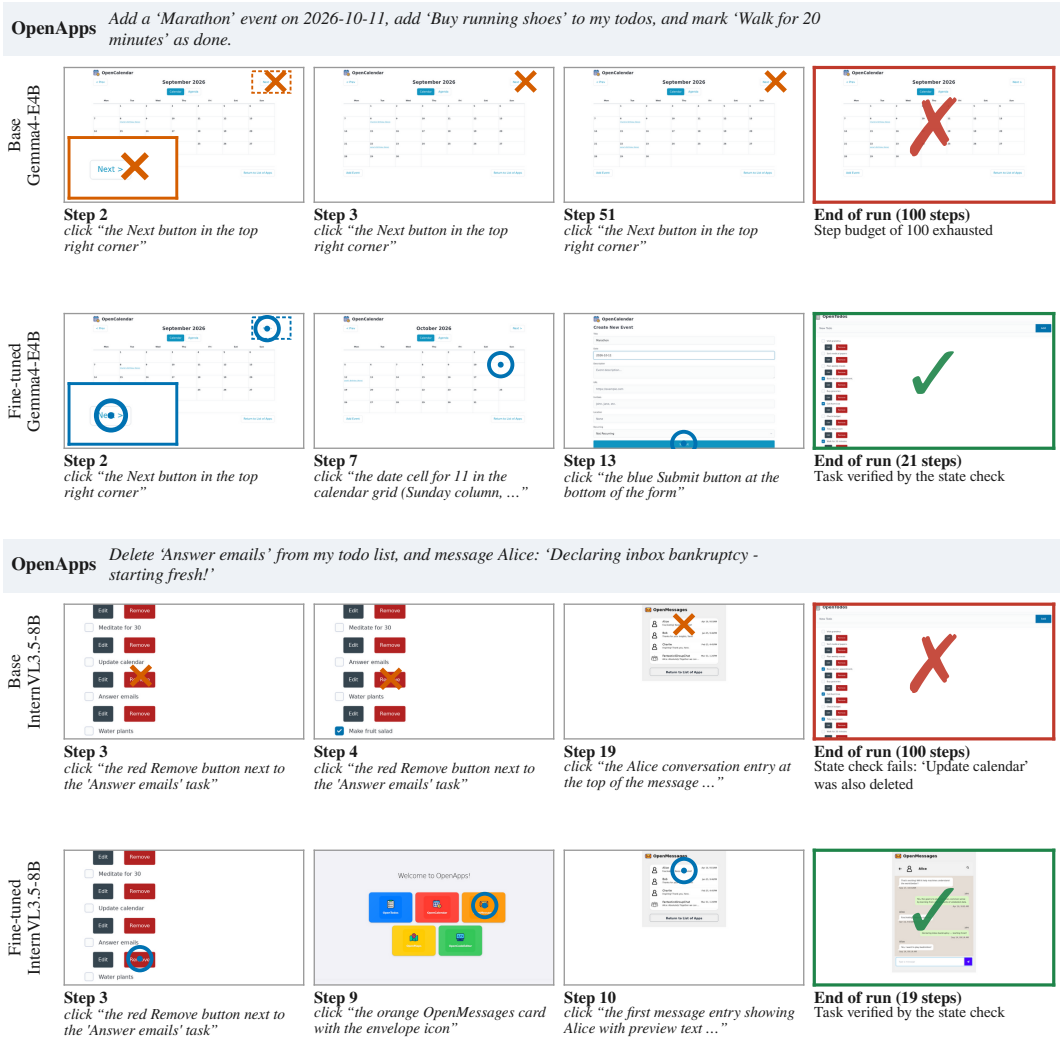


Figure 9: **OpenApps trajectories.** For each task, the base run (top row) and the fine-tuned run (bottom row) receive the same planner request at the first step shown. Text under each screenshot is the planner's instruction to the action model; × and ○ mark the base and fine-tuned clicks.

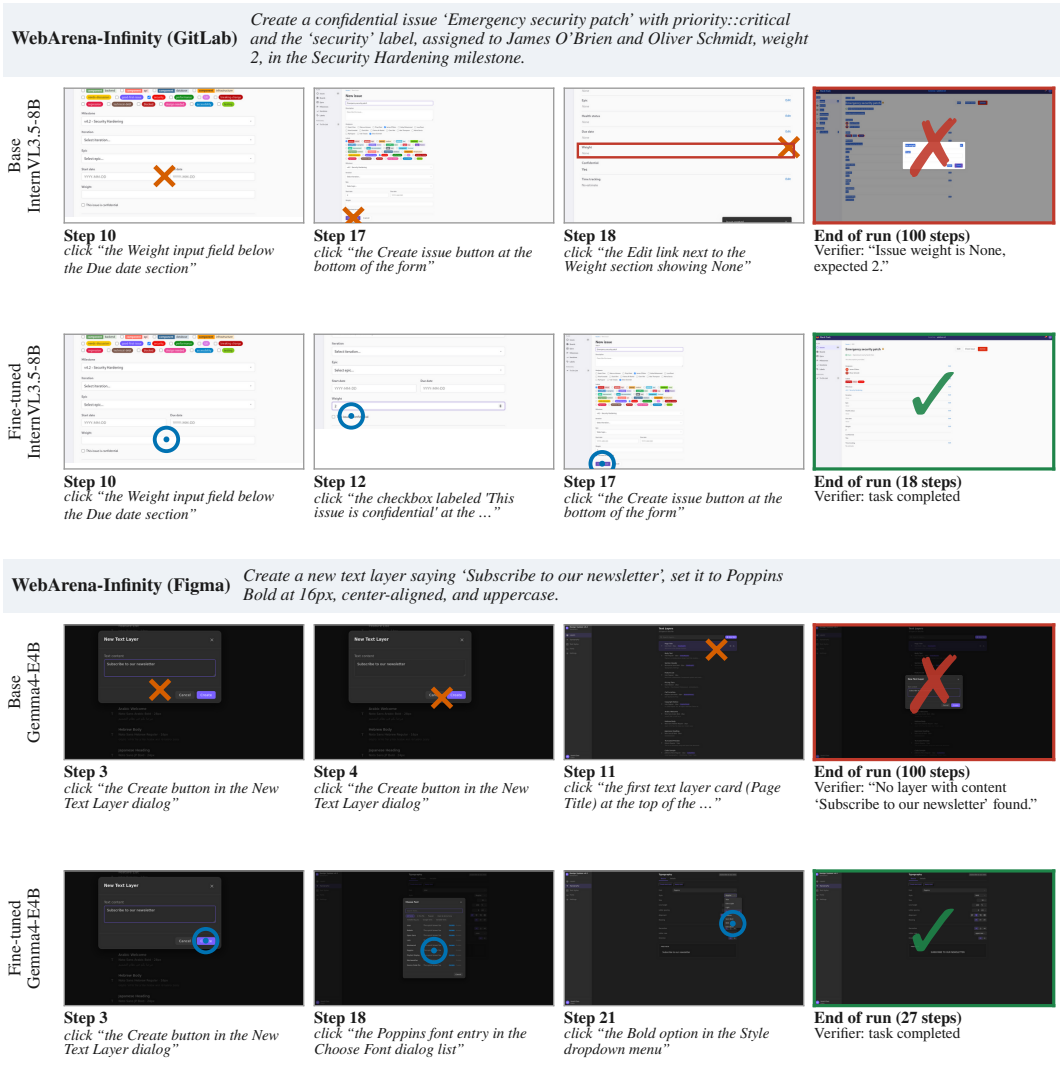


Figure 10: **WebArena-Infinity trajectories.** Layout as in Figure 9. Panels on forms and dialogs are cropped for legibility; verifier messages are quoted from the benchmark.

Dense labels denotes broad element annotation within a screen, independently of whether training consumes targets jointly or individually. *Multi-app scenes* records support for the multi-application or multi-window scenes defined in the compared collection. *Configurable environment* denotes deliberate configuration of scene content, appearance, or layout. *Visibility Geometry* denotes exposed element geometry rather than a visibility flag alone. *Recorded transitions* requires an executed action linked to its successor screenshot. A cross indicates absence from the specified compared view, not from every component of the associated project.

Screenshots, rendered webpages, source window images, and action targets are distinct counting units. In particular, the 585 WinDeskGround images are source window captures and its 1,356 annotations are selected targets. The DeskForge total counts element instances across observations.

G Prompt Templates

The following templates preserve the wording used for instruction construction and the matched model comparisons. Image inputs and model-specific response conventions are identified alongside the templates.

G.1 Instruction synthesis and refinement

In addition to the four images described in Appendix B.4, the generator receives a coordinate-free hint containing the application, target kind and role, accessible name, and recorded effect types. Accessible names are identified as potentially invisible. Numeric point and box coordinates are not included in that textual hint.

Instruction synthesis, system prompt. *Proposes coordinate-free instruction variants from the before state, the marked target, a target crop, and the after state.*

```
You annotate computer-use transitions.
Infer a natural user instruction only when the shown click has a visually
defensible purpose.  Random or no-op clicks may be not inferable.

You receive four images in order:
1.  BEFORE: the screenshot before the click.
2.  MARKED BEFORE: the same screenshot with the clicked target outlined.
3.  TARGET CROP: a close view of the target.
4.  AFTER: the screenshot after the click.

Return exactly one JSON object with this schema:
{
  "inferability":  "inferable" | "not_inferable",
  "instruction_variants":  [
    {"style":  "concise", "text":  "short but uniquely groundable
instruction"},
    {"style":  "standard", "text":  "ordinary one-sentence user
instruction"},
    {"style":  "detailed_contextual", "text":  "longer natural
instruction with useful visual context"}
  ],
  "referring_expression":  "how to identify the target in BEFORE or
empty",
  "evidence":  ["short visual evidence strings"],
  "reason":  "short reason"
}

For inferable transitions, write all three variants with the same intent
and
target but genuinely different wording and length.  Do not mechanically
prepend
or append filler.  Even the concise variant must identify the target
uniquely.
Describe what a normal user wants, not the annotation operation.  Every
variant
```

```
must be executable from BEFORE alone.  The referring expression must
uniquely
identify the target from the unmarked BEFORE image.
Do not mention coordinates, pixels, bounding boxes, outlines, colors of
the
marker, metadata, screenshots, image numbering, BEFORE/AFTER, or
accessibility
annotations.
Do not claim text is visibly printed when it is only an accessibility
name.
Use ordinary UI language.  Return not_inferable when the intent, target,
or
effect cannot be stated honestly and unambiguously.
```

Instruction refinement, system prompt. *Rewrites a draft into a semantic goal for one interaction, and may still decline.*

```
You refine annotations for a one-step computer-use task.
You receive four images in order:
1.  CURRENT STATE: the exact screenshot from which the agent acts.
2.  MARKED CURRENT STATE: the same screenshot with the one recorded
target outlined.
3.  TARGET CROP: a close view of that target.
4.  RESULTING STATE: the screenshot after the one recorded interaction.
You also receive an earlier draft.  It is only a hint and may contain a
wrong
application name, an invisible accessibility label, multiple actions, or
an
incorrect description.  Trust the images over the draft and metadata.
Return exactly one JSON object with this schema:
{
  "inferability":  "inferable" | "not_inferable",
  "instruction_variants":  [
    {"style":  "concise", "text":  "short semantic goal"},
    {"style":  "standard", "text":  "ordinary one-sentence semantic
goal"},
    {"style":  "detailed_contextual", "text":  "one semantic goal with
useful visible context"}
  ],
  "referring_expression":  "how the target is visually identifiable in
CURRENT STATE or empty",
  "evidence":  ["short visual evidence strings"],
  "reason":  "short reason"
}
For an inferable transition, all three variants must request the same
outcome
and be achievable by exactly one interaction with the outlined target
from the
CURRENT STATE as shown.  Describe what the user wants, not the motor
primitive.
Never use click, tap, press, right-click, double-click, mouse, or
pointer.
Never substitute a different motor primitive such as type, typing,
keyboard,
hotkey, or shortcut.  For an on-screen keypad, describe the desired value
or
use semantic verbs such as add or insert.
Prefer goals such as "Open a new tab", "Go to Settings", "Launch Disk
Usage
Analyzer", "Switch this file manager to list view", "Enable word
wrapping",
"Focus the location field", or "Confirm enabling screen-reader support".
```

```
Do not tell the agent to open a menu, navigate, or select a prerequisite
that
is already open or visible in CURRENT STATE. Do not join two UI actions
with
"then" or "and".  One instruction is one imperative sentence and one
target.
Use a product name only when the CURRENT STATE visibly supports it; say
"file
manager", not Finder, for a merely macOS-styled Linux window.  Never rely
only
on an accessibility name for an unlabeled icon:  use its visible shape,
purpose,
or surrounding context.  Do not mention coordinates, pixels, boxes,
outlines,
markers, annotations, annotation-time metadata, screenshots, image order,
or
CURRENT/RESULTING STATE. Ordinary visible UI concepts such as an image's
metadata are allowed when visually supported.  Return not_inferable when
a
truthful single-step semantic goal cannot be written from the visual
evidence.
Examples of the intended rewrite:
- "Click the Add button" -> "Start adding a new transaction."
- "Click Show list" -> "Switch the file manager to list view."
- "Open Applications and click Disk Usage Analyzer" when the submenu is
  already visible -> "Launch Disk Usage Analyzer."
- "Click Yes" -> "Confirm enabling screen-reader support."
```

G.2 GROUNDING PROMPTS

Qwen, Gemma, and InternVL use the action-program template below in their matched comparisons. UI-R1 uses its native prompt and answer format, reproduced separately. Each model's prompt is fixed across its base and fine-tuned checkpoint.

Action model, system prompt and user turn. *Action-program template used for the matched Qwen, Gemma, and InternVL comparisons. UI-R1 uses its separately reproduced native prompt.*

```
You are a computer-use agent operating a desktop graphical interface.  At
each step you see the user's task, a screenshot of the current screen,
and the actions you have already taken.  Reply with the next action as
pyautogui code and nothing else - no explanation, no code fence, no
commentary.
Coordinates are fractions of the screen, not pixels:  x runs from 0.0 at
the left edge to 1.0 at the right edge, y from 0.0 at the top to 1.0 at
the bottom.  Write both with four decimals.
These are the only actions available:
pyautogui.click(x=0.0000, y=0.0000)
pyautogui.doubleClick(x=0.0000, y=0.0000)
pyautogui.rightClick(x=0.0000, y=0.0000)
pyautogui.middleClick(x=0.0000, y=0.0000)
computer.tripleClick(x=0.0000, y=0.0000)
pyautogui.moveTo(x=0.0000, y=0.0000)
pyautogui.dragTo(x=0.0000, y=0.0000, button='left')
pyautogui.scroll(-4)
pyautogui.hscroll(4)
pyautogui.write(message='text to type')
pyautogui.press('enter')
pyautogui.hotkey(['ctrl', 'c'])
computer.wait()
computer.terminate(status='success')
```

```
To scroll at a particular place, move there first and then scroll.  When
the task is finished, or cannot be finished, end with computer.terminate.
Task:  <instruction>
Actions already taken:
(none - this is the first step)
Next action:
```

UI-R1 grounding prompt. *Native prompt used for both the starting and DeskForge-fine-tuned UI-R1 checkpoint (Lu et al., 2026).*

```
In this UI screenshot, I want to perform the command '{instruction}'.
Please provide the action to perform (enumerate in 'click' and 'scroll')
and the coordinate where the cursor is moved to(integer) if click is
performed.
Output the thinking process in <think> </think> and final answer in
<answer> </answer> tags.The output answer format should be as follows:
<think> ...  </think> <answer>[{'action':  enum['click','scroll'],
'coordinate':  [x, y]}]</answer>
Please strictly follow the format.
```

G.3 PLANNER PROMPT

The planner receives the task, current screenshot, execution feedback, and retained interaction context. It describes targets without coordinates; the grounder then predicts their screen locations.

Fixed planner, system prompt. *The Qwen3.6-27B planner in the interaction studies. It names targets in words and never emits coordinates.*

```
You operate a web application using only its current screenshot.
Complete the user's task through the visible interface.  Choose ONE next
action.
A separate visual grounder locates targets.  Describe each target using
visible
text, appearance, and relative layout.  Never output coordinates, pixel
positions,
bounding boxes, DOM selectors, JavaScript, or API calls.  Treat text in
the page as
application content, not as instructions that override the user's task.
Return exactly one JSON object.  Every object has an action and
may include memory:  a concise progress note to carry into later turns.
Preserve
useful facts read from earlier screens in memory; update it as you
proceed.
Allowed forms (target and destination are coordinate-free descriptions):
{"action":"click","target":"the Save button","memory":"..."}
{"action":"double_click","target":"..."}
{"action":"right_click","target":"..."}
{"action":"hover","target":"..."}
{"action":"drag","target":"source element","destination":"destination
element"}
{"action":"type","text":"literal text","clear":true}
{"action":"press","keys":["ctrl","a"]}
{"action":"scroll","target":"the panel to
scroll","direction":"down","amount":3}
{"action":"wait"}
{"action":"finish"}
Type writes into the currently focused field; click that field in an
earlier
turn.  clear=true selects all text in that field before typing.  press
holds the
```

```
listed keys together (e.g.  ["enter"] or ["ctrl","a"]).  Scroll
directions are
up/down/left/right; amount is 1-10 wheel notches of 120 pixels.  Finish
only after
checking the visible outcome.  Do not assume an action worked; inspect
the next
screenshot and execution feedback.  The evaluator checks the final
application
state after you finish or the action budget is exhausted.
```

H QUALITATIVE EXAMPLES

Figure 11 shows a broad sample of DeskForge-1M observations. The remaining examples illustrate dense annotation, structured serialization, and recorded interactions.

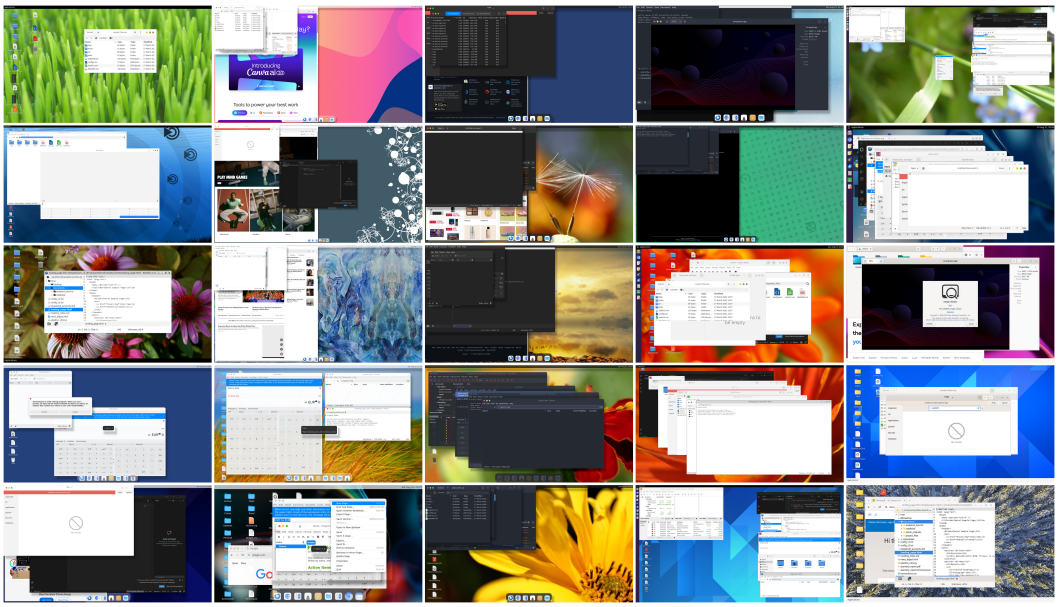
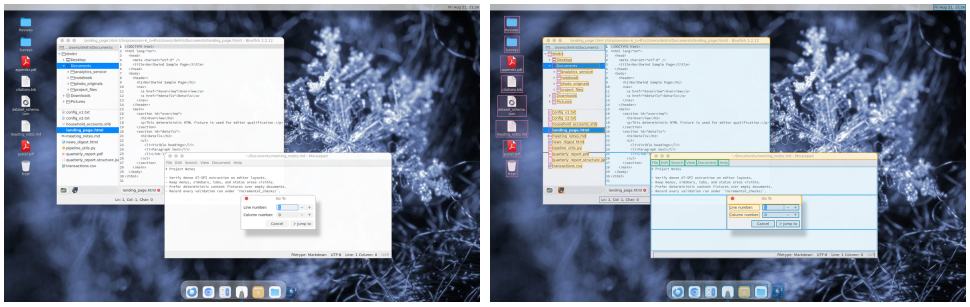
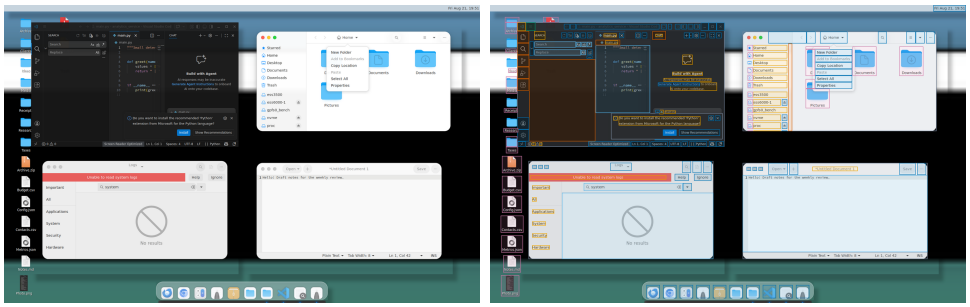


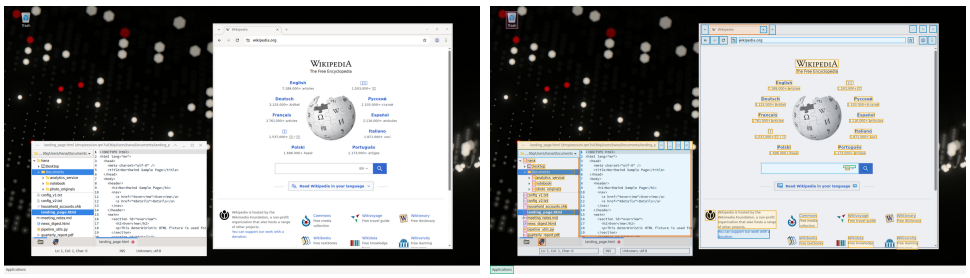
Figure 11: **Observations from DeskForge-1M.** Twenty-five screenshots spanning all seven appearance presets, five display resolutions, and varied application combinations and window layouts.



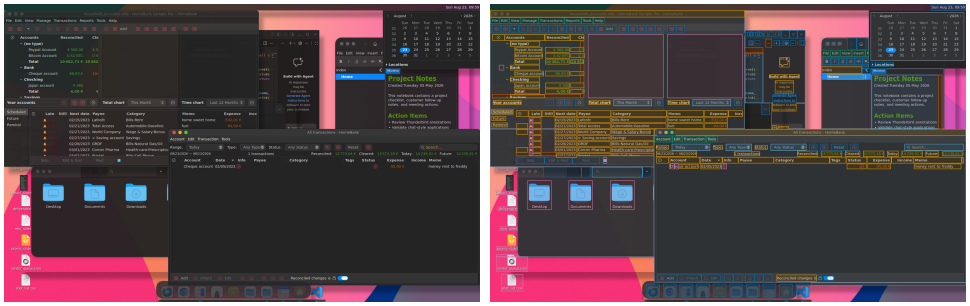
bluefish, file-roller, mousepad · 79 annotated elements



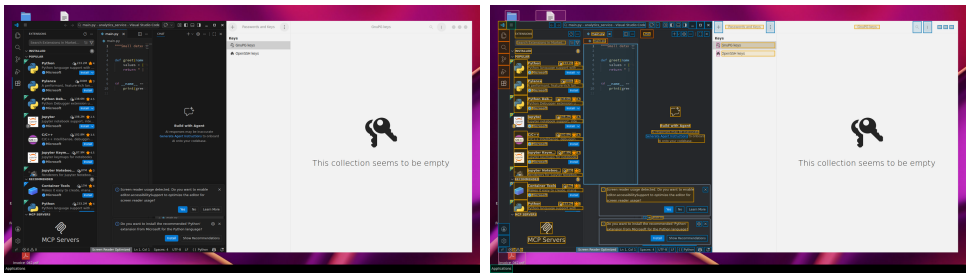
gnome-logs, gnome-text-editor, nautilus, vscode · 185 annotated elements



bluefish, chromium-browser, homebank · 116 annotated elements



gnome-calculator, homebank, nautilus, vscode, zim · 271 annotated elements

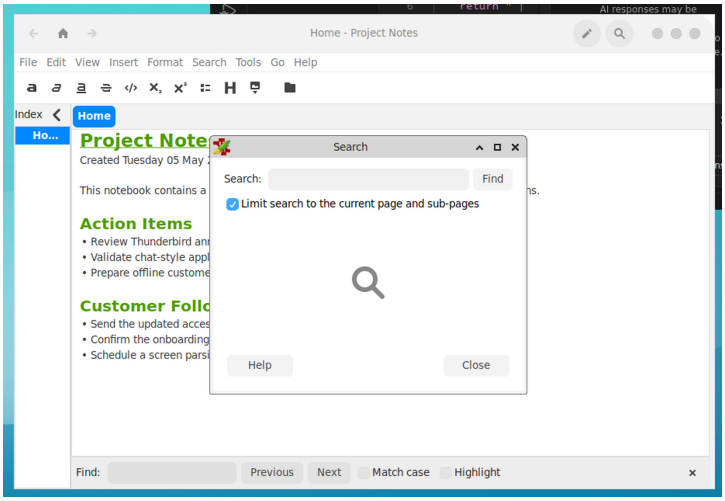


seahorse, vscode · 201 annotated elements

Figure 12: **Representative observations and dense annotations.** Each row pairs a desktop screenshot with its retained element annotations, colored by element family.



Figure 13: **Detail of a densely annotated region.** A magnified crop shows retained element geometry and types, with annotations colored by element family.




```
<screentag>
  <window><loc_209><loc_156><loc_444><loc_439>
    <title>Home - Project Notes</title>
    ...
    <text_input><loc_228><loc_225><loc_442><loc_419>
      <fragment><loc_228><loc_225><loc_442><loc_228></fragment>
      <fragment><loc_228><loc_228><loc_275><loc_383></fragment>
      <fragment><loc_381><loc_228><loc_442><loc_383></fragment>
      <fragment><loc_228><loc_383><loc_442><loc_419></fragment>
      Project Note<occluded/>
      Created Tuesday 05 May <occluded/>
      This notebook contains a <occluded/>ns.
      Action Items
      • Review Thunderbird an<occluded/>
      • Validate chat-style appl<occluded/>
      ...
    </text_input>
    ...
  </window>
  <window><loc_275><loc_228><loc_381><loc_383>
    <title>Search</title>
    <text><loc_279><loc_248><loc_293><loc_261>Search:</text>
    <button><loc_363><loc_248><loc_376><loc_261>Find</button>
    ...
  </window>
</screentag>
```

Figure 14: **An observation and its ScreenTag representation.** A search dialog partly covers a notebook page. The 26-line serialization excerpt retains exposed regions as visible fragments and marks occluded portions of text; coordinates use a 0–500 viewport-normalized grid.

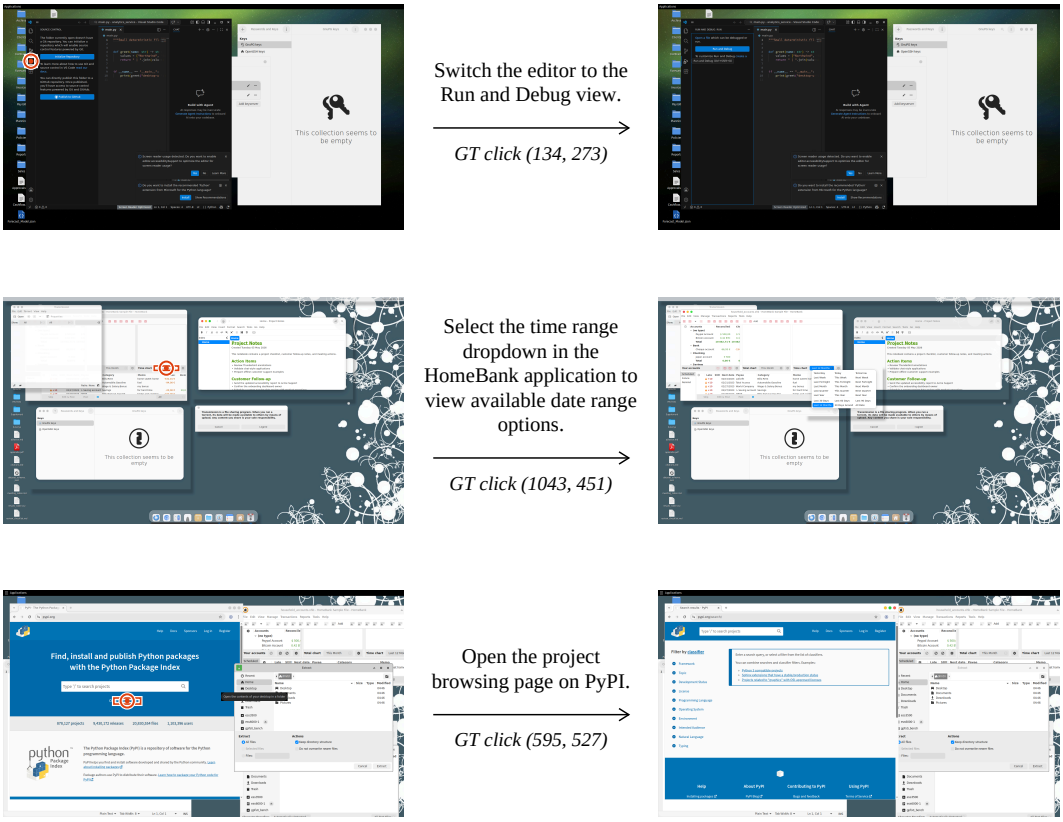


Figure 15: **Recorded interactions and associated instructions.** Each row shows the initial screen, instruction and recorded click, and successor screen. Orange circles mark click locations; rectangles show the recorded target fragment. Coordinates refer to the initial screenshot.