AUTOGUIWORLD: IMAGE GENERATORS AS VISUAL WORLD MODELS FOR GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents require high-quality interaction trajectories to learn how software environments respond to actions, maintain state, and support multi-step workflows. However, the diversity of available trajectories is constrained by the applications, interface states, and workflows accessible in the underlying environments. Expanding this coverage requires deploying increasingly diverse and complex software, with specialized applications imposing additional installation, configuration, and runtime costs. We introduce **AutoGUIWorld**, a data generation framework that combines the visual priors of image generators with the task knowledge of a planner to synthesize GUI interaction trajectories without deploying or running the corresponding software environments. AutoGUIWorld samples initial scenes from a structured GUI world space and generates tasks conditioned on those scenes. A planner then specifies atomic actions and their intended visual consequences, while an image generator iteratively edits the current screenshot to produce subsequent observations. Action grounding and transition-level quality control provide spatial annotations and filter defective samples, yielding trajectories for agent training. Fine-tuning Qwen3.5-35B-A3B on AutoGUIWorld trajectories improves the mean task score on OSWorld from 33.0% to 40.8% and the task success rate on ScienceBoard from 14.0% to 32.2%. These results demonstrate the utility of generated interaction trajectories for training GUI agents to operate real desktop and scientific software. Code is available at https://anonymous.4open.science/r/agw-0E72.



Figure 1: Overview of AutoGUIWorld as an environment-free GUI data engine. It samples a structured GUI world, realizes a seed screenshot, plans action-conditioned transitions, and uses Image2 as a visual world model to synthesize coherent screenshot-action-screenshot trajectories.

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.

1 INTRODUCTION

GUI agents automate software tasks through screenshots, mouse actions, and keyboard inputs. Large-scale interaction data improves GUI perception, grounding, and task execution (Qin et al., 2025; Wang et al., 2025a; Liu et al., 2025). Effective interaction also requires environment knowledge: how actions change observable states, what persists, and how earlier operations constrain later ones (Lu et al., 2024; Chae et al., 2024). An agent must relate the current screenshot to previous actions and the remaining task, distinguishing content that should persist from changes needed to make progress. High-quality trajectories provide supervision for learning these dependencies by connecting instructions and actions to observable outcomes. Their successive observations show how intermediate decisions shape later states across multi-step workflows.

Trajectory diversity depends on the breadth and complexity of the environments available for collection: their reachable interface states, action semantics, and workflow structures determine which interactions can be observed. Scientific, CAD, and electronic-design software expose specialized states and operations (Sun et al., 2025; Dong et al., 2026; Li et al., 2025a), while creative and professional workflows require content preservation and dependent editing steps (Wang et al., 2026c; Pang et al., 2026; Ai et al., 2025; Zhu et al., 2026). Collecting more trajectories within an existing environment can increase task coverage, but cannot supply interactions specific to software absent from the collection. Expanding the environment pool requires deploying and configuring software, preparing task states, and maintaining reproducible execution. For specialized software, this entails accommodating different dependencies, runtime requirements, and initialization procedures, alongside possible licensing or account constraints (Sun et al., 2025; Dong et al., 2026; Aggarwal et al., 2026). These costs motivate generating training experience without deploying or running each corresponding software environment.

Existing acquisition methods use human demonstrations (Deng et al., 2023; Rawles et al., 2023), automated exploration and filtering (Sun et al., 2024; Xiao et al., 2025), or extraction from tutorials and recorded videos (Zhang et al., 2025; Lu et al., 2025a). Their coverage remains tied to accessible executable environments or the interfaces and workflows present in existing records. Generation offers a complementary way to expand the available interaction experience.

Generative approaches construct executable environments with tasks and verifiers (Zhang et al., 2026; Wu et al., 2026; Wang et al., 2026a), or simulate observations through structured states (Wang et al., 2025b; Xiao et al., 2026; Ding et al., 2026), visual prediction (Luo et al., 2025; Rivard et al., 2025; Xiang et al., 2025; Guan et al., 2026), and renderable code (Zheng et al., 2026; Koh et al., 2026). Pretrained image generators offer another source of visual priors (Gabeur et al., 2026), but turning these priors into training trajectories requires observations that follow intended actions and preserve context across steps (Fu et al., 2026). Visual plausibility alone does not establish these properties: opening a dialog should preserve the document behind it, and editing one field should leave unrelated content intact. Generated screenshots also need corresponding action coordinates to provide spatial supervision. These requirements motivate a process that connects task-level intent to individual visual changes, grounds actions in screenshots, and checks the resulting transitions. We investigate whether task planning and transition-level quality control can make pretrained image generators a practical source of training data for GUI agents operating in real environments.

We introduce **AutoGUIWorld**, which combines structured environment sampling, scene-conditioned tasks, and visual trajectory generation to construct training experiences without deploying or running each sampled environment. A Meta Planner specifies the overall interaction plan; Voyager uses that plan and the current screenshot to describe the next scene, which Image2 generates by editing the screenshot. Grounding supplies action coordinates, and quality control repairs annotations and removes defective samples. Our contributions are:

- **GUI trajectory generation.** We combine task planning and image generation to synthesize trajectories without deploying or running the corresponding software environments.
- **Curated training data.** We construct 79,266 grounded and filtered step-level samples across Ubuntu, Windows, macOS, and Chrome, and analyze their coverage and transition defects.
- **Transfer to real environments.** Fine-tuning Qwen3.5-35B-A3B improves all four interactive benchmarks, including OSWorld from **33.0% to 40.8%** mean task score and ScienceBoard from **14.0% to 32.2%** task success.

2 Related Work

GUI trajectory collection. Human demonstrations provide trajectories from existing interfaces, including Mind2Web, WebLINX, Android in the Wild, and GUIOdyssey (Deng et al., 2023; Lù et al., 2024; Rawles et al., 2023; Lu et al., 2024). Tutorial and video pipelines recover action supervision from recorded interaction (Zhang et al., 2025; Jang et al., 2025; Xiong et al., 2026). Automated approaches expand collection through exploration, task proposal, replay, and filtering: OS-Genesis infers tasks from explored states, and UI-Genie combines generation with verification (Sun et al., 2024; Xiao et al., 2025). OpenCUA scales human demonstration collection and processing, while ScaleCUA combines automated agents with human experts (Wang et al., 2025a; Liu et al., 2025). A complementary line expands the executable substrate itself. AndroidWorld and OSWorld provide resettable environments, while InfiniteWeb, AutoWebWorld, and CUA-Gym synthesize environments, tasks, or verifiers (Rawles et al., 2024; Xie et al., 2024; Zhang et al., 2026; Wu et al., 2026; Wang et al., 2026a). These approaches obtain transitions by executing applications or constructed environments. AutoGUIWorld generates new screenshot-level transitions directly, making visual appearance and scene composition explicit sampling dimensions. This trades executable state for a cheaper visual target; action correctness and final task success still require validation in real systems.

GUI world models. Textual and structured world models support imagined planning and learning: WebDreamer and SimuRA predict action outcomes for lookahead, and DynaWeb uses imagined rollouts for policy learning (Gu et al., 2024; Deng et al., 2025; Ding et al., 2026). UI-Simulator synthesizes structured UI states and uses guided rollouts to construct training trajectories (Wang et al., 2025b). Visual models generate future screens through different intermediate representations. ViMo combines diffusion with symbolic-text information, NeuralOS couples recurrent state tracking with a renderer, and UISim first predicts abstract layouts (Luo et al., 2025; Rivard et al., 2025; Xiang et al., 2025). The Computer-Using World Model predicts textual state changes before rendering, whereas Code2World and gWorld generate code that is executed to obtain the next visual state (Guan et al., 2026; Zheng et al., 2026; Koh et al., 2026). AutoGUIWorld uses a general-purpose image generator as the visual transition component of a planner-guided data engine. The planner supplies task context and intended changes; the generator chain-edits clean observations to produce training trajectories. Its focus is controllable data generation across platforms and appearances, rather than using a learned simulator for test-time search. Appendix G gives the extended comparison.

3 AutoGUIWorld

AutoGUIWorld synthesizes GUI interaction trajectories for training GUI agents without deploying or running the corresponding software environments. Each trajectory consists of a task instruction τ, screenshots, and atomic actions:

$$\xi = (\tau, \tilde{s}_0, a_0, \tilde{s}_1, \ldots, a_{T-1}, \tilde{s}_T), \tag{1}$$

where $\tilde{s}_t$ is the screenshot before action a_t, and T is the number of actions in the trajectory.

GUI World and Task Generation. We first sample a structured world specification w comprising three factors: an *OS substrate*, which specifies the platform, screen geometry, interaction conventions, and application ecosystem; *visual appearance*, which specifies the theme, typography, density, and materials; and an *initial GUI state*, which specifies window layouts, foreground relations, visible controls, and associated content.

The sampled specification is compiled into a visual description of the initial scene. Image2, the image generation and editing model used in our pipeline, renders this description into the initial screenshot $\tilde{s}_0$. The screenshot and its seed metadata provide the initial environment context for task generation and trajectory planning.

An LLM-based task generator then produces a task instruction τ from the platform context and the visible elements described in the seed. A task registry discourages duplicate tasks and encourages coverage across applications and interaction types.

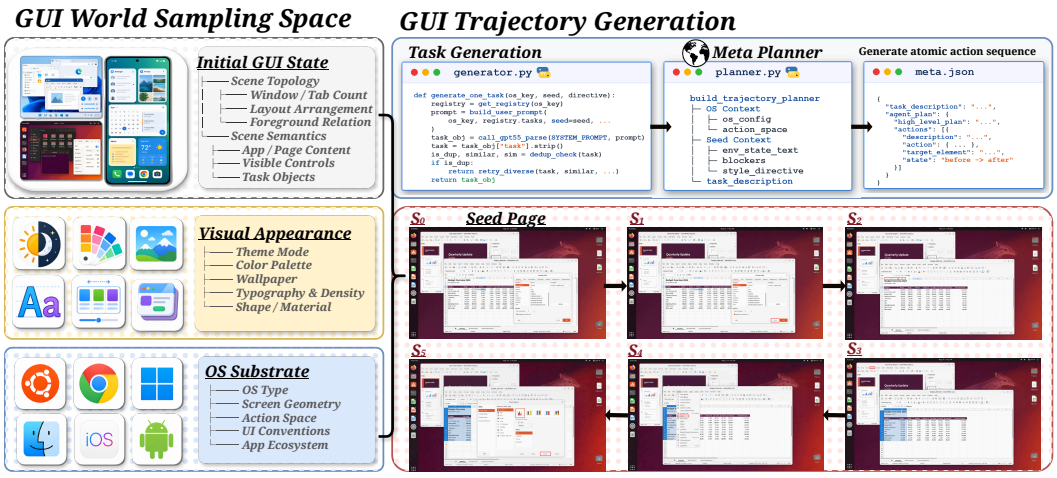

Figure 2: Overview of AutoGUIWorld. A sampled GUI world is rendered into an initial screenshot and used to generate a task. The meta planner organizes the task into an overall interaction plan. Screenshots are then generated step by step and annotated with action targets.

Planner-Guided Trajectory Generation. Given the task instruction τ and initial environment context, the meta planner Π_ψ constructs an overall interaction plan P:

$$P \sim \Pi_\psi(\cdot \mid \tau, w, \tilde{s}_0). \tag{2}$$

The plan decomposes the task into an ordered sequence of atomic actions, with semantic targets and descriptions of the expected interface changes. Planning accounts for dependencies between steps, placing prerequisite operations before the actions that depend on them. For example, saving a file requires opening a save dialog, choosing a directory, entering a filename, and confirming.

Voyager, denoted by V, refines the overall plan into step-specific scene descriptions. At step t, it uses plan P and screenshot $\tilde{s}_t$ to produce a detailed scene prompt δ_t for the next screenshot:

$$\delta_t \sim V(\cdot \mid P, \tilde{s}_t, t). \tag{3}$$

Here, t identifies the current plan step. The scene descriptions in the plan specify the intended interface changes; Voyager grounds these changes in the current screenshot and elaborates them into a detailed visual description, including the elements to change and the scene content to preserve. For example, when opening a save dialog, Voyager uses the current editor view to describe the dialog and its relation to the background window.

Image2 serves as the visual generator M_ϕ^V. It uses the current screenshot as an image reference and edits it according to δ_t to produce the next screenshot:

$$\tilde{s}_{t+1} \sim M_\phi^V(\cdot \mid \tilde{s}_t, \delta_t). \tag{4}$$

As illustrated in Figure 3, the generated screenshot is then returned to Voyager to refine the scene description for the following step. Through this step-by-step feedback, each generation step builds on the existing image, helping carry layout, appearance, and task-relevant content across the trajectory. Subsequent data curation checks consistency between actions and generated visual changes.

Action Annotation. During trajectory generation, Image2 follows Voyager's scene prompt to directly synthesize the screen expected after a planned action succeeds. This process provides a semantic action description and its intended visual outcome, but no interaction is executed to obtain the corresponding action coordinates. Spatial actions such as clicking and dragging therefore require an additional grounding step that localizes their semantic targets in the pre-action screenshot, providing coordinate supervision for GUI-agent training.

We use LocateAnything (Wang et al., 2026b) to predict a bounding box b_t from the target description e_t and pre-action screenshot $\tilde{s}_t$, and use it to construct spatial action labels. Actions such as typing and hotkeys require no additional localization.



Figure 3: Step-by-step trajectory generation. Guided by the overall interaction plan, Voyager observes the current screenshot and produces a detailed prompt describing the next scene. Image2 edits the current screenshot accordingly. The resulting screenshot is returned to Voyager for the next step. The example illustrates the progression of actions and generated observations.

Spatial annotations are drawn on a separate copy of the pre-action screenshot, keeping the policy input and subsequent generation reference free of overlays. The overall plan P and scene prompts δ_t are excluded from policy inputs. Each retained trajectory supplies step-level examples pairing the task, clean screenshot, and observation–action history with the target action. Section 4 details annotation checks and grounding repair.

4 Dataset Construction and Analysis

4.1 Dataset Overview

Table 1 summarizes the filtered training set of 79,266 step-level samples spanning Ubuntu, Windows, macOS, and Chrome. Each instance pairs a clean pre-action screenshot, task instruction, and prior history with an action. Grounding annotations supply spatial targets, serialized as coordinate-based tool calls by the AGW-35B harness (Appendix E.1). The generator is absent during policy training. The transition audit and trajectory statistics use separate cohorts rather than a shared training-set denominator.

Table 1: Retained training samples by domain. Counts are step-level samples, not complete trajectories; percentages are rounded.

Domain	Samples	Share
Ubuntu	24,250	30.6%
Windows	14,779	18.6%
macOS	5,028	6.3%
Chrome	35,209	44.4%
Total	79,266	100%

4.2 Quality Control and Filtering

A Gemini VLM checker receives the task, serialized action, clean before/after screenshots, and an annotated target frame for pointing actions. It checks target existence, box correctness and extent, action validity, and agreement between the action and the generated visual change. Code validates the structured verdict and routes steps to repair, removal, or retention. We reground 3,390 boxes whose targets remain visible and check them again; steps with absent targets, invalid actions, or unrepairable grounding are removed. Transition failures are filtered after checking the input action and target. Appendix D provides the QC prompts, routing rules and failure cases.

Table 2 summarizes stage totals and platform-level endpoints. Desktop counts are measured; Chrome counts before the final stage are estimated using desktop retention ratios. Final retained counts are reported directly. Table 1 gives the final platform shares. The 3.04% audit rate below measures pre-filtering desktop action–image inconsistency, not the residual error rate of the final training set.

Table 2: Training-data filtering. † marks counts that include estimates for Chrome; final counts are reported.

Stage	Samples	Change†
Generated	84,373†	—
VLM QC	75,314†	−9,059
Box repair	80,849†	+5,535
Final set	79,266	−1,583
Domain	**Generated**	**Retained**
Ubuntu	25,765	24,250
Windows	15,821	14,779
macOS	5,310	5,028
Chrome	37,477†	35,209

Table 3 separates overall inconsistencies from those remaining with valid input specifications. Within the latter subset, rates are 7.58% for drag, 7.09% for scroll, 6.35% for text entry, 3.23% for hotkeys, 2.16% for key presses, and 0.67% for clicks. Examples include command substitution, premature formatting, missing action effects, and unintended changes to persistent content. Action and platform composition may contribute to these differences. VLM judgments are not execution-based ground truth, and checker error has not been calibrated by a human audit.

Table 3: VLM-flagged action–image inconsistencies in the desktop audit. The second row conditions on valid input actions and acceptable or inapplicable target/grounding checks. These are audit rates before removing transition failures, not estimated error rates of the final retained data.

Audit cohort	Transitions	Flagged	Rate
All evaluable desktop transitions	42,526	1,293	3.04%
Valid action and target subset	39,351	818	2.08%

4.3 Visual Distribution and Appearance

We compare synthetic screenshots with real corpora within each OS domain, using ScaleCUA as the resolution-matched reference. In Qwen3.5-9B's mean-pooled 4096-dimensional PatchMerger space, ρ divides synthetic–ScaleCUA maximum mean discrepancy (MMD) by median real–real MMD. Figure 4a reports $\rho = 1.24/0.85/1.14/0.59$ for Ubuntu/Windows/Web/macOS with group-bootstrap 95% intervals. Windows and macOS fall below their real-source references, while Ubuntu and Web are 24% and 14% above them. The ratios compare empirical distance scales; Windows and macOS each have only one real-source pair. Dataset identity remains detectable: classifier two-sample test (C2ST) AUC is about 1.00 for synthetic–real and 0.95–1.00 for real–real pairs. DINOv2 exposes a larger Ubuntu gap, showing that the pattern depends on the representation. Appendix C.3 retains source-pair distances, reference checks, and independent-encoder results.

Random-patch residuals show no common high-frequency decay across domains: Ubuntu synthetic patches have lower high-frequency power than the real-source range, whereas Web patches have higher power. Figure 4b shows that, after matching local content in regions of interest (ROIs) by edge density, brightness, and color entropy, text/icon C2ST AUC is 0.44–0.57 on Ubuntu and Windows, while Ubuntu backgrounds retain an AUC of 0.65. These local probes do not imply indistinguishable full screens. Appendix C.4 gives full frequency, edge, and OCR diagnostics. Token count depends on image resolution and aspect ratio: fixed-resolution desktop synthesis yields 1,792 tokens per image. Count differences between sources therefore need not indicate semantic differences. Appendix C.2 separates token-count and resolution effects.

4.4 Interface Coverage and Trajectory Structure

Figure 4c shows that detected UI regions cover more screen area than ScaleCUA in all four domains at all three confidence thresholds. OmniParser-v2.0 detects the regions, and coverage measures the union of their boxes, counting overlapping area once. At threshold 0.15, synthetic coverage is

29.2%, 35.1%, 33.2%, and 35.9% on Ubuntu, Windows, Web, and macOS, respectively, compared with 17.0%, 21.2%, 20.0%, and 31.9% for ScaleCUA.

Figure 4d reports the mean detected elements per page at threshold 0.15. Synthetic pages contain 103/142/68/122 elements across the four domains, versus 98/154/48/85 in ScaleCUA. Windows illustrates why count and coverage should be read together: it has fewer elements yet 13.9 percentage points more covered area. This reflects broader screen occupancy rather than uniformly higher counts. A manual audit of 180 overlays found no systematic texture over-detection. Appendix C.5 gives full threshold results and OCR examples.

Synthetic trajectories average 11.9/11.4/7.1 steps on Ubuntu/Windows/macOS; their 90th percentiles are 24/25/14 steps. macOS has more open applications (2.50 on average, versus 1.62/2.06 on Ubuntu/Windows) and higher normalized action-type entropy (0.92 versus 0.79/0.82), despite its shorter sequences. These complementary structural statistics have no matched real-trajectory control. Full results and interpretation are in Appendix C.6.

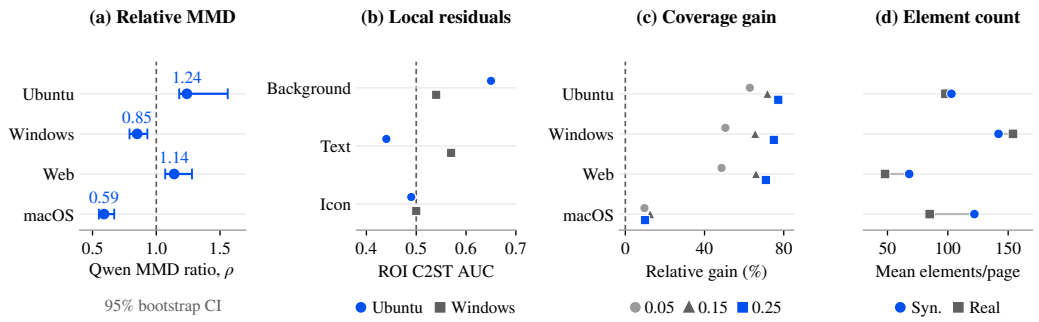

Figure 4: Data-feature summary. (a) Relative MMD with 95% bootstrap intervals. (b) Content-matched ROI C2ST AUC. (c) Relative union-coverage gains over ScaleCUA at thresholds 0.05/0.15/0.25. (d) Mean elements per page at threshold 0.15, using the domain rows in (c). Syn./Real denote synthetic/ScaleCUA. Appendix C provides full figures and results.

5 Experiments

5.1 Experimental Setup

We evaluate AutoGUIWorld on two types of benchmarks: **real-world task execution** and **visual grounding**. We fine-tune Qwen3.5-35B-A3B on AutoGUIWorld data for three epochs to obtain AGW-35B and compare it with the base model. For real-world task execution, we use OSWorld (Xie et al., 2024), WindowsAgentArena (WAA) (Bonatti et al., 2024), macOSWorld (Yang et al., 2025), and ScienceBoard (Sun et al., 2025) to assess agents' ability to complete multi-step tasks in software environments. We report mean task scores for OSWorld and WAA, and success rates for macOSWorld and ScienceBoard. For visual grounding, we evaluate on 1,581 English examples from ScreenSpot-Pro (Li et al., 2025b), measuring whether the predicted coordinates fall within the target bounding box. Appendices E and F detail the training and evaluation configurations.

5.2 Task Execution in Real Environments

Table 4 shows improvements on all four interactive benchmarks. OSWorld rises from 33.0% to 40.8%, and ScienceBoard from 14.0% to 32.2%. The equal-weight mean of the four benchmark scores rises from 23.6% to 36.5% (+12.8 percentage points, calculated before rounding).

Table 4: Interactive evaluation (%). AGW-35B: checkpoint 469; Δ: gain in percentage points, computed before rounding.

Benchmark	Tasks	Base	AGW-35B	Δ
OSWorld	361	32.98	**40.77**	+7.79
WAA	154	19.41	**27.92**	+8.51
macOSWorld	231	28.14	**45.02**	+16.88
ScienceBoard	143	13.99	**32.17**	+18.18

Domain-level transfer. Across the four benchmarks, 25 of 35 domains improve, seven remain unchanged, and three decline. OSWorld multi-app scores rise from 12.0% to 20.9%; macOSWorld multi-app success rises from 13.8% to 27.6%. ScienceBoard improves in all five domains, with KAlgebra and ChimeraX accounting for 18 of the 26 net task successes. Transfer is uneven: Chrome declines from 39.0% to 26.0% on OSWorld and from 11.2% to 0.0% on WAA; macOSWorld Safety declines from 20.7% to 17.2%. Appendix F.4 provides all domain results and figures.

Performance over training. Every interactive benchmark improves over the base by step 80, and all four reach their highest evaluated score at step 469. ScienceBoard improves at each checkpoint, while OSWorld and WAA fluctuate. The four-benchmark mean increases at every evaluation. Figure 5 shows the progression; Appendix Table 23 gives AGW-35B checkpoint scores. These checkpoints are from one run and are not independent repetitions.

Synthetic versus real post-training data. AgentNet contains human demonstrations from real desktop environments (Wang et al., 2025a). Both runs start from Qwen3.5-35B-A3B: AGW-35B uses approximately 80k synthetic examples, while the AgentNet run uses 350k examples and continues through step 686. AGW-35B at step 469 exceeds the best evaluated AgentNet checkpoint on all four benchmarks, with the largest gap on ScienceBoard (32.2% versus 16.8%). These results show that synthetic GUI data can match or exceed real demonstrations for post-training in this comparison. The runs use different inference settings (Appendix F.1).

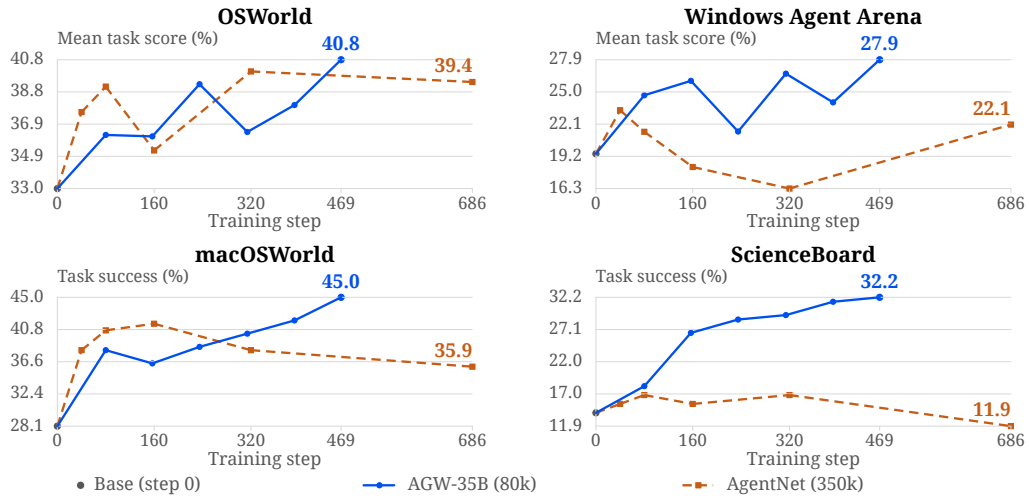

Figure 5: Post-training performance of AGW-35B (blue circles) and the AgentNet-trained model (orange dashed lines and squares). Gray marks the base, shared across all four benchmarks. Curves use actual training steps and panel-specific score ranges.

5.3 Grounding in Professional Interfaces

Table 5 reports a 25.4-point overall gain, from 501 to 902 correct predictions. Text accuracy improves by 30.7 points and icon accuracy by 16.7 points. Both target types improve in every application domain. Office has the largest gains: 41.2 points for text and 24.5 for icons. Scientific text accuracy reaches 79.9%, while icon accuracy stays below text accuracy throughout.

Training progression and interpretation. Results come from one training run; checkpoints are not independent repetitions. All six improve overall accuracy over the base model, ranging from 55.5% to 57.1%, without a monotonic increase across training. Invalid-format responses fall from 226 to 55 between the base and final models, so the aggregate improvement includes better output-format adherence and cannot be attributed entirely to spatial localization. Appendix E preserves these counts and domain breakdowns. The comparison establishes a gain over the base model under this protocol. Task-completion gains are measured separately in Section 5.2.

Table 5: ScreenSpot-Pro accuracy (%) overall and across six application domains. AGW denotes AGW-35B at checkpoint 469; Δ is the gain in percentage points, computed before rounding. N counts all examples; text and icon accuracies use their respective subsets.

		Overall			Text			Icon		
Domain	N	Base	AGW	Δ	Base	AGW	Δ	Base	AGW	Δ
All apps	1,581	31.7	**57.1**	+25.4	41.2	**72.0**	+30.7	16.2	**32.9**	+16.7
CAD	261	28.7	**47.9**	+19.2	36.0	**55.8**	+19.8	6.3	**23.4**	+17.2
Dev.	299	33.8	**53.8**	+20.1	48.7	**76.0**	+27.3	17.9	**30.3**	+12.4
Creative	341	31.4	**58.9**	+27.6	40.9	**74.7**	+33.8	18.2	**37.1**	+18.9
Scientific	254	31.1	**59.4**	+28.3	39.6	**79.9**	+40.3	20.0	**32.7**	+12.7
Office	230	33.9	**71.3**	+37.4	39.0	**80.2**	+41.2	17.0	**41.5**	+24.5
OS	196	31.1	**51.0**	+19.9	46.7	**66.4**	+19.6	12.4	**32.6**	+20.2

6 What Should a GUI World Model Model?

Must a GUI world model reproduce the full software state? The required fidelity depends on its intended use. For training a screenshot-based policy, AutoGUIWorld targets visible action consequences rather than reconstructing application internals. This makes synthesis possible without implementing each target application, but screenshots alone cannot certify file persistence, hidden preconditions, or task completion. The interactive and grounding gains in Section 5 support this use as a source of supervision; they do not establish the reliability needed for an execution environment or a reward model. An observation-level model still needs history-sensitive transitions and faithful visible content, even when full software-state reconstruction is outside its scope.

How should visual generation be coupled with explicit control? AutoGUIWorld separates task generation, overall action planning, Voyager scene refinement, spatial grounding, and transition inspection. The failure cases in Appendix D motivate this separation: rendering a completed artifact after one text-entry action can satisfy a task-level intention while violating the atomic action; a cursor movement should not rewrite persistent code. These observations suggest making the permitted change and the content that must remain fixed explicit in the generation interface. They do not show that the current controller enforces those constraints reliably. VLM checks provide a practical filter, while human or executable validation remains necessary to assess checker errors. The design implication is to evaluate visual generation together with its action constraints and verification mechanism. Controlled comparisons under matched training budgets and component ablations remain necessary to isolate the effects of the data source and individual generation components.

Is visual similarity sufficient to assess training utility? Section 4.3 shows that synthetic and real screenshots remain distinguishable in global visual representations, while Section 5 reports improved task execution after training on the generated trajectories. A detectable visual distribution gap can therefore coexist with useful supervision, although these results do not identify which visual properties drive the gains. GUI world models should be assessed jointly through visual distributions, action–transition consistency, and the downstream value of their generated data. Coverage and similarity describe data properties; neither substitutes for evaluating task execution in real environments.

7 Conclusion

AutoGUIWorld combines structured GUI world sampling, scene-conditioned tasks, and planner-guided visual transitions to generate agent training trajectories without deploying or running the corresponding software environments. Quality control filters defective actions, annotations, and transitions; data analysis characterizes visual alignment and interface coverage. Fine-tuning improves real task execution on four desktop and scientific benchmarks, as well as grounding in professional interfaces. These gains support image generators as a source of GUI supervision. Uneven domain transfer and imperfect visual transitions motivate stronger verification and matched-budget controls to isolate the contributions of the data source and generation components.

AI USE STATEMENT

Generative models are central to AutoGUIWorld's data-generation pipeline: they generate tasks, plan interaction sequences, refine scene descriptions, and synthesize screenshots. Grounding models provide spatial action annotations, and a vision-language model supports quality checking and filtering. During manuscript preparation, an AI assistant helped draft and revise text, organize related work, and edit LaTeX formatting. The authors remain responsible for the manuscript's claims, citations, results, and content. Model-based quality checks are distinguished from execution-based benchmark evaluation throughout the paper.

ETHICS STATEMENT

AutoGUIWorld is intended to support research on GUI agents through synthetic interaction data. Generating screenshots reduces the need to record private user interactions, but synthetic data can still inherit model biases and contain misleading text or incorrect action consequences. Our quality filters do not establish that all retained samples are correct or safe. Improved GUI automation can also enable unauthorized operations; practical deployment requires appropriate permissions and safeguards for sensitive actions. Benchmark gains should not be interpreted as evidence of safe autonomous deployment, particularly in specialized scientific workflows. Redistribution of generated interfaces and third-party materials should respect applicable usage and licensing restrictions.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix A describe the generation pipeline, while Appendix B specifies the action interface. Section 4 and Appendix C report dataset statistics and analysis protocols; Appendix D provides quality-checking prompts, filtering rules, and failure cases. Training hyperparameters, message construction, and the grounding evaluation protocol are documented in Appendix E. Appendix F specifies execution environments, benchmark subsets, scoring, and checkpoint results. These details document the reported experiments; reproducing data generation also requires access to the generation models and their corresponding versions.

REFERENCES

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-Anything: Turn any Software into an Agent Environment. *arXiv preprint arXiv:2604.06126*, 2026. doi: 10.48550/arXiv.2604.06126. URL https://arxiv.org/abs/2604.06126.

Jiaxin Ai, Yukang Feng, Fanrui Zhang, Jianwen Sun, Zizhen Li, Chuanhao Li, Yifan Chang, Wenxiao Wu, Ruoxi Wang, Mingliang Zhai, and Kaipeng Zhang. ProSoftArena: Benchmarking Hierarchical Capabilities of Multimodal Agents in Professional Software Environments. *arXiv preprint arXiv:2601.02399*, 2025. doi: 10.48550/arXiv.2601.02399. URL https://arxiv.org/abs/2601.02399.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal OS agents at scale. *arXiv preprint arXiv:2409.08264*, 2024. doi: 10.48550/arXiv.2409.08264. URL https://arxiv.org/abs/2409.08264.

Yilin Cao, Yufeng Zhong, Zhixiong Zeng, Liming Zheng, Jing Huang, Haibo Qiu, Peng Shi, Wenji Mao, and Guanglu Wan. MobileDreamer: Generative sketch world model for GUI agent. *arXiv preprint arXiv:2601.04035*, 2026. doi: 10.48550/arXiv.2601.04035. URL https://arxiv.org/abs/2601.04035.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*, 2024. doi: 10.48550/arXiv.2410.13232. URL https://arxiv.org/abs/2410.13232.

Mingkai Deng, Jinyu Hou, Zhiting Hu, and Eric Xing. General agentic planning through simulative reasoning with world models. *arXiv preprint arXiv:2507.23773*, 2025. doi: 10.48550/arXiv.2507.23773. URL https://arxiv.org/abs/2507.23773.

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. *arXiv preprint arXiv:2306.06070*, 2023. doi: 10.48550/arXiv.2306.06070. URL https://arxiv.org/abs/2306.06070.

Hang Ding, Peidong Liu, Junqiao Wang, Ziwei Ji, Meng Cao, Rongzhao Zhang, Lynn Ai, Eric Yang, Tianyu Shi, and Lei Yu. DynaWeb: Model-based reinforcement learning of web agents. *arXiv preprint arXiv:2601.22149*, 2026. doi: 10.48550/arXiv.2601.22149. URL https://arxiv.org/abs/2601.22149.

Zihan Dong, Yuanzhe Liu, Zhiyuan Ma, Qishi Zhan, Dehan Kong, Guohao Li, and Kaixin Li. CAD-World: Computer-Use Benchmark for Long-Horizon Computer-Aided Design. *arXiv preprint arXiv:2609.16251*, 2026. doi: 10.48550/arXiv.2609.16251. URL https://arxiv.org/abs/2609.16251.

FAIR CodeGen Team. CWM: An open-weights LLM for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025. doi: 10.48550/arXiv.2510.02387. URL https://arxiv.org/abs/2510.02387.

Zhuohang Fan, Beichen Zhang, Yuanfa Li, Changqiao Wu, Wei Liu, Jian Luan, and Weigang Zhang. SEE: Structure-aware exploring and exploiting for long-horizon GUI agent trajectory synthesis. *arXiv preprint arXiv:2607.18046*, 2026. doi: 10.48550/arXiv.2607.18046. URL https://arxiv.org/abs/2607.18046.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. WebEvolver: Enhancing web agent self-improvement with coevolving world model. *arXiv preprint arXiv:2504.21024*, 2025. doi: 10.48550/arXiv.2504.21024. URL https://arxiv.org/abs/2504.21024.

Lin Fu, Zheyuan Yang, Tianhui Zhang, Jinbiao Wei, Guo Gan, Boxu Liu, Yilun Zhao, and Yu Rong. GUI-CC: Benchmarking contextual consistency of GUI world models as agent environments. *arXiv preprint arXiv:2609.00048*, 2026. doi: 10.48550/arXiv.2609.00048. URL https://arxiv.org/abs/2609.00048.

Valentin Gabeur, Shangbang Long, Songyou Peng, Paul Voigtlaender, Shuyang Sun, Yanan Bao, Karen Truong, Zhicheng Wang, Wenlei Zhou, Jonathan T. Barron, Kyle Genova, Nithish Kannen, Sherry Ben, Yandong Li, Mandy Guo, Suhas Yogin, Yiming Gu, Huizhong Chen, Oliver Wang, Saining Xie, Howard Zhou, Kaiming He, Thomas Funkhouser, Jean-Baptiste Alayrac, and Radu Soricut. Image generators are generalist vision learners. *arXiv preprint arXiv:2604.20329*, 2026. doi: 10.48550/arXiv.2604.20329. URL https://arxiv.org/abs/2604.20329.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024. doi: 10.48550/arXiv.2411.06559. URL https://arxiv.org/abs/2411.06559.

Yiming Guan, Rui Yu, John Zhang, Lu Wang, Chaoyun Zhang, Liqun Li, Bo Qiao, Si Qin, He Huang, Fangkai Yang, Pu Zhao, Lukas Wutschitz, Samuel Kessler, Huseyin A. Inan, Robert Sim, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Computer-using world model. *arXiv preprint arXiv:2602.17365*, 2026. doi: 10.48550/arXiv.2602.17365. URL https://arxiv.org/abs/2602.17365.

Yifei He, Pranit Chawla, Yaser Souri, Subhojit Som, and Xia Song. WebSTAR: Scalable data synthesis for computer use agents with step-level filtering. *arXiv preprint arXiv:2512.10962*, 2025. doi: 10.48550/arXiv.2512.10962. URL https://arxiv.org/abs/2512.10962.

Yunseok Jang, Yeda Song, Sungryull Sohn, Lajanugen Logeswaran, Tiange Luo, Dong-Ki Kim, Kyunghoon Bae, and Honglak Lee. Scalable video-to-dataset generation for cross-platform mobile agents. *arXiv preprint arXiv:2505.12632*, 2025. doi: 10.48550/arXiv.2505.12632. URL https://arxiv.org/abs/2505.12632.

Deyang Jiang, Jing Huang, Xuanle Zhao, Lei Chen, Liming Zheng, Fanfan Liu, Haibo Qiu, Peng Shi, and Zhixiong Zeng. TreeCUA: Efficiently scaling GUI automation with tree-structured verifiable evolution. *arXiv preprint arXiv:2602.09662*, 2026. doi: 10.48550/arXiv.2602.09662. URL https://arxiv.org/abs/2602.09662.

Linjia Kang, Zhimin Wang, Yongkang Zhang, Duo Wu, Jinghe Wang, Ming Ma, Haopeng Yan, and Zhi Wang. Learning with challenges: Adaptive difficulty-aware data generation for mobile GUI agent training. *arXiv preprint arXiv:2601.22781*, 2026. doi: 10.48550/arXiv.2601.22781. URL https://arxiv.org/abs/2601.22781.

Woosung Koh, Sungjun Han, Segyu Lee, Se-Young Yun, and Jamin Shin. Generative visual code mobile world models. *arXiv preprint arXiv:2602.01576*, 2026. doi: 10.48550/arXiv.2602.01576. URL https://arxiv.org/abs/2602.01576.

Chunyi Li, Longfei Li, Zicheng Zhang, Xiaohong Liu, Min Tang, Weisi Lin, and Guangtao Zhai. Using GUI Agent for Electronic Design Automation. *arXiv preprint arXiv:2512.11611*, 2025a. doi: 10.48550/arXiv.2512.11611. URL https://arxiv.org/abs/2512.11611.

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. *arXiv preprint arXiv:2504.07981*, 2025b. doi: 10.48550/arXiv.2504.07981. URL https://arxiv.org/abs/2504.07981.

Musen Lin, Minghao Liu, Taoran Lu, Lichen Yuan, Yiwei Liu, Haonan Xu, Yu Miao, Yuhao Chao, and Zhaojian Li. GUI-ReWalk: Massive data generation for GUI agent via stochastic exploration and intent-aware reasoning. *arXiv preprint arXiv:2509.15738*, 2025. doi: 10.48550/arXiv.2509.15738. URL https://arxiv.org/abs/2509.15738.

Zhaoyang Liu, Jingjing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, Shenglong Ye, Qingyun Li, Xuan Dong, Yue Yu, Chenyu Lu, YunXiang Mo, Yao Yan, Zeyue Tian, Xiao Zhang, Yuan Huang, Yiqian Liu, Weijie Su, 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. *arXiv preprint arXiv:2509.15221*, 2025. doi: 10.48550/arXiv.2509.15221. URL https://arxiv.org/abs/2509.15221.

Dunjie Lu, Yiheng Xu, Junli Wang, Haoyuan Wu, Xinyuan Wang, Zekun Wang, Junlin Yang, Hongjin Su, Jixuan Chen, Junda Chen, Yuchen Mao, Jingren Zhou, Junyang Lin, Binyuan Hui, and Tao Yu. VideoAgentTrek: Computer use pretraining from unlabeled videos. *arXiv preprint arXiv:2510.19488*, 2025a. doi: 10.48550/arXiv.2510.19488. URL https://arxiv.org/abs/2510.19488.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. GUIOdyssey: A comprehensive dataset for cross-app GUI navigation on mobile devices. *arXiv preprint arXiv:2406.08451*, 2024. doi: 10.48550/arXiv.2406.08451. URL https://arxiv.org/abs/2406.08451.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024. doi: 10.48550/arXiv.2402.05930. URL https://arxiv.org/abs/2402.05930.

Zhengxi Lu, Jiabo Ye, Fei Tang, Yongliang Shen, Haiyang Xu, Ziwei Zheng, Weiming Lu, Ming Yan, Fei Huang, Jun Xiao, and Yueting Zhuang. UI-S1: Advancing GUI automation via semi-online reinforcement learning. *arXiv preprint arXiv:2509.11543*, 2025b. doi: 10.48550/arXiv.2509.11543. URL https://arxiv.org/abs/2509.11543.

Dezhao Luo, Bohan Tang, Kang Li, Georgios Papoudakis, Jifei Song, Shaogang Gong, Jianye Hao, Jun Wang, and Kun Shao. ViMo: A generative visual GUI world model for app agents. *arXiv preprint arXiv:2504.13936*, 2025. doi: 10.48550/arXiv.2504.13936. URL https://arxiv.org/abs/2504.13936.

Bo Pang, Jiaqi Pan, Xiaocheng Zhang, Jiacheng Xu, Guoping Wang, and Peng-Shuai Wang. ViSculpt: Visual-Centric Agentic Geometry Editing. *arXiv preprint arXiv:2608.24169*, 2026. doi: 10.48550/arXiv.2608.24169. URL https://arxiv.org/abs/2608.24169.

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. *arXiv preprint arXiv:2501.12326*, 2025. doi: 10.48550/arXiv.2501.12326. URL https://arxiv.org/abs/2501.12326.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023. doi: 10.48550/arXiv.2307.10088. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024. doi: 10.48550/arXiv.2405.14573. URL https://arxiv.org/abs/2405.14573.

Luke Rivard, Sun Sun, Hongyu Guo, Wenhu Chen, and Yuntian Deng. NeuralOS: Towards simulating operating systems via neural generative models. *arXiv preprint arXiv:2507.08800*, 2025. doi: 10.48550/arXiv.2507.08800. URL https://arxiv.org/abs/2507.08800.

Rui Shao, Ruize Gao, Bin Xie, Yixing Li, Kaiwen Zhou, Shuai Wang, Weili Guan, and Gongwei Chen. HATS: Hardness-aware trajectory synthesis for GUI agents. *arXiv preprint arXiv:2603.12138*, 2026. doi: 10.48550/arXiv.2603.12138. URL https://arxiv.org/abs/2603.12138.

Chan Hee Song, Yiwen Song, Palash Goyal, Yu Su, Oriana Riva, Hamid Palangi, and Tomas Pfister. Watch and learn: Learning to use computers from online videos. *arXiv preprint arXiv:2510.04673*, 2025. doi: 10.48550/arXiv.2510.04673. URL https://arxiv.org/abs/2510.04673.

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. *arXiv preprint arXiv:2412.19723*, 2024. doi: 10.48550/arXiv.2412.19723. URL https://arxiv.org/abs/2412.19723.

Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, Jianing Wang, Qintong Li, Xiangru Tang, Tianbao Xie, Xiachong Feng, Xiang Li, Ben Kao, Wenhai Wang, Biqing Qi, Lingpeng Kong, and Zhiyong Wu. ScienceBoard: Evaluating multimodal autonomous agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*, 2025. doi: 10.48550/arXiv.2505.19897. URL https://arxiv.org/abs/2505.19897.

Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. *arXiv preprint arXiv:2402.12275*, 2024. doi: 10.48550/arXiv.2402.12275. URL https://arxiv.org/abs/2402.12275.

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. *arXiv preprint arXiv:2605.25624*, 2026a. doi: 10.48550/arXiv.2605.25624. URL https://arxiv.org/abs/2605.25624.

Shihao Wang, Shilong Liu, Yuanguo Kuang, Xinyu Wei, Yangzhou Liu, Zhiqi Li, Yunze Man, Guo Chen, Andrew Tao, Guilin Liu, Jan Kautz, Lei Zhang, and Zhiding Yu. Locateanything: Fast and high-quality vision-language grounding with parallel box decoding. *arXiv preprint arXiv:2605.27365*, 2026b. doi: 10.48550/arXiv.2605.27365. URL https://arxiv.org/abs/2605.27365.

Wenkai Wang, Tao Xiong, Jingchen Ni, Yunpeng Bao, Xiyun Li, Tianqi Liu, Hongcan Guo, Zilong Huang, and Shengyu Zhang. DeskCraft: Benchmarking Desktop Agents on Professional Workflows and Human-in-the-Loop Collaboration. *arXiv preprint arXiv:2606.03103*, 2026c. doi: 10.48550/arXiv.2606.03103. URL https://arxiv.org/abs/2606.03103.

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, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025a. doi: 10.48550/arXiv.2508.09123. URL https://arxiv.org/abs/2508.09123.

Yiming Wang, Da Yin, Yuedong Cui, Ruichen Zheng, Zhiqian Li, Zongyu Lin, Di Wu, Xueqing Wu, Chenchen Ye, Yu Zhou, and Kai-Wei Chang. LLMs as scalable, general-purpose simulators for evolving digital agent training. *arXiv preprint arXiv:2510.14969*, 2025b. doi: 10.48550/arXiv.2510.14969. URL https://arxiv.org/abs/2510.14969.

Yifan Wu, Yiran Peng, Yiyu Chen, Jianhao Ruan, Zijie Zhuang, Cheng Yang, Jiayi Zhang, Man Chen, Yenchi Tseng, Zhaoyang Yu, Liang Chen, Yuyao Zhai, Bang Liu, Chenglin Wu, and Yuyu Luo. AutoWebWorld: Synthesizing infinite verifiable web environments via finite state machines. *arXiv preprint arXiv:2602.14296*, 2026. doi: 10.48550/arXiv.2602.14296. URL https://arxiv.org/abs/2602.14296.

Jiannan Xiang, Yun Zhu, Lei Shu, Maria Wang, Lijun Yu, Gabriel Barcik, James Lyon, Srinivas Sunkara, and Jindong Chen. UISim: An interactive image-based UI simulator for dynamic mobile environments. *arXiv preprint arXiv:2509.21733*, 2025. doi: 10.48550/arXiv.2509.21733. URL https://arxiv.org/abs/2509.21733.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, Shuai Ren, Yafei Wen, Xiaoxin Chen, Aojun Zhou, and Hongsheng Li. UI-Genie: A self-improving approach for iteratively boosting MLLM-based mobile GUI agents. *arXiv preprint arXiv:2505.21496*, 2025. doi: 10.48550/arXiv.2505.21496. URL https://arxiv.org/abs/2505.21496.

Zikai Xiao, Jianhong Tu, Chuhang Zou, Yuxin Zuo, Zhi Li, Peng Wang, Bowen Yu, Fei Huang, Junyang Lin, and Zuozhu Liu. WebWorld: A large-scale world model for web agent training. *arXiv preprint arXiv:2602.14721*, 2026. doi: 10.48550/arXiv.2602.14721. URL https://arxiv.org/abs/2602.14721.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent. *arXiv preprint arXiv:2505.16827*, 2025a. doi: 10.48550/arXiv.2505.16827. URL https://arxiv.org/abs/2505.16827.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. AgentSynth: Scalable task generation for generalist computer-use agents. *arXiv preprint arXiv:2506.14205*, 2025b. doi: 10.48550/arXiv.2506.14205. URL https://arxiv.org/abs/2506.14205.

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. *arXiv preprint arXiv:2404.07972*, 2024. doi: 10.48550/arXiv.2404.07972. URL https://arxiv.org/abs/2404.07972.

Weimin Xiong, Shuhao Gu, Bowen Ye, Zihao Yue, Lei Li, Feifan Song, Sujian Li, and Hao Tian. Video2GUI: Synthesizing large-scale interaction trajectories for generalized GUI agent pretraining. *arXiv preprint arXiv:2605.14747*, 2026. doi: 10.48550/arXiv.2605.14747. URL https://arxiv.org/abs/2605.14747.

Weikai Xu, Kun Huang, Yunren Feng, Jiaxing Li, Yuhan Chen, Yuxuan Liu, Zhizheng Jiang, Heng Qu, Pengzhi Gao, Wei Liu, Jian Luan, Xiaolin Hu, and Bo An. How mobile world model guides GUI agents? *arXiv preprint arXiv:2605.10347*, 2026. doi: 10.48550/arXiv.2605.10347. URL https://arxiv.org/abs/2605.10347.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*, 2024. doi: 10.48550/arXiv.2412.09605. URL https://arxiv.org/abs/2412.09605.

Pei Yang, Hai Ci, and Mike Zheng Shou. macOSWorld: A multilingual interactive benchmark for GUI agents. *arXiv preprint arXiv:2506.04135*, 2025. doi: 10.48550/arXiv.2506.04135. URL https://arxiv.org/abs/2506.04135.

Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. TongUI: Internet-scale trajectories from multimodal web tutorials for generalized GUI agents. *arXiv preprint arXiv:2504.12679*, 2025. doi: 10.48550/arXiv.2504.12679. URL https://arxiv.org/abs/2504.12679.

Ziyun Zhang, Zezhou Wang, Xiaoyi Zhang, Zongyu Guo, Jiahao Li, Bin Li, and Yan Lu. InfiniteWeb: Scalable web environment synthesis for GUI agent training. *arXiv preprint arXiv:2601.04126*, 2026. doi: 10.48550/arXiv.2601.04126. URL https://arxiv.org/abs/2601.04126.

Jiani Zheng, Lu Wang, Fangkai Yang, Chaoyun Zhang, Lingrui Mei, Wenjie Yin, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. VEM: Environment-free exploration for training GUI agent with value environment model. *arXiv preprint arXiv:2502.18906*, 2025. doi: 10.48550/arXiv.2502.18906. URL https://arxiv.org/abs/2502.18906.

Yuhao Zheng, Li'an Zhong, Yi Wang, Rui Dai, Kaikui Liu, Xiangxiang Chu, Linyuan Lv, Philip Torr, and Kevin Qinghong Lin. Code2world: A GUI world model via renderable code generation. *arXiv preprint arXiv:2602.09856*, 2026. doi: 10.48550/arXiv.2602.09856. URL https://arxiv.org/abs/2602.09856.

Liya Zhu, Jingzhe Ding, Jian Zhang, Jianbo Xue, Shihao Liang, Ge Zhang, Yi Zhu, Duju Zeng, Xiang Gao, Qingshui Gu, Mailun Gao, Huimin Che, Yan Zhao, Peiheng Zhou, Haojun Wang, Chaobo Xian, Lili Le, Chi Wu, Yiwei Liu, Shengda Long, Jiale Yang, Fangzhi Xu, Sijin Wu, Haodong Duan, Chao He, Zhaojian Li, Minchao Wang, Huan Zhou, Jiani Hou, Chuqian Yu, Weiran Shi, Hongwan Gao, Jiamin Chen, Guanhong Chen, Tingqin Luo, Kaiyuan Zhang, Zhixin Yao, Qing Hua, Yuhao Jiang, Jin Chen, Pu Chen, Zhenyu Hu, Xingyu Li, Zhengxuan Jiang, Meng Cao, Tianfeng Long, Haozhe Wang, Mingzhang Wang, Yichen Zhang, Yiming Dai, Chenchen Zhang, Jiaying Wang, Xinying Liu, Xingzu Liu, Lingling Zhang, Xinjie Chen, Yujia Qin, Wangchunshu Zhou, Zhiyong Wu, Yang Liu, Jiaheng Liu, Lei Zhang, Shen Yan, Wenhao Huang, Zaiyuan Wang, and Xiaolong Chang. Workflow-GYM: Towards Long-Horizon Evaluation of Computer-use Agentic tasks in Real-World Professional Fields. *arXiv preprint arXiv:2606.11042*, 2026. doi: 10.48550/arXiv.2606.11042. URL https://arxiv.org/abs/2606.11042.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, et al. Qwen-AgentWorld: Language world models for general agents. *arXiv preprint arXiv:2606.24597*, 2026. doi: 10.48550/arXiv.2606.24597. URL https://arxiv.org/abs/2606.24597.

APPENDIX CONTENTS

A Generation Details **17**

A.1 Observation-Level Formulation 17

A.2 Seed Context and Training Records . . 17

B Action Space **18**

C Supplementary Data Analysis **18**

C.1 Sources and Measurement Protocol . . 18

C.2 Token Counts and Resolution 19

C.3 Visual Representation Distributions . . 20

C.4 Low-Level Appearance and Generation Residuals 21

C.5 Interface Coverage, Detection, and OCR 22

C.6 Trajectory Structure and Interpretation 23

D Transition Fidelity and Data Filtering **24**

D.1 VLM Quality-Control Protocol 25

D.2 Transition Audit Results 27

D.3 Representative Failure Modes 28

E Grounding Evaluation Details **29**

E.1 Training and Inference Harness 29

E.2 ScreenSpot-Pro Protocol 29

E.3 Checkpoint Progression and Domain Results 30

F Evaluation Infrastructure **31**

F.1 Agent Execution and Task Sets 31

F.2 ScienceBoard Task Selection 32

F.3 Score Aggregation 33

F.4 Interactive Domain Results and Training Progression 33

G Extended Related Work **36**

G.1 GUI Trajectory Collection 36

G.2 GUI World Model 36

H Qualitative Trajectory Examples **37**

H.1 Ubuntu Desktop 37

H.2 Windows 11 41

H.3 macOS 45

H.4 Chrome 49

I Professional Application Workflows **53**

I.1 Engineering CAD and 3D Authoring . 54

I.2 Visual and Media Design 56

I.3 Data Analysis and Econometrics . . . 58

I.4 Electronic Design Automation 59

I.5 Audio Production 60

I.6 Scientific Computing and Research . . 61

I.7 Development and Automation 63

J Qualitative Seed Examples **64**

J.1 Ubuntu Desktop 64

J.2 Windows 11 67

J.3 macOS 70

J.4 Chrome 73

A GENERATION DETAILS

A.1 OBSERVATION-LEVEL FORMULATION

GUI trajectory. We cast GUI interaction as a partially observable Markov decision process (POMDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \tilde{\mathcal{S}}, \Omega)$ together with a task instruction τ. The latent state $s_t \in \mathcal{S}$ is the full computer configuration at step t, including operating-system state, application internals, file-system contents, and any hidden controller state, and the transition $T(s_{t+1} \mid s_t, a_t)$ is governed by the host system. The agent never reads s_t directly; it only observes the rendered screenshot $\tilde{s}_t = \Omega(s_t) \in \tilde{\mathcal{S}}$ and emits an atomic action $a_t \in \mathcal{A}$ from the OS-dependent action space listed in the appendix. Writing $h_t = (\tilde{s}_{\le t}, a_{<t})$ for the observation–action history, a GUI policy takes the form

$$a_t \sim \pi_\theta(\cdot \mid h_t, \tau). \tag{5}$$

We condition on the full history because single screenshots are generally non-Markov: scroll positions, dialog stacks, in-progress text edits, and dynamic page content all carry information that is not visible in $\tilde{s}_t$ alone.

Interaction fidelity. A screenshot-based agent never consumes s_t and never invokes T, so its training data is fully determined by the observation-level transition $K(\tilde{s}_{t+1} \mid h_t, a_t)$ induced by marginalising the latent dynamics. We refer to faithfulness with respect to K as *interaction fidelity*: a synthetic trajectory has high interaction fidelity if its post-action screenshot $\tilde{s}_{t+1}$ matches the distribution a real system would produce under the same history, even when the underlying s_{t+1} is never reconstructed. AutoGUIWorld is organised around this relaxation: we model K directly with a visual world model and never instantiate $\mathcal{S}$ or T.

Data-generation mechanism. In synthetic generation, the meta planner Π_ψ constructs an overall interaction plan P from the task τ, world specification w, and initial screenshot $\tilde{s}_0$. The plan specifies the ordered actions and intended interface changes. At each step, Voyager V conditions on this plan, the current screenshot, and the step index to produce a detailed scene prompt δ_t. Image2 then generates the next screenshot:

$$P \sim \Pi_\psi(\cdot \mid \tau, w, \tilde{s}_0), \tag{6}$$

$$\delta_t \sim V(\cdot \mid P, \tilde{s}_t, t), \tag{7}$$

$$\tilde{s}_{t+1} \sim M_\phi^V(\cdot \mid \tilde{s}_t, \delta_t). \tag{8}$$

The prompt δ_t describes the next scene in the context of the current screenshot. It is an auxiliary generation signal rather than a component of the underlying software state. The generated screenshot is returned to Voyager for the next step. The plan and scene prompts are excluded from policy inputs; policy training uses the task and observation–action history. Here, K denotes the observation transition induced by real software, while P denotes the synthetic trajectory plan. Interaction fidelity concerns how faithfully the generated transitions reflect the corresponding software behavior.

A.2 SEED CONTEXT AND TRAINING RECORDS

The seed record preserves the sampled state, visible elements, target surface, and visual style alongside the screenshot. Task generation and planning reuse this context. To adapt a supplied task, the sampler can fix the required foreground application and add related background windows for workflows that span applications. These details complement the task-generation procedure in Section 3.

Spatial annotations are drawn on a copy of the pre-action screenshot; the clean image remains the policy input and the reference for subsequent image transitions. Typing, scrolling, hotkeys, and waiting require no target box. Each trajectory yields one supervised example per target step; Appendix E.1 specifies the message format and training loss.

B Action Space

We use a unified `computer_use` interface across Windows, macOS, Ubuntu, and Chrome browser trajectories. Each action a_t is serialized as one or more ordered tool calls. Each call contains a `name` field set to `computer_use` and an `arguments` object whose `action` field selects one of the 14 primitives in Table 6. Multiple calls within a step are ordered by execution.

Action	Definition	Arguments
`key`	Presses keys in order and releases them in reverse order.	`keys`
`key_down`	Holds the specified keys until release.	`keys`
`key_up`	Releases the specified keys in reverse order.	`keys`
`type`	Enters the specified text.	`text`
`mouse_move`	Moves the pointer to the target location.	$C(x, y)$
`left_click`	Left-clicks at the target location.	$C(x, y)$
`left_click_drag`	Presses at the start point, drags to the end point, and releases.	$C_1(x, y)$, $C_2(x, y)$
`right_click`	Right-clicks at the target location.	$C(x, y)$
`middle_click`	Middle-clicks at the target location.	$C(x, y)$
`double_click`	Left-clicks twice at the target location.	$C(x, y)$
`triple_click`	Left-clicks three times at the target location.	$C(x, y)$
`scroll`	Scrolls the mouse wheel by the specified amount.	`pixels`
`wait`	Waits for the specified duration in seconds.	`time`
`terminate`	Ends the task with a completion status.	`status`

Table 6: Unified action space for desktop and Chrome browser trajectories. $C(x, y)$ denotes a screen-coordinate pair; C_1 and C_2 denote the drag start and end points, respectively. The `keys` argument is an array, `text` is a string, `pixels` is a signed scroll amount, and `time` is a duration in seconds. The `status` argument is either `success` or `failure`.

Coordinate arguments. The notation $C(x, y)$ abbreviates the `coordinate` $= [x, y]$ argument, with both values on a resolution-independent 0–999 integer grid. For dragging, C_1 corresponds to `start_coordinate` and C_2 to `coordinate`; each contains its own (x, y) pair.

C Supplementary Data Analysis

This appendix gives the full figures, numerical results, and measurement protocols summarized in Sections 4.3 and 4.4. The analysis characterizes screenshots and trajectories; downstream training results are reported in Section 5.

C.1 Sources and Measurement Protocol

We compare gpt-image-2 screenshots with screenshots collected from real GUI systems within each OS domain. Ubuntu pairs AutoGUI with ScaleCUA, aria_ui, and AgentNet; Windows uses ScaleCUA and AgentNet. Web pairs Image2GUI with ScaleCUA, WebSTAR, and Mind2Web; macOS uses ScaleCUA, AgentNet, and OS-Atlas. This is the corpus pool, and each metric uses the sources for which its measurements are available. In particular, the Qwen Windows and macOS cross-source references each contain one real–real pair. ScaleCUA is the designated reference because its native full-screen resolution is closest to the corresponding synthetic domain.

Splits preserve the interaction groups available in the source extracts: AutoGUI trajectory folders, ScaleCUA session identifiers, and WebSTAR task or site groups. AgentNet and aria extracts are treated as single-frame groups, since compatible sequence identifiers are unavailable in these extracts. Grouped sources keep adjacent frames together; single-frame sources cannot provide the same sequence-level separation guarantee.

We mean-pool the 4096-dimensional Qwen3.5-9B PatchMerger outputs after merge factor 2. DINOv2 and pixel statistics provide independent diagnostics. All pairs within a domain share joint

Table 7: Experimental configuration. Qwen features follow the visual path used to construct the model input. Auxiliary encoders and pixel statistics test whether the result depends on that representation.

Component	Configuration
Qwen representation	Qwen3.5-9B PatchMerger output; merge factor 2; 4096-dimensional merged tokens; image-level mean pooling
Image preprocessing	`smart_resize` with `max_pixels`$= 2048 \times 32 \times 32$, matching the data pipeline
MMD	Per-domain joint z-score; RBF kernel; one bandwidth selected from the median heuristic over the union of sources in that domain
C2ST	GroupKFold logistic-regression probe; AUC 0.5 denotes chance-level discrimination
Sampling	About 1,500 images per source for Qwen embeddings; 500 for DINOv2; 80 for OCR; 200–500 for element detection
Pixel diagnostics	300 images/source $\times$ 20 random 64^2 patches; 250 images/source $\times$ 12 artifact-probe 128^2 patches
Uncertainty	Group bootstrap with 400–1,000 resamples; joint resampling for distance ratios; 95% confidence intervals
UI detection	OmniParser-v2.0 `icon_detect`; confidence $\{0.05, 0.15, 0.25\}$; `imgsz`$= 1280$; `max_det`$= 500$; union coverage rasterized on a 256×256 grid

z-score normalization and one RBF bandwidth selected by the median heuristic over the source union. Within-source group splits estimate a noise floor; distances between real datasets provide an empirical cross-source reference. The principal ratio is

$$\rho = \frac{D(\text{synthetic}, \text{ScaleCUA})}{\text{median}_{i<j}\, D(\text{real}_i, \text{real}_j)}.$$

Group bootstrap uses 400–1,000 resamples, with joint resampling for ratios. We report 95% intervals without choosing a post-hoc acceptance threshold. A classifier two-sample test (C2ST) uses a logistic-regression probe and GroupKFold; AUC 0.5 denotes chance-level discrimination. High C2ST AUC can coexist with a modest relative MMD because the statistics measure different aspects of distributional separation.

C.2 Token Counts and Resolution

Token counts are measured after Qwen's `smart_resize` preprocessing. They depend directly on resolution and aspect ratio, and are analyzed separately from embedding distances. Table 8 retains the source report's available token-count summaries. AutoGUI desktop screenshots use a fixed 1792×1024 generation resolution and yield 1,792 tokens. Image2GUI Web screenshots use multiple resolutions. Most real sources also have only a few token-count values, reflecting their capture settings.

Table 8: Visual token counts after preprocessing. Intervals accompany means; percentiles are P10/P50/P90/P95.

Domain / source	Unique values	Mean [95% CI]	Percentiles
Ubuntu synthetic	1	1792 [1792,1792]	1792/1792/1792/1792
Ubuntu ScaleCUA	2	1445 [1174,2040]	880/880/2040/2040
Ubuntu aria	1	2040 [2040,2040]	2040/2040/2040/2040
Windows synthetic	1	1792 [1792,1792]	1792/1792/1792/1792
Windows ScaleCUA	1	2040 [2040,2040]	2040/2040/2040/2040
Windows AgentNet	13	919 [910,929]	880/880/912/912
Web synthetic	2	1879 [1857,1899]	1296/1980/1980/1980
Web ScaleCUA	4	1918 [1754,2002]	1995/1995/2013/2013
Web WebSTAR	1	576 [576,576]	576/576/576/576
macOS synthetic	1	1792 [1792,1792]	1792/1792/1792/1792
macOS ScaleCUA	1	1995 [1995,1995]	1995/1995/1995/1995

Token-count KS statistics can reach 1.000 and Cliff's delta can reach ± 1 because of nonoverlapping capture resolutions. These statistics alone do not measure semantic or interface diversity. Token-

count matching is infeasible for Ubuntu, Windows, and macOS because the synthetic and ScaleCUA samples have no overlapping token counts. Web has only one shared token count, with too few matched samples to estimate a matched embedding distance. Real-source token buckets themselves have MMD 0.221 on Ubuntu (880 versus 2040 tokens) and 0.350 on Web (1995 versus 2013 tokens). These associations do not establish a causal effect of resolution or layout.

The conditional-analysis export lists synthetic–ScaleCUA MMD values of 0.162/0.257/0.162/0.210 for Ubuntu/Windows/Web/macOS. These differ slightly from the primary estimates in Table 9. They are retained as a separate exploratory export and are not substituted into the primary figure or combined with its confidence intervals.

C.3 Visual Representation Distributions

Qwen embedding distances. Figure 6 preserves the complete Qwen distribution figure. The matched synthetic–ScaleCUA ratios range from 0.59 to 1.24. Windows and macOS lie below their current real-source references; Web and Ubuntu are 14% and 24% above their real-source medians. Table 9 gives the raw distances and intervals.

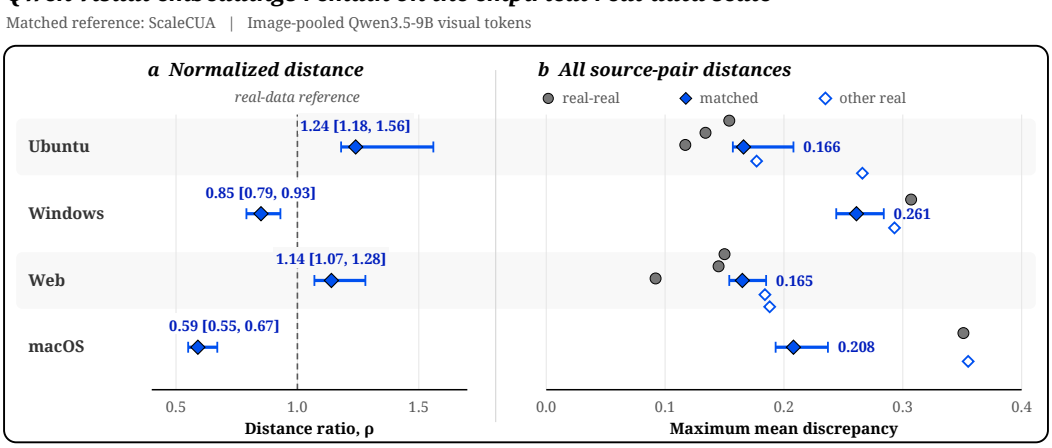

Figure 6: Distribution alignment in the Qwen3.5-9B visual-token space. Panel a reports the matched synthetic–ScaleCUA MMD normalized by the median real–real MMD; horizontal bars give group-bootstrap 95% confidence intervals and $\rho = 1$ marks the empirical real-data scale. Panel b shows every available real–real distance, the matched synthetic–ScaleCUA estimate with its interval, and synthetic distances to the remaining real sources.

Table 9: MMD in the Qwen3.5-9B visual-token space. Real–real entries list all available pairwise distances in ascending order. "Other real" lists distances from the synthetic source to non-ScaleCUA real sources. Brackets give group-bootstrap 95% confidence intervals.

Domain	Real–real MMD	Synthetic–ScaleCUA	Synthetic–other real	ρ
Ubuntu	.117/.134/.154	.166 [.157,.208]	.177/.266	1.24 [1.18,1.56]
Windows	.307	.261 [.244,.284]	.293	.85 [.79,.93]
Web	.092/.145/.150	.165 [.154,.185]	.184/.188	1.14 [1.07,1.28]
macOS	.351	.208 [.193,.237]	.355	.59 [.55,.67]

Reference sensitivity. ScaleCUA group-split floors are 0.109 on Ubuntu, 0.003 on Windows, 0.007 on Web, and 0.004 on macOS. Ubuntu has substantial within-source session variation. Secondary-real comparisons also expose domain-specific deviations: synthetic–aria MMD reaches 0.266 on Ubuntu ($\rho = 1.99$), and synthetic–AgentNet reaches 0.355 on macOS ($\rho = 1.01$) under a resolution mismatch. Windows and macOS each have one real-source pair in this analysis; their denominators are current references, not stable estimates across many independent datasets.

Source fingerprints. Synthetic–real C2ST AUC is approximately 1.00, while real–real AUC is 0.95–1.00. A leave-one-real-source test trains on synthetic versus one real source and evaluates against another real source. Its mean AUC is 0.996 on Ubuntu and 0.976 on Web, compared with real-source controls of 0.960 and 0.925. The excesses, 0.036 and 0.051, have no reported bootstrap intervals and remain descriptive. Windows and macOS reach 0.990 and 0.912, but lack leave-one-real controls because only two real sources enter the test. Synthetic screenshots have a detectable source fingerprint, alongside strong fingerprints among real datasets.

Independent representation. DINOv2 CLS features test whether the relative distance pattern depends on Qwen. Figure 7 and Table 10 report the exploratory 500-image-per-source analysis. Windows lies below the real reference, Web is at the same scale, and macOS falls inside the real-source range. Ubuntu is the exception: its synthetic–real range lies wholly above the observed real–real range. The Qwen result is therefore representation- and domain-dependent.

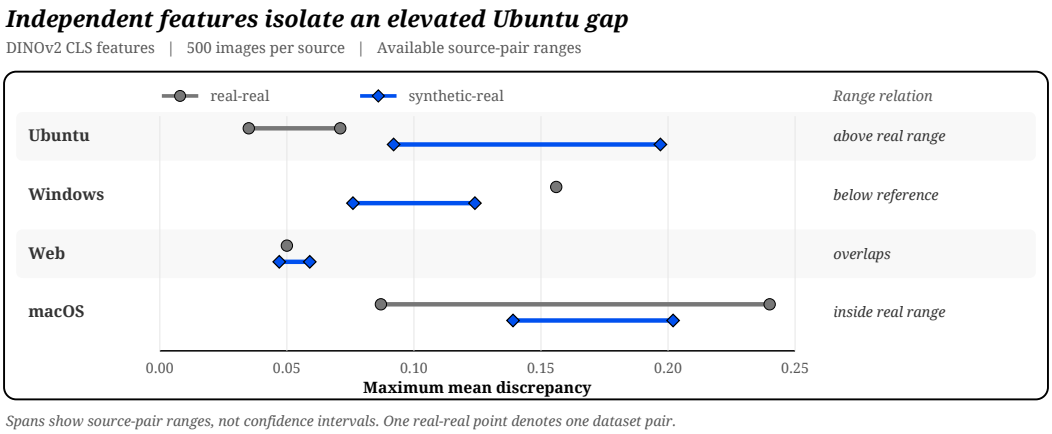

Figure 7: Exploratory DINOv2 MMD using 500 images per source. Gray and blue spans collect the available real–real and synthetic–real source-pair distances within each domain; single available comparisons appear as points. The spans are not confidence intervals. Ubuntu is the only domain whose synthetic–real range lies wholly above the observed real–real range.

Table 10: Exploratory DINOv2 MMD using 500 images per source. Ranges collect the available source-pair distances within each domain.

Domain	Real–real MMD	Synthetic–real MMD	Relation
Ubuntu	.035–.071	.092–.197	1.3–5.6× real–real
Web	.050	.047–.059	Same scale
Windows	.156	.076–.124	Below real–real
macOS	.087–.240	.139–.202	Inside real–real range

C.4 Low-Level Appearance and Generation Residuals

Random patches probe high-frequency decay, periodic texture, edge spread, and text degradation. Figure 8a and Table 11 show departures in different directions: Ubuntu synthetic patches have less high-frequency power than the real range, while Web patches have more. Peak kurtosis is higher on Ubuntu and Web, but lower on Windows and macOS. The observed statistics do not show a common high-frequency-decay pattern across all domains.

Random patches mix content composition and source artifacts. We therefore separate flat backgrounds, text, icon edges, and photos, then match edge density, brightness, and color entropy within each ROI class. Figure 8b reports ROI results for backgrounds, text, and icons on the resolution-matched Ubuntu and Windows subsets. Text/icon C2ST AUC ranges from 0.44 to 0.57. These point estimates are near chance; no confidence intervals or equivalence tests are available. Ubuntu back-

grounds retain an AUC of 0.65, the clearest local residual. Web and macOS do not support the same ROI interpretation because their real-source Retina captures are resolution-mismatched.

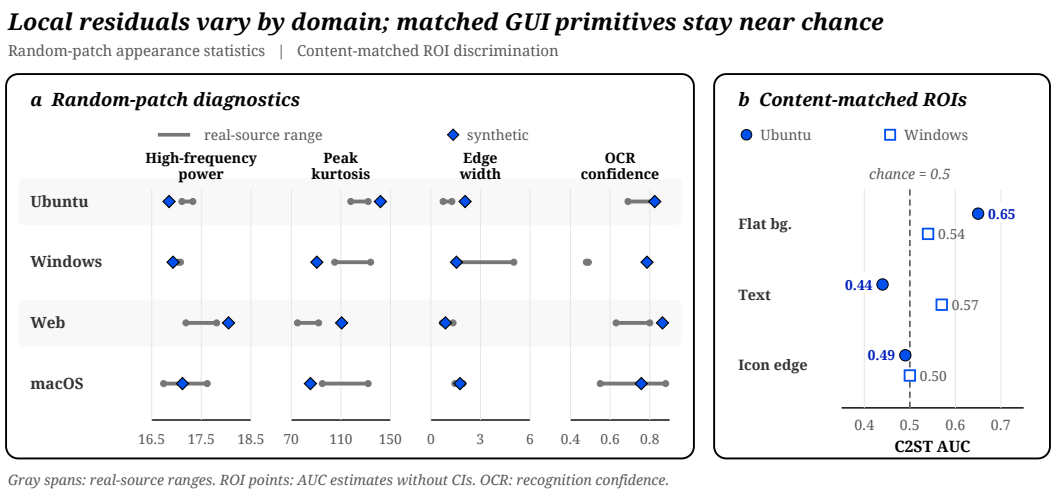

Figure 8: Low-level appearance and generation-residual diagnostics. Panel a places each synthetic random-patch statistic against the range of real sources in the same domain; the gray spans are source ranges, not confidence intervals, and OCR denotes recognition confidence. Panel b reports content-matched ROI C2ST AUC without confidence intervals; 0.5 is chance. Text and icon patches remain near chance, while Ubuntu flat backgrounds retain the clearest residual.

Table 11: Random-patch diagnostics, reported as synthetic / real-source range. OCR is recognition confidence, not a measure of text correctness.

Domain	High-frequency power	Peak kurtosis	Edge width	OCR confidence
Ubuntu	16.85 / 17.11–17.33	141.9 / 118–132	2.07 / .74–1.27	.825 / .69–.83
Web	18.05 / 17.19–17.81	110.6 / 75–92	.88 / .70–1.34	.864 / .63–.80
Windows	16.93 / 17.04–17.08	90.6 / 105–134	1.55 / 1.43–5.02	.786 / .48–.49
macOS	17.12 / 16.74–17.62	85.4 / 95–132	1.76 / 1.45–1.94	.757 / .55–.88

Table 12: Content-matched patch C2ST AUC. Values near 0.5 indicate chance-level separation.

Domain	Flat background	Text	Icon edge
Ubuntu	.65	.44	.49
Windows	.54	.57	.50

OCR confidence in these diagnostics measures recognizer confidence, not transcription accuracy or semantic correctness. Full-screen OCR examples in Appendix C.5 illustrate outputs of a separate measurement and are not examples from the content-matched ROI experiment.

C.5 Interface Coverage, Detection, and OCR

OmniParser-v2.0 detects candidate UI regions. We rasterize the union of its boxes on a 256×256 grid, avoiding repeated area from overlapping boxes. Figure 9 retains both coverage and element counts across confidence thresholds 0.05, 0.15, and 0.25. Coverage is higher for synthetic screenshots in all twelve comparisons against ScaleCUA. Relative gains are 63–77% on Ubuntu, 51–75% on Windows, 49–71% on Web, and 10–13% on macOS. The compact main figure computes gains as $100(C_s/C_r - 1)$ from the published coverage values; small rounding differences follow from the three-decimal source precision.

Element count is not uniformly higher. At threshold 0.15, Windows has 142 synthetic elements versus 154 in ScaleCUA, while union coverage rises from 0.212 to 0.351. The result describes screen

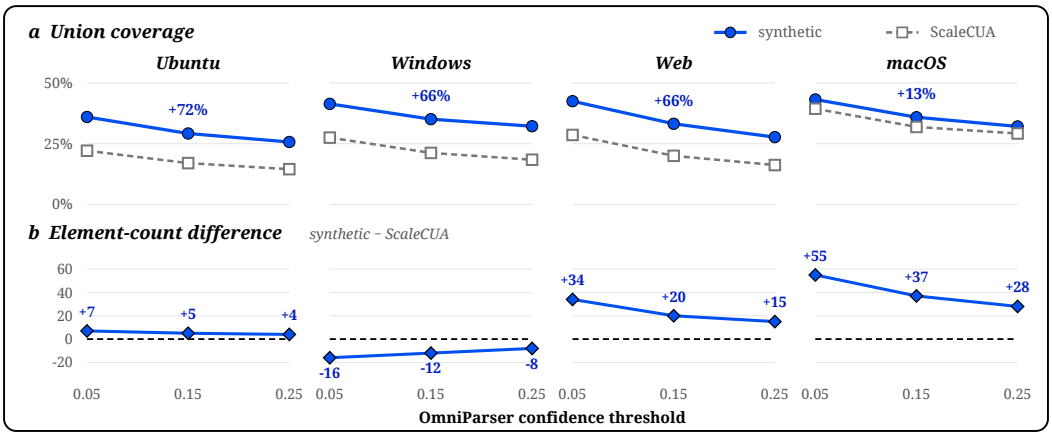

Figure 9: Static interface complexity across OmniParser confidence thresholds. Panel a compares the rasterized union coverage of synthetic and ScaleCUA screenshots; the annotated percentage is the relative gain at the middle threshold, 0.15. Panel b reports the element-count difference, synthetic minus ScaleCUA. Coverage is higher in all twelve comparisons, including Windows where the detector returns fewer synthetic elements.

Table 13: Static complexity at confidence thresholds 0.05/0.15/0.25. Each cell lists values in that order. Coverage is the rasterized union of detected boxes.

	Union coverage		Element count	
Domain	Synthetic	ScaleCUA	Synthetic	ScaleCUA
Ubuntu	.360/.292/.257	.221/.170/.145	137/103/90	130/98/86
Windows	.414/.351/.322	.275/.212/.184	184/142/126	200/154/134
Web	.425/.332/.277	.286/.200/.162	103/68/56	69/48/41
macOS	.432/.359/.321	.394/.319/.292	165/122/105	110/85/77

area occupied by detected regions, rather than a universal ranking of element count or task difficulty. A manual audit of 180 detection overlays found no systematic tendency to label generated texture as UI elements. This qualitative check does not supply a calibrated detector error rate. Coverage was compared with the matched ScaleCUA reference, not every real corpus.

Detection and OCR examples. Figure 10 shows selected Windows screenshots with full-screen detection and OCR overlays. They are not matched by task or application, and do not establish a synthetic–real ranking. These overlays illustrate parser outputs separately from the aggregate experiment: the export's `max_det` setting is unverified, and detected regions may include non-control image content.

C.6 Trajectory Structure and Interpretation

The analyzed synthetic trajectory cohort contains 2,168 Ubuntu, 1,394 Windows, and 591 macOS trajectories. Its denominator differs from the filtered training sample count and the transition audit. Ubuntu and Windows have the longest sequences; macOS combines more open applications with higher action-type entropy. The source extracts do not supply compatible real-trajectory metadata for a matched comparison.

Precondition and blocker fields are zero throughout this cohort, and rendered terminal states are almost always labeled successful. The non-success fractions are 0%, 0.29%, and 0.68% for Ubuntu, Windows, and macOS. These are annotation statistics, not executable task scores. Observed varia-

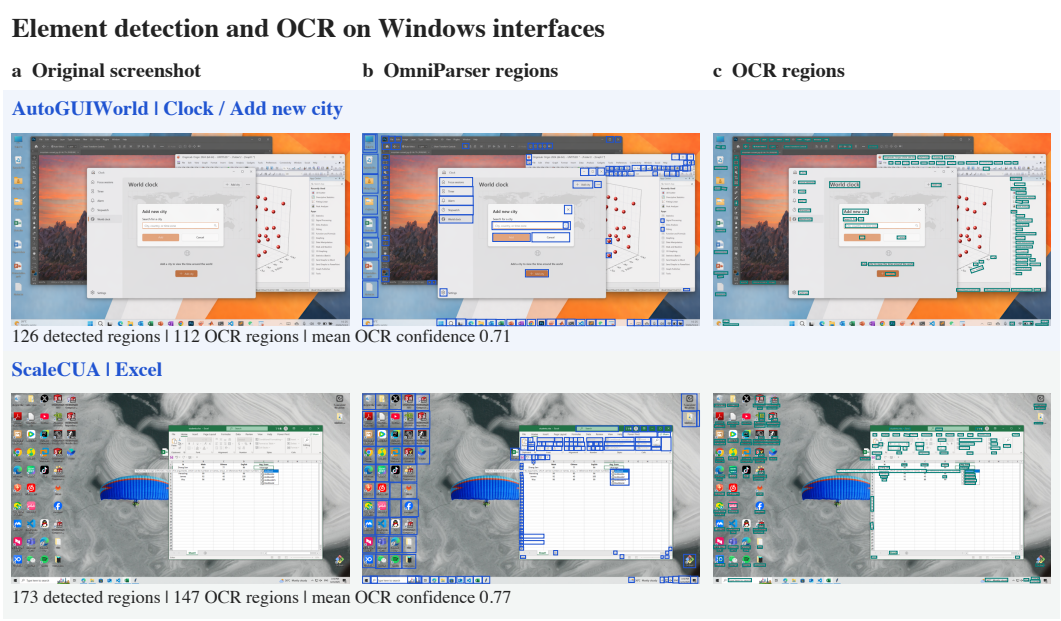

Figure 10: Element detection and OCR examples on Windows interfaces. Each row shows the same full-screen view without annotations, with blue OmniParser boxes at confidence threshold 0.15, and with teal OCR regions. Row summaries report full-image counts and mean OCR confidence. OCR confidence is not transcription accuracy. These selected examples illustrate parser behavior, not a paired quality comparison.

Table 14: Synthetic trajectory structure. Step percentiles are P10/P50/P90; application percentiles are P50/P90. Failure is the fraction of rendered terminal states not marked as success.

Domain	N	Steps	Action types	Entropy	Open apps	Failure
Ubuntu	2168	11.9 (4/10/24)	4.33	.79	1.62 (2/2)	0%
Windows	1394	11.4 (2/8/25)	4.05	.82	2.06 (2/3)	.29%
macOS	591	7.1 (2/6/14)	3.95	.92	2.50 (2/3)	.68%

tion lies in step count, action mix, and application count; it does not establish greater difficulty than real trajectories.

Interpretation across measurements. The results jointly characterize relative visual distance, local appearance, and screen composition. They do not establish zero distribution shift: source classifiers remain effective, Ubuntu and Web have $\rho > 1$, and DINOv2 exposes a larger Ubuntu gap. Local ROI scores do not prove full-screen indistinguishability. Greater union coverage is specific to the ScaleCUA comparison and does not imply more elements in every domain. Transition consistency is measured separately in Appendix D; real task execution and grounding are evaluated in Section 5.

D Transition Fidelity and Data Filtering

Static screenshot quality does not determine whether a generated next screen faithfully realizes the action that precedes it. We therefore audit each desktop transition using the clean pre-action screenshot, the serialized action and its parameters, an annotated target frame for pointing actions, and the Image2-generated post-action screenshot. The original QC schema also evaluates trajectory reasoning; the analysis here uses only the five image–action dimensions in Table 15.

D.1 VLM Quality-Control Protocol

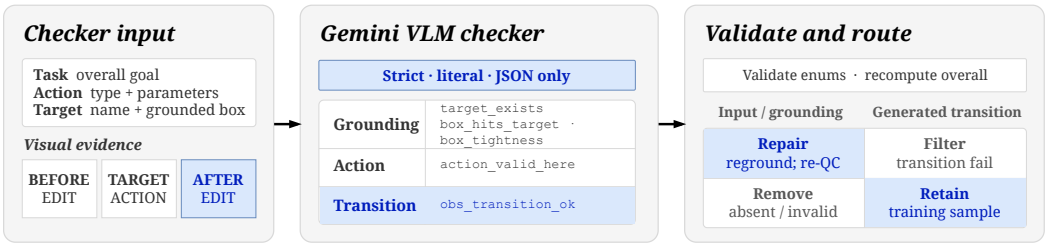

Figure 11: Step-level quality control with a Gemini VLM checker. Each request combines the task and exact action context with an ordered set of visual evidence. Gemini evaluates grounding, action validity, and visual-transition fidelity, then returns a strict structured verdict. Code validates the per-dimension values, recomputes the aggregate label, and routes the step to target-box repair, removal, transition filtering, or retention. Repaired target boxes are evaluated again before retention.

Checker input and prompt. For every trajectory step, the QC runner constructs one multimodal request. The textual context contains the overall task, the action type, all non-empty action parameters, the target element named by the planner, and the element name associated with each grounded box. The original request also includes the step reasoning when available, although reasoning judgments are excluded from the analysis reported here. The visual context contains up to three images in an explicitly stated order: BEFORE, the clean pre-action observation; TARGET, the same observation with a red target box for a pointing action; and AFTER, the distinct post-action observation generated by Image2. The system instruction identifies Gemini as a meticulous GUI trajectory inspector, directs it to be "strict and literal about what the images actually show," and requires JSON without Markdown or explanatory text. The user prompt defines every field, states when a field is not applicable, and asks for one short reason naming the most serious problem.

Exact Checker Prompt

For reproducibility, the two blocks below reproduce the system message and user-message template verbatim from the QC runner. Braced terms are populated from each trajectory step at runtime. The image payload follows the order given by `{image_legend}`. The original prompt includes two reasoning checks; as noted above, those fields are excluded from the image–action analysis in this paper.

System message.

```
You are a meticulous GUI trajectory quality inspector. You are shown a single step
     of a GUI agent trajectory and must judge whether the red box targets the
    right element, whether the action fits the screen, and whether the reasoning
    matches what is visible. Be strict and literal about what the images actually
    show. Return strict JSON only, no Markdown, no extra prose.
```

User message template.

```
You are quality-checking ONE step of a GUI agent trajectory.

Overall task the agent is doing:
{task}

This step:
- action type: {action_name}
- action parameters: {action_params}
- planner target element: {target_element}
- grounded box element name(s): {box_elements}
- agent thinking for this step: {thinking}

Images provided (in order):
{image_legend}
```

```
Judge each dimension below from what the images ACTUALLY show. Use "na" only when
    the dimension does not apply (e.g. grounding checks for a non-pointing action,
     or transition check when no AFTER image).

GROUNDING (only for pointing actions that have a red box):
- box_hits_target: does the red box enclose the UI element the action intends to
    operate on (the planner target)? fail if it boxes a different element or empty
     space.
- box_tightness: is the box a reasonable tight fit around the interactive element?
     warn if far too large (whole panel/row) or too small (only part of the glyph)
    . "na" if no box.
- target_exists: is the planner's target element actually visible on the BEFORE
    screen? fail if that element is NOT present on screen (so any box is
    necessarily wrong).

ACTION vs SCREEN (all steps):
- action_valid_here: given the BEFORE screen state, is this action sensible and
    executable now? fail if it makes no sense (e.g. type_text with no focused
    input, click a control that isn't there).
- obs_transition_ok: comparing BEFORE and AFTER, did the screen change in a way
    consistent with this action? warn if the change looks unrelated; "na" if no
    AFTER image or a non-visual action (wait/answer).

THINKING (only if thinking text is provided):
- thinking_matches_screen: does the thinking describe elements/state that are
    actually visible? fail if it hallucinates UI not present.
- thinking_wavering: does the thinking contradict itself or express confusion
    about what to do (e.g. "but wait", "actually", second-guessing the target)?
    fail if it wavers, pass if it is a clean single decision.

Then give:
- overall: "ok" (all good), "minor" (only warnings or a thinking wobble), "major"
    (any of box_hits_target/target_exists/action_valid_here is fail).
- confidence: your confidence in this judgment (high/medium/low) given image
    clarity.
- reason: ONE short sentence naming the single most serious problem, or "clean" if
     none.

Output strict JSON only, exactly this shape:
{
  "grounding": {"box_hits_target": "pass|warn|fail|na", "box_tightness": "pass|
    warn|fail|na", "target_exists": "pass|fail|na"},
  "action": {"action_valid_here": "pass|warn|fail", "obs_transition_ok": "pass|
    warn|fail|na"},
  "thinking": {"thinking_matches_screen": "pass|warn|fail|na", "thinking_wavering
    ": "pass|fail|na"},
  "overall": "ok|minor|major",
  "confidence": "high|medium|low",
  "reason": "<one sentence>"
}
```

Inspection layers. The checker first verifies whether the intended target exists and whether the grounded box hits it with a reasonable extent. It then asks whether the exact serialized action is sensible and executable in the pre-action state. Finally, it compares BEFORE and AFTER to determine whether the visual change realizes the action and its payload. This last check is deliberately literal: a plausible task-level outcome is still inconsistent when it changes the wrong text, invents unsupported content, skips intermediate actions, or fails to produce the requested local effect. The five dimensions and their operational meanings are listed below.

Structured verdict and deterministic normalization. Gemini returns `pass`, `warn`, `fail`, or `na` for each applicable dimension, together with an `overall` label, confidence, and a one-sentence reason. The runner does not trust the model-produced aggregate label directly. It first discards values outside the allowed enums and then recomputes `overall`: a failure in `target_exists`, `box_hits_target`, or `action_valid_here` is `major`; any remaining failure or warning is `minor`; otherwise the step is `ok`. Filtering still uses the individual dimensions rather than `overall`

Table 15: Image–action dimensions used by the transition-quality audit.

Group	Dimension	Criterion
Grounding	`target_exists`	The intended target is visible in the pre-action screenshot.
Grounding	`box_hits_target`	The annotated box encloses the intended target rather than another element or empty space.
Grounding	`box_tightness`	The target box has a reasonable spatial extent around the interactive element.
Action	`action_valid_here`	The action is executable and meaningful in the current GUI state.
Transition	`obs_transition_ok`	The generated post-action screenshot shows a visual consequence consistent with the action.

alone. Thus, an `obs_transition_ok` failure is removed after the input action and grounding are verified, even though that field by itself maps to the `minor` aggregate category.

Repair and filtering. The routing separates a defective input specification from an Image2 transition error. We reground 3,390 target boxes from major steps for which the target remains visible, then run the checker again. Steps with an absent target, an invalid action, or unrepairable grounding are removed. Among steps with a valid input action and target, any failure of `obs_transition_ok` is filtered as an action–image inconsistency; the remaining steps are eligible for the training set.

Table 2 in Section 4.2 summarizes the sample flow through these stages. Figure 12 retains the platform-level flow visualization.

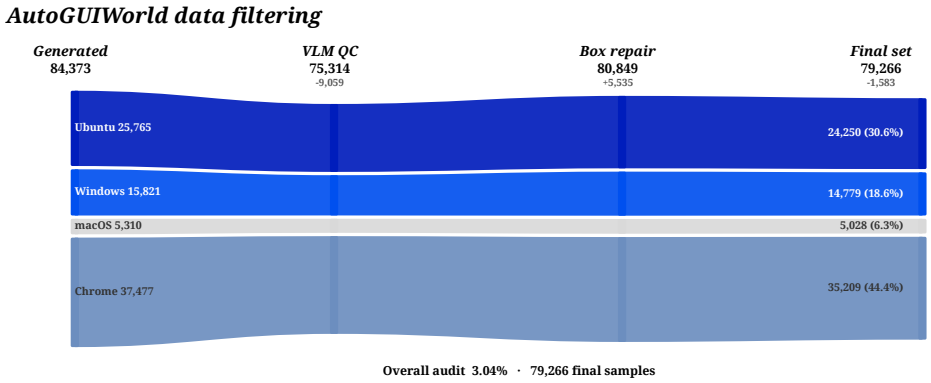

Figure 12: AutoGUIWorld training-data filtering. The proportional ribbons track Ubuntu, Windows, macOS, and Chrome samples through VLM quality control, target-box repair, and transition filtering. Desktop counts are measured directly. Chrome contributes 35,209 retained samples; its preceding stage counts are inferred using the desktop-stage retention ratios. The resulting training set contains 79,266 samples.

D.2 Transition Audit Results

The audit identifies 1,293 failures among 42,526 evaluable desktop transitions, giving an overall action–image inconsistency rate of 3.04%. To isolate failures in the generated next observation, we additionally retain only active actions for which the action-validity check passes and the target existence and grounding checks either pass or do not apply. Under this filtered view, 818 of 39,351 transitions remain inconsistent (2.08%).

The filtered rate is action-dependent: 7.58% for drag, 7.09% for scroll, 6.35% for text entry, 3.23% for hotkeys, 2.16% for key presses, and 0.67% for clicks. Text-entry failures are especially common in Ubuntu terminal workflows, whereas Windows contributes prominent drag, hotkey, and document-editing failures. Click errors fall sharply after target and action checks, while text-entry

Table 16: Transition-fidelity audit by operating system. Overall rates use all evaluable transitions. Filtered rates remove steps with an invalid action, absent target, or incorrect target grounding.

OS	Evaluable	Fail	Rate	Filtered	Fail	Rate
Ubuntu	23,625	731	3.09%	21,762	516	2.37%
Windows	14,429	479	3.32%	13,541	286	2.11%
macOS	4,472	83	1.86%	4,048	16	0.40%
All	42,526	1,293	3.04%	39,351	818	2.08%

errors remain nearly unchanged, indicating that exact content preservation is a distinct challenge for visual transition synthesis.

D.3 Representative Failure Modes

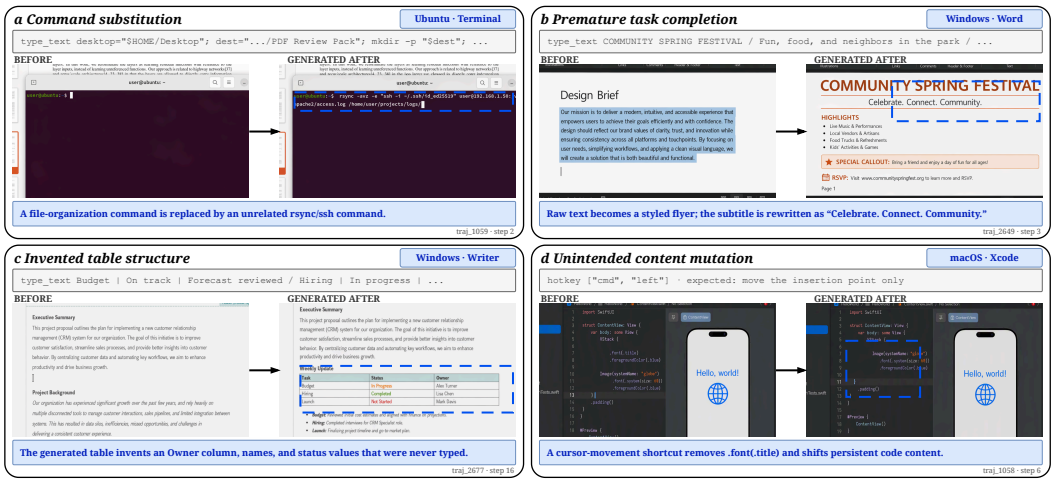

Figure 13: Representative Image2 transition failures. Each case shows the same 16:9 crop before and after an atomic action, with the inconsistent region highlighted in blue. The examples show terminal-command substitution, premature rendering of a styled artifact, invented table structure and values, and persistent code mutation following a cursor-movement shortcut.

Exact-content substitution. Image2 can replace a specified terminal command, script, document passage, name, or numerical value with different but visually plausible content. This creates a direct conflict between the action payload and the resulting observation.

Semantic over-completion. A text-entry action can directly produce the task's final visual artifact, such as a styled flyer or formatted table. The screen is plausible at the task level, but the generated transition skips later formatting actions and breaks the intended atomic action sequence.

Missing or incorrect action effects. Drag and scroll actions can leave the relevant region unchanged or produce an unrelated interface change. Similar errors occur when a click does not toggle the intended control or a key press produces an unrelated application state.

Persistent-content mutation. Low-impact keyboard actions can unexpectedly alter persistent content. In the Xcode example, a cursor-movement shortcut removes and rearranges code rather than changing only the insertion-point position.

E GROUNDING EVALUATION DETAILS

E.1 TRAINING AND INFERENCE HARNESS

MESSAGE CONSTRUCTION

The documented message protocol assembles the task, screenshots, and interaction history using the following format. Each ms-swift multimodal JSONL record contains `messages` and `images` fields and supervises one target step. The system message defines the `computer_use` signature within `<tools>` tags and specifies the response format. Each user turn contains an `<image>` marker associated with its screenshot. The first retained user turn also includes the task instruction and, when available, a `Previous steps` section for older actions.

Response serialization. An assistant response begins with `# Step N:`, followed by an `Action:` description and one or more `<tool_call>` blocks. The action description is a single imperative sentence naming the interface element, the operation, and its immediate purpose. Each tool-call block contains a JSON object using the action contract in Sec. B. An example response is shown below.

```
# Step 1:
Action: Click the search icon to open the search field.
<tool_call>
{"name":"computer_use","arguments":
 {"action":"left_click","coordinate":[500,120]}}
</tool_call>
```

SUPERVISION AND INFERENCE EXECUTION

Training target. Each example pairs the observation before an action with the corresponding assistant response. With `loss_scale=last_round`, only the final assistant response contributes to the training loss; earlier messages serve as context. The supervised output contains the action description and its ordered tool calls.

Inference execution. At inference, the message sequence ends at the current user observation, and the model generates the next assistant response. The parser extracts the tool calls in order, and the execution interface maps their coordinates to the target screen. After execution, the response is added to the history and a new screenshot supplies the next observation. A `terminate` call ends the interaction and reports the agent's `success` or `failure` status.

TRAINING CONFIGURATION

We fine-tune Qwen3.5-35B-A3B with ms-swift Megatron on 32 GPUs across four nodes. The model combines a mixture-of-experts architecture with Gated DeltaNet (GDN). We use tensor parallelism of 2, pipeline parallelism of 1, data parallelism of 16, and expert parallelism of 8 for the 256 experts. Tensor parallelism matches the two key-value heads under the GDN configuration. The global batch size is 512, with a micro-batch size of 1. The maximum sequence length is 16,384 tokens, and overlength examples are removed with `truncation_strategy=delete`.

Training runs for three epochs with a learning rate of 10^{-5}, a warmup fraction of 0.03, a minimum learning rate of 10^{-6}, and weight decay of 0.1. The vision encoder is frozen, and full activation recomputation is enabled.

E.2 SCREENSPOT-PRO PROTOCOL

The base Qwen3.5-35B-A3B model and AGW-35B checkpoints 80, 157, 235, 314, 392, and 469 use the same `qwen3_5_autogui` harness. Inference is served with vLLM and tensor parallelism of two, with `temperature` $= 0$, `top_p` $= 1$, and a server-side image processor limit of 16,777,216 pixels. The evaluation selects `task=all`, `language=en`, `gt_type=positive`, and `inst_style=instruction`. Its 1,581 examples comprise 927 Windows, 604 macOS, and 50 Linux screenshots.

The model receives the screenshot and instruction and produces an action description and tool calls without a separate thinking block. Coordinates on the 0–999 grid are mapped to pixels by multiplying by the screenshot width or height and dividing by 1,000. For drag responses, the parser uses the start point. A point is correct only if it lies inside the annotated box, without additional tolerance. Invalid-format predictions remain in the denominator and receive zero credit. Overall accuracy aggregates individual examples rather than averaging subgroup percentages.

E.3 Checkpoint Progression and Domain Results

Table 17 reports every evaluated checkpoint from one training run. The main result uses the final checkpoint after three epochs; intermediate checkpoints are retained to show the non-monotonic trajectory and are not treated as independent runs or component ablations. The first evaluated checkpoint already reaches 56.4% overall accuracy; the final checkpoint reaches 57.1%. Invalid-format counts vary throughout training, and the smallest count occurs at checkpoint 235 rather than the final checkpoint.

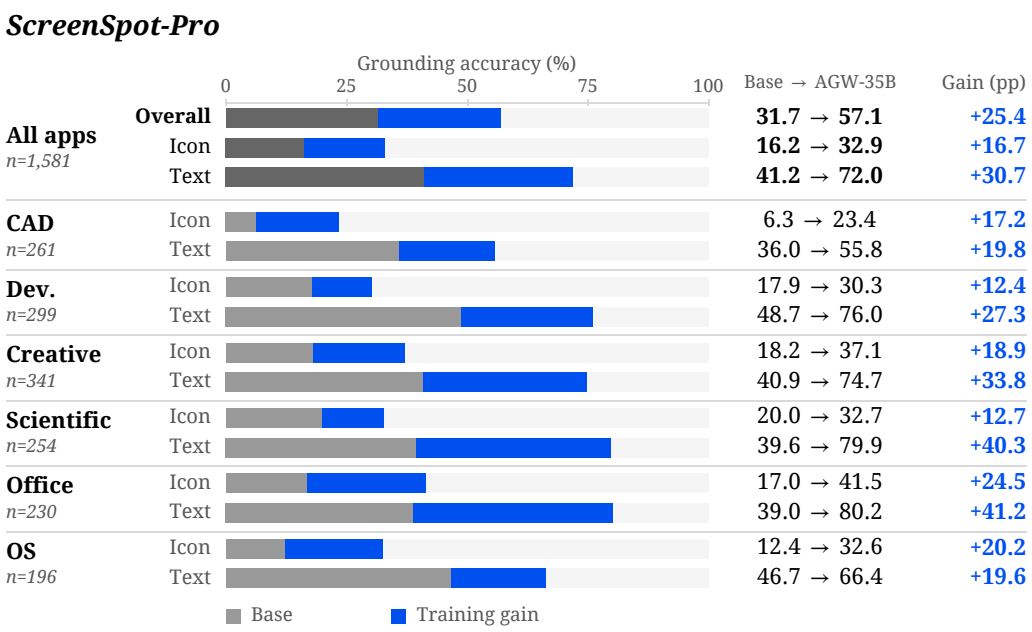

Figure 14: Grounding gains across six ScreenSpot-Pro application domains. Gray bars show base-model accuracy and blue segments show the gain from AutoGUIWorld fine-tuning (checkpoint 469). The top group summarizes all applications. Each n counts all examples in the group; text and icon accuracies use their respective subsets. Gains are computed before rounding.

Table 17: Complete ScreenSpot-Pro checkpoint results. All accuracies are percentages; correct predictions and invalid formats are counts out of 1,581. Text/icon denominators are 977/604, inferred from rounded subgroup accuracies.

Checkpoint	Epoch	Correct	Overall	Text	Icon	Invalid format
Base	0	501	31.7	41.2	16.2	226
80	0.5	892	56.4	69.9	34.6	58
157	1.0	881	55.7	70.9	31.1	48
235	1.5	886	56.0	69.3	34.6	35
314	2.0	877	55.5	69.4	32.9	41
392	2.5	897	56.7	71.0	33.6	52
469	3.0	902	57.1	72.0	32.9	55

Table 18: Overall grounding accuracy (%) by application domain for all evaluated checkpoints. Values reproduce the supplied four-decimal summaries as percentages with two decimal places.

Checkpoint	CAD	Creative	Dev.	OS	Office	Scientific
Base	28.74	31.38	33.78	31.12	33.91	31.10
80	41.76	59.24	55.18	50.00	75.22	57.09
157	44.44	59.24	53.51	45.92	69.57	60.24
235	39.85	60.12	53.18	51.02	72.61	59.45
314	43.68	58.65	51.17	50.51	70.87	58.27
392	45.59	58.94	55.18	50.00	71.74	58.66
469	47.89	58.94	53.85	51.02	71.30	59.45

Table 19: Text grounding accuracy (%) by application domain for all evaluated checkpoints. Values reproduce the supplied four-decimal summaries as percentages with two decimal places.

Checkpoint	CAD	Creative	Dev.	OS	Office	Scientific
Base	36.04	40.91	48.70	46.73	38.98	39.58
80	49.24	74.75	74.68	65.42	82.49	74.31
157	52.79	76.77	74.03	64.49	78.53	79.86
235	46.70	73.23	75.32	63.55	80.23	79.17
314	51.27	73.74	72.08	65.42	77.40	78.47
392	53.81	73.74	76.62	65.42	80.79	77.08
469	55.84	74.75	75.97	66.36	80.23	79.86

Table 20: Icon grounding accuracy (%) by application domain for all evaluated checkpoints. Values reproduce the supplied four-decimal summaries as percentages with two decimal places.

Checkpoint	CAD	Creative	Dev.	OS	Office	Scientific
Base	6.25	18.18	17.93	12.36	16.98	20.00
80	18.75	37.76	34.48	31.46	50.94	34.55
157	18.75	34.97	31.72	23.60	39.62	34.55
235	18.75	41.96	29.66	35.96	47.17	33.64
314	20.31	37.76	28.97	32.58	49.06	31.82
392	20.31	38.46	32.41	31.46	41.51	34.55
469	23.44	37.06	30.34	32.58	41.51	32.73

Data provenance and rounding. The five CSV files in `assets/evaluation/screenspot-pro/` reproduce the source evaluation summaries without modification. The source archive is `screenspot_pro_results_20260924.tar.gz`. The application groups contain 261 CAD, 341 Creative, 299 Dev., 196 OS, 230 Office, and 254 Scientific examples. Integer text/icon partitions recovered from the rounded summaries are unique across all seven models and sum to 977 text and 604 icon examples; these are derived counts rather than separately supplied annotations. The main table and figure use count-derived percentages and differences before rounding, so displayed gains can differ slightly from subtracting two rounded endpoints. The source CSVs, reconstruction checks, and figure builder preserve that distinction. Only aggregate summaries are available here; they do not support a paired significance test or a per-example analysis separating format correction from improved localization.

F Evaluation Infrastructure

F.1 Agent Execution and Task Sets

We evaluate checkpoints 80, 157, 235, 314, 392, and 469 from the AGW-35B training run, which fine-tunes Qwen3.5-35B-A3B on AutoGUIWorld trajectories. Inference uses the message format and `computer_use` interface described in Appendix E.1. Responses contain an action description

and tool calls, without a separate thinking block. Point coordinates use a 0–999 grid and are mapped to screen pixels by multiplying by the screenshot width or height and dividing by 1,000. Drag actions specify both endpoints.

AgentNet comparison. The comparison run fine-tunes Qwen3.5-35B-A3B on approximately 350k examples from AgentNet (Wang et al., 2025a) and is evaluated at steps 40, 80, 160, 320, and 686. AgentNet contains human demonstrations collected on real Windows, macOS, and Ubuntu desktops, augmented with generated reasoning annotations. The 350k count refers to training examples used in our run, rather than the number of human-demonstrated tasks in the original dataset. Its harness enables thinking and represents drag with a single coordinate; AGW-35B uses non-thinking responses and specifies both drag endpoints. Both runs use the same benchmark task sets, evaluators, and fixed-denominator score aggregation. Both runs initialise from Qwen3.5-35B-A3B. Across all four benchmarks, the curves share the existing base-model score at step 0, measured with the AGW harness. ScienceBoard's AgentNet curve connects this shared base to all five evaluated checkpoints. Its step-40 score is 15.4%, with 22 successful tasks out of the fixed 143-task set. The training-step axis preserves each run's checkpoint steps; it does not normalise epochs, tokens, or compute across runs.

Execution environments. The model is served through vLLM with tensor parallelism of two. OSWorld and WAA run in Docker-managed QEMU virtual machines, macOSWorld uses Docker-OSX, and ScienceBoard uses the OSWorld Docker provider. The interactive loop executes the predicted actions in the benchmark environment and supplies a fresh screenshot for the next step. ScienceBoard allows up to 100 model responses per task. The benchmark evaluator determines the final score from the resulting state, independently of the agent's own completion declaration.

Table 21: Evaluation task sets and scoring units. Each checkpoint uses the same task set within a benchmark. Counts describe the evaluated samples, not model results.

Benchmark	Tasks	Evaluation subset	Metric (%)
OSWorld	361	`test_nogdrive`: excludes eight Google Drive tasks	Mean task score
WAA	154	`test_all`	Mean task score
macOSWorld	231	English task and interface, including 29 safety tasks	Task success
ScienceBoard	143	Five application domains, excluding Lean	Task success
ScreenSpot-Pro	1,581	English instructions, positive targets	Grounding accuracy

Grounding inference. ScreenSpot-Pro uses `task=all`, `language=en`, `gt_type=positive`, and `inst_style=instruction`. The `qwen3_5_autogui` harness receives the screenshot and instruction. The base model and all six checkpoints use the same configuration, with greedy decoding (`temperature` $= 0$, `top_p` $= 1$) and a server-side image processor limit of 16,777,216 pixels. Predicted coordinates are evaluated against the annotated target box without an added tolerance. If the response specifies a drag, the parser uses its start point. The sample set contains 927 Windows, 604 macOS, and 50 Linux examples. We retain text/icon, platform, and application group breakdowns for analysis.

F.2 ScienceBoard Task Selection

Our ScienceBoard evaluation uses all 143 tasks in the five non-Lean domains: 31 KAlgebra, 33 Celestia, 29 ChimeraX, 34 GRASS GIS, and 16 TeXstudio tasks. The 26 Lean tasks are excluded, giving $169 - 26 = 143$ evaluated tasks. All six checkpoints use identical application–task identifiers.

The Lean exclusion follows the task-validity concerns documented in ScienceBoard issue #7.[1] The report identifies 13 distinct Lean tasks with false statements, mis-specified objectives, or checks that do not enforce executable solutions. These issues affect both Raw and VM task configurations. Table 22 summarises the reported defects. Our evaluation excludes the Lean domain as a whole, while retaining every task in the other five domains.

[1] ScienceBoard issue #7: `https://github.com/OS-Copilot/ScienceBoard/issues/7`. The audit refers to repository revision `c8d5010bdba3`.

Table 22: Lean task issues reported in ScienceBoard issue #7. The 13 flagged tasks are part of the 26-task Lean domain excluded from our evaluation. Task identifiers are local to Lean.

Reported issue	Task identifiers	Effect on evaluation
False theorem statements	A-02, A-04, B-05, B-06, B-07, C-05, D-02, D-03, D-07	The formal statements admit counterexamples.
Incorrect objective	B-01	The predicate formalises a different coprimality condition.
Vacuous proof	B-02	Contradictory premises permit a proof without the intended argument.
Executability not enforced	E-01, E-02	The evaluators accept noncomputable completions for tasks intended to test executable decision procedures.

F.3 Score Aggregation

For each interactive benchmark, the aggregate is $100\,N^{-1}\sum_{i=1}^{N} s_i$, where N is the task count in Table 21 and $s_i \in [0,1]$ is the task's evaluator score. OSWorld and WAA retain fractional scores from `result.txt`. macOSWorld normalises the binary 0/100 values in `eval_result.txt` to 0/1. ScienceBoard uses the pass/fail value in `result.out`. Missing or nonnumeric task results receive zero credit and remain in the fixed denominator. Application scores use the same rule within each domain.

For the aggregate across interactive benchmarks, we compute $\text{Overall} = \frac{1}{4}\sum_{b=1}^{4} S_b$, where S_b is the percentage score of OSWorld, WAA, macOSWorld, or ScienceBoard. Each benchmark has equal weight regardless of its task count. Within a benchmark, tasks retain equal weight when computing S_b. ScreenSpot-Pro is reported separately. Domain comparisons use the base model and final training checkpoint (469), with gains calculated from unrounded scores. Training curves place the base model at step 0 and plot all six checkpoints at their recorded training steps.

ScreenSpot-Pro accuracy divides the number of correct target points by all 1,581 examples, including responses with an invalid coordinate format. Text, icon, platform, and application scores use their respective sample counts. Overall scores aggregate individual samples rather than averaging the subgroup percentages.

F.4 Interactive Domain Results and Training Progression

Desktop application domains. AGW-35B reaches 40.8% on OSWorld, up from 33.0% for the base model. System tasks gain 25.0 points and GIMP gains 19.2 points. Calc and Impress improve by 8.5 and 12.8 points. Multi-app tasks rise from 12.0% to 20.9% and contribute the largest share of the overall OSWorld gain. On WAA, the score rises from 19.4% to 27.9%. Writer and VLC lead the improvement, gaining 26.3 and 23.8 points. Calc, VS Code, and Edge also improve, while six domains remain unchanged. Chrome declines on OSWorld (39.0% to 26.0%) and WAA (11.2% to 0.0%).

macOS and scientific workflows. Figure 16 shows that macOSWorld success rises from 28.1% to 45.0%. System & Interface improves from 31.0% to 62.1%, Advanced from 13.3% to 40.0%, and System Apps from 44.7% to 65.8%. Multi-app success doubles from 13.8% to 27.6%. Safety decreases from 20.7% to 17.2%. Across macOSWorld and OSWorld, multi-app performance improves but remains below many single-application domains.

ScienceBoard success rises from 14.0% to 32.2%, with gains in all five software domains. KAlgebra improves from 6.5% to 48.4%, and ChimeraX from 37.9% to 55.2%. Together, they account for 18 of the 26 net additional successful tasks, as the total increases from 20 to 46. Celestia, GRASS GIS, and TeXstudio gain 9.1, 11.8, and 6.3 points. Across the four benchmarks, 25 of 35 domains improve, seven remain unchanged, and three decline. Training on AutoGUIWorld trajectories improves real task execution across desktop and scientific applications.

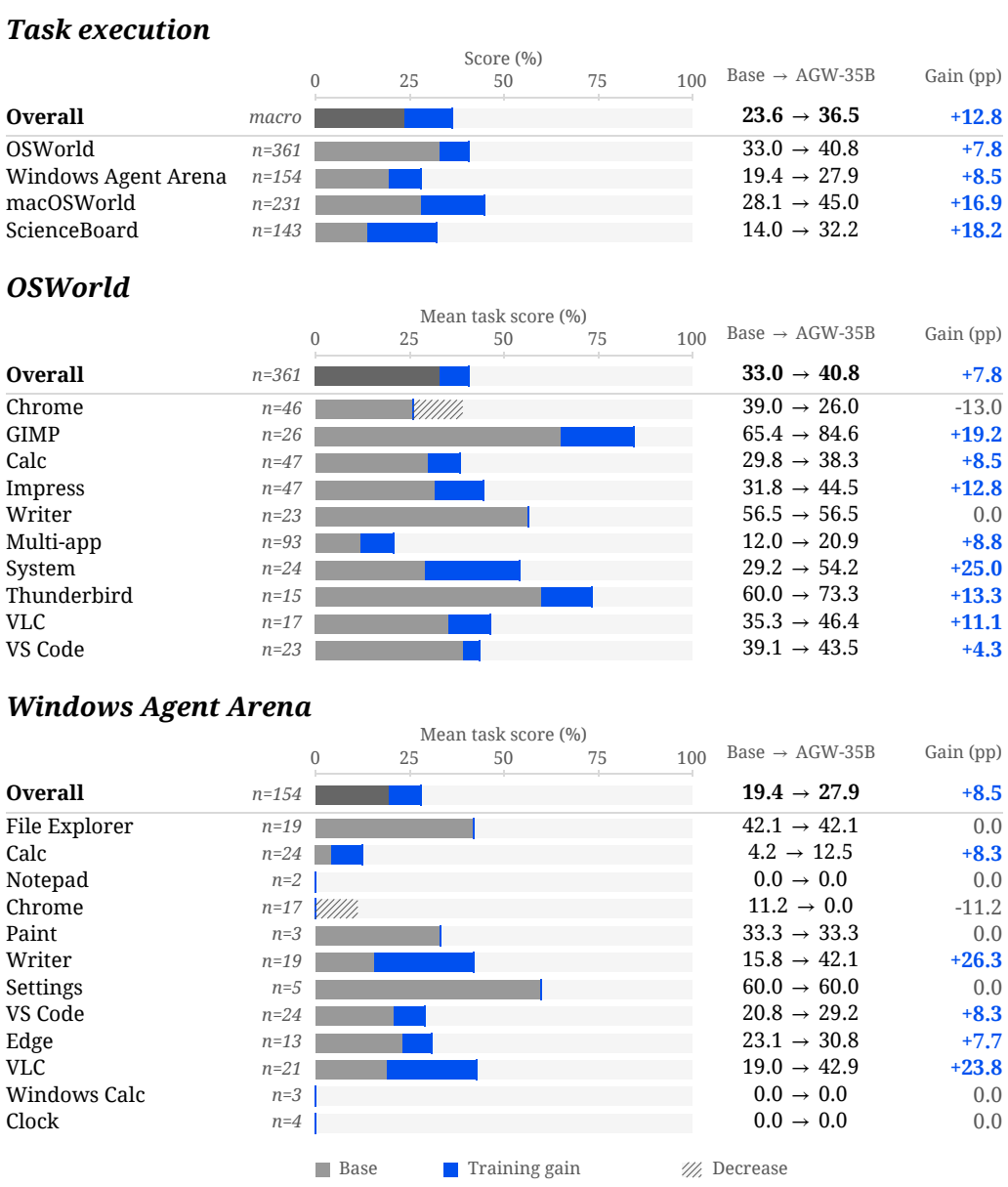

Figure 15: Task execution scores for Qwen3.5-35B-A3B and AGW-35B (checkpoint 469), with all OSWorld and WAA domains. The top Overall averages the four benchmark scores equally. Within each benchmark, Overall averages task scores, retaining partial credit on OSWorld and WAA. Gray shows base scores, blue shows gains, and hatching marks decreases. Blue ticks mark AGW-35B scores. n counts tasks, and gains are computed before rounding.

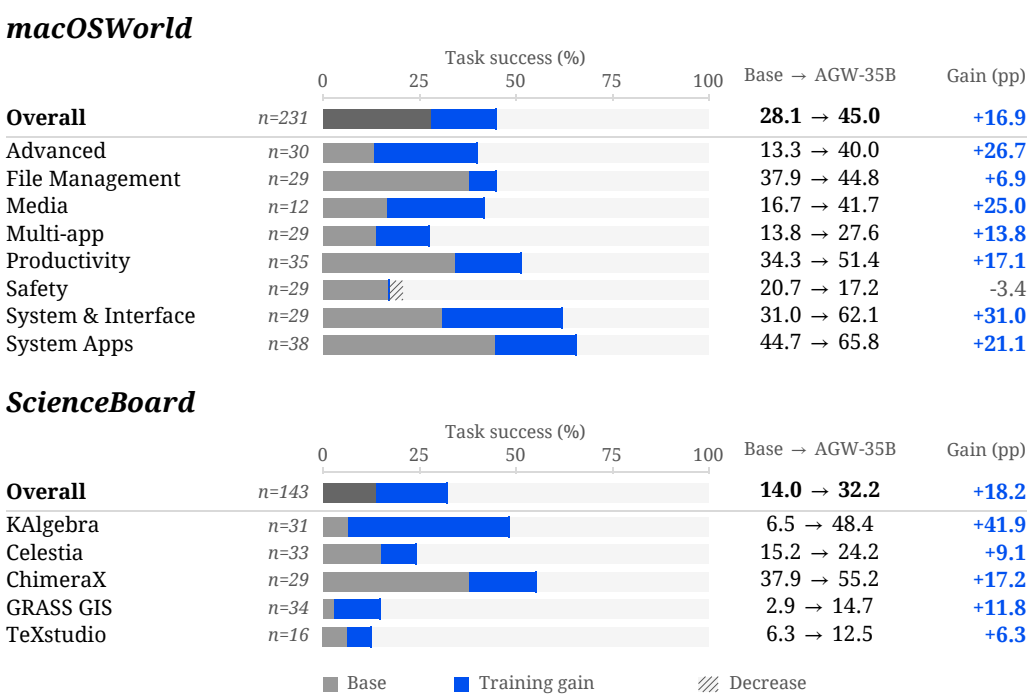

Figure 16: All task domains on macOSWorld and ScienceBoard. Gray bars show base success rates, blue extensions show gains, and gray hatching marks decreases. Blue ticks identify AGW-35B scores. Overall uses all tasks in each benchmark. ScienceBoard uses the 143-task subset in Appendix F.2.

Performance over training. As shown in Figure 5, AGW-35B improves on every benchmark by step 80 and reaches its highest score on all four at step 469. ScienceBoard success rises from 14.0% to 32.2%, increasing at each evaluated checkpoint. macOSWorld rises from 28.1% to 45.0%, with continued gains after step 157. OSWorld and WAA fluctuate between checkpoints and finish at 40.8% and 27.9%, respectively. The four-benchmark mean increases at every evaluation, from 23.6% for the base model to 36.5% at the final checkpoint.

Table 23: Interactive scores across training (%). Overall is the equal-weight mean of the four benchmark scores, rounded to one decimal place.

Checkpoint	OSWorld	WAA	macOSWorld	ScienceBoard	Overall
Base	32.98	19.41	28.14	13.99	23.6
80	36.22	24.71	38.1	18.18	29.3
157	36.13	26.01	36.36	26.57	31.3
235	39.29	21.43	38.53	28.67	32.0
314	36.4	26.66	40.26	29.37	33.2
392	38.03	24.06	41.99	31.47	33.9
469	40.77	27.92	45.02	32.17	36.5

The domain and checkpoint figures use the corrected per-task evaluation records archived with the manuscript. Missing or nonnumeric results receive zero credit under the fixed-denominator aggregation above. The checkpoints come from one training run; their progression does not quantify variation across independent runs.

G Extended Related Work

G.1 GUI Trajectory Collection

GUI trajectories have traditionally come from people operating live websites or mobile devices, as in Mind2Web, WebLINX, Android in the Wild, and GUIOdyssey (Deng et al., 2023; Lù et al., 2024; Rawles et al., 2023; Lu et al., 2024). Recent pipelines scale this supervision by converting tutorials and screen recordings into action traces. TongUI and MONDAY process web tutorials or instructional videos (Zhang et al., 2025; Jang et al., 2025), while Watch and Learn, VideoAgentTrek, and Video2GUI recover executable trajectories through inverse dynamics (Song et al., 2025; Lu et al., 2025a; Xiong et al., 2026). Automated pipelines instead explore executable interfaces: OS-Genesis infers tasks retrospectively, AgentTrek performs tutorial-guided replay, and GUI-explorer mines transition knowledge (Sun et al., 2024; Xu et al., 2024; Xie et al., 2025a). UI-Genie and AgentSynth combine task proposal, execution, and verification (Xiao et al., 2025; Xie et al., 2025b), while GUI-ReWalk, MobileGen, TreeCUA, HATS, and SEE improve exploration diversity, difficulty, and long-horizon coverage (Lin et al., 2025; Kang et al., 2026; Jiang et al., 2026; Shao et al., 2026; Fan et al., 2026). UI-TARS, OpenCUA, ScaleCUA, and WebSTAR further turn rollout and filtering into data flywheels (Qin et al., 2025; Wang et al., 2025a; Liu et al., 2025; He et al., 2025). Another line scales the rollout substrate: AndroidWorld and OSWorld provide resettable environments, while InfiniteWeb, AutoWebWorld, and CUA-Gym synthesize environments, tasks, and verifiers (Rawles et al., 2024; Xie et al., 2024; Zhang et al., 2026; Wu et al., 2026; Wang et al., 2026a). Despite different supervision sources, these methods obtain new transitions by executing actions in real, sandboxed, or synthetic environments, tying scale to environment construction, reset, execution, and verification. VEM and UI-S1 avoid online execution but optimize over existing offline trajectories (Zheng et al., 2025; Lu et al., 2025b). AutoGUIWorld instead synthesizes *new* screenshot-level trajectories without constructing or running the target software: a planner specifies an atomic action and intended state change, and an image generator produces the next observation. Replacing environment rollout with visual state generation decouples trajectory scale from executable GUI availability while retaining real environments for final validation.

G.2 GUI World Model

A GUI world model approximates the action-conditioned transition of an interface before the action is executed. More broadly, digital world models capture computational dynamics: WorldCoder induces executable transition programs, CWM learns from Python and Docker interaction traces, and Qwen-AgentWorld scales language-based next-state prediction across multiple agent domains (Tang et al., 2024; FAIR CodeGen Team, 2025; Zuo et al., 2026). For the web, WMA abstracts changes between long HTML observations (Chae et al., 2024); WebDreamer and SimuRA imagine action outcomes for lookahead planning (Gu et al., 2024; Deng et al., 2025); WebEvolver co-evolves a transition predictor with the agent (Fang et al., 2025); and DynaWeb and WebWorld support multi-step imagined rollouts for policy learning and inference-time search (Ding et al., 2026; Xiao et al., 2026). These methods largely use textual or structured states. Visual GUI world models instead generate future screens. ViMo combines conditional diffusion over a symbolic-text representation with separate text generation (Luo et al., 2025), while NeuralOS couples recurrent state tracking with a diffusion renderer (Rivard et al., 2025). UISim predicts an abstract layout before synthesizing the next image (Xiang et al., 2025); UI-Simulator generates structured UI transitions for trajectory training (Wang et al., 2025b); and MobileDreamer predicts compact textual sketches (Cao et al., 2026). The Computer-Using World Model predicts a textual state change before rendering the screenshot (Guan et al., 2026), whereas Code2World and gWorld generate executable code and obtain the visual state through a renderer (Zheng et al., 2026; Koh et al., 2026). Recent work also compares delta text, full text, diffusion images, and renderable code as prediction targets (Xu et al., 2026). Across these representations, most methods train dedicated transition models on interactions executed in web, mobile, desktop, or computational environments, primarily for planning or policy learning. AutoGUIWorld instead uses a general-purpose image generator with broadly useful visual priors (Gabeur et al., 2026) as an implicit GUI world model. It chain-edits screenshots from atomic actions and planner-specified transitions, directly producing environment-free screenshot–action–screenshot training rollouts across platforms, tasks, appearances, and interaction patterns.

H Qualitative Trajectory Examples

We organize additional qualitative examples by GUI environment. Each row is a trajectory, and the three columns show the initial state, a key spatial action, and the final state. The row-level text summarizes the task, the intermediate action, and the resulting state, while red boxes and crosshairs indicate the target region or point for the key action.

H.1 Ubuntu Desktop

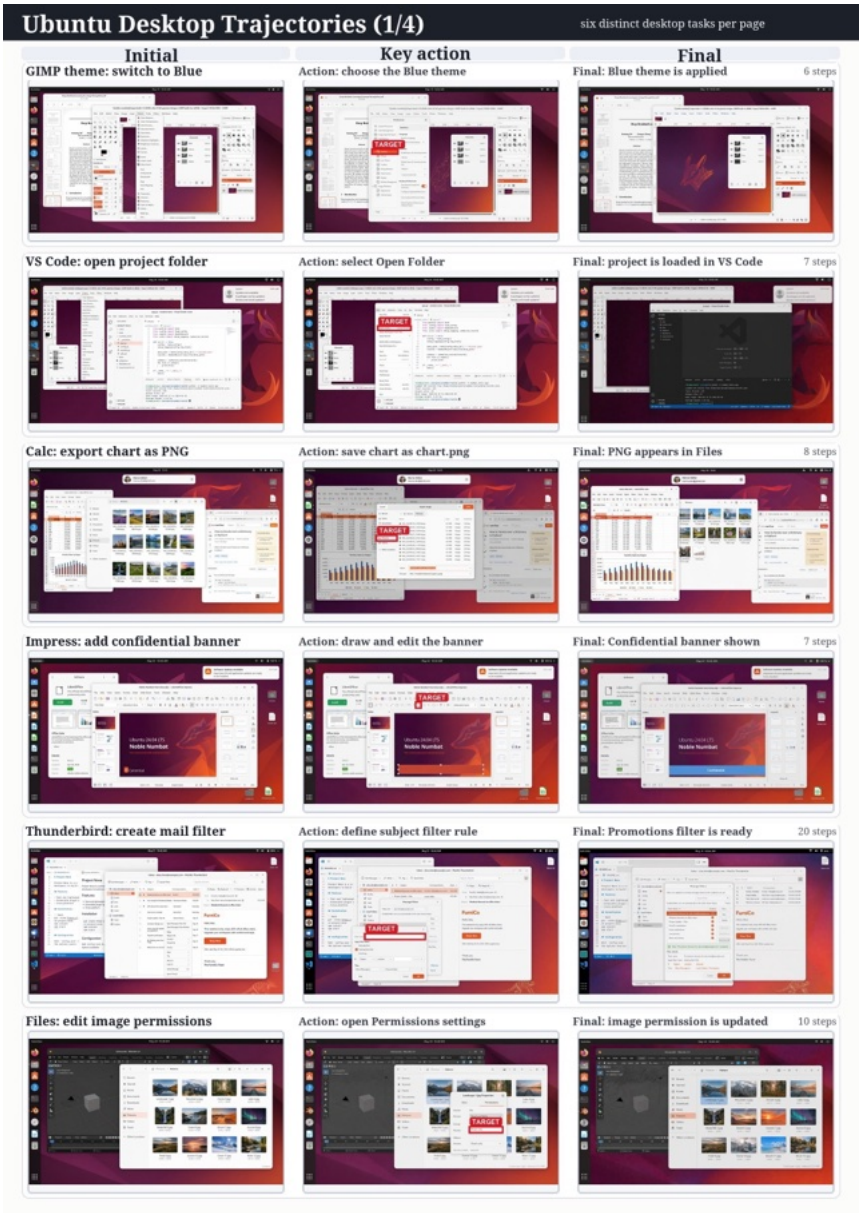

Figure 17: Ubuntu desktop trajectories (1/4).

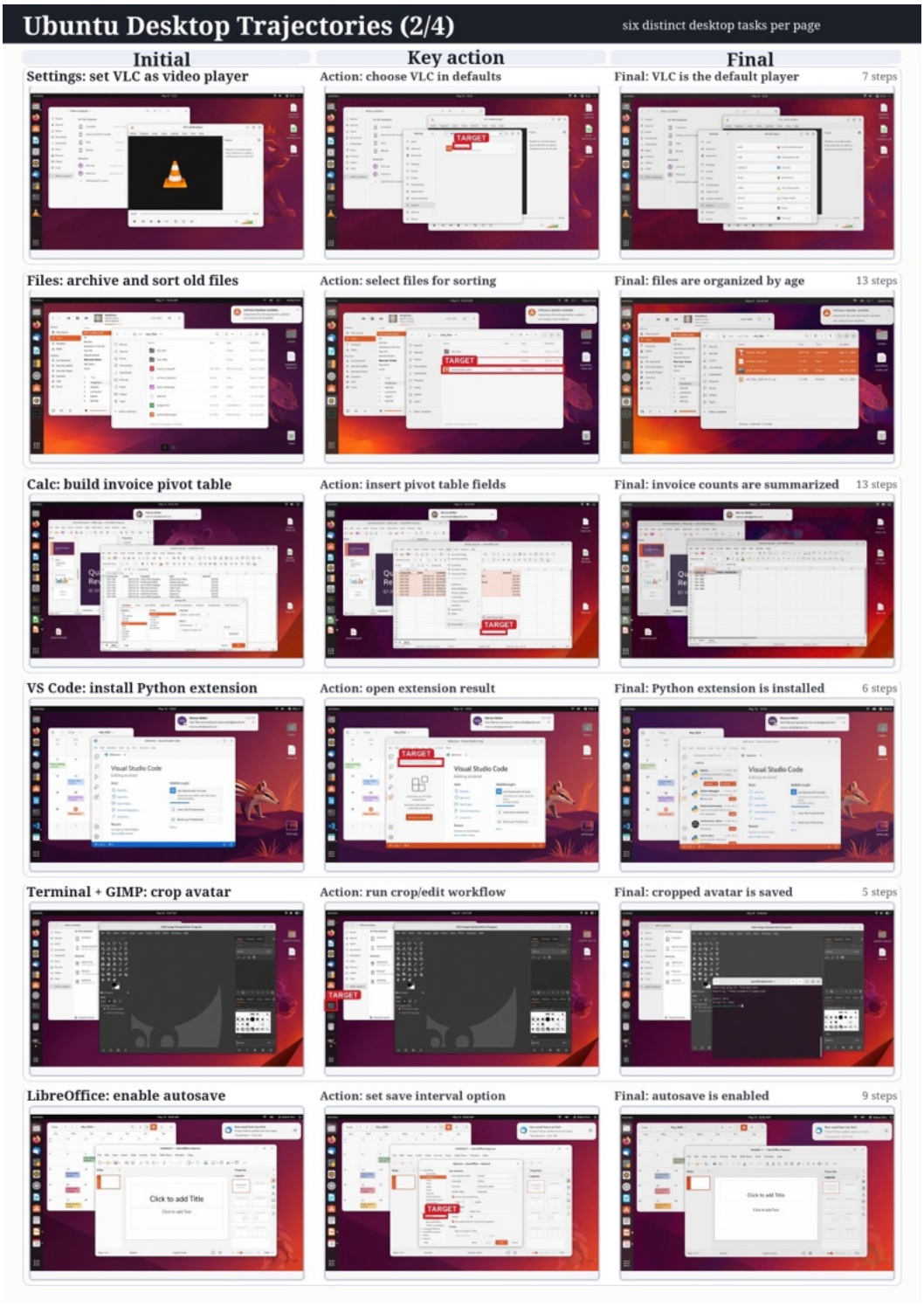

Figure 18: Ubuntu desktop trajectories (2/4).

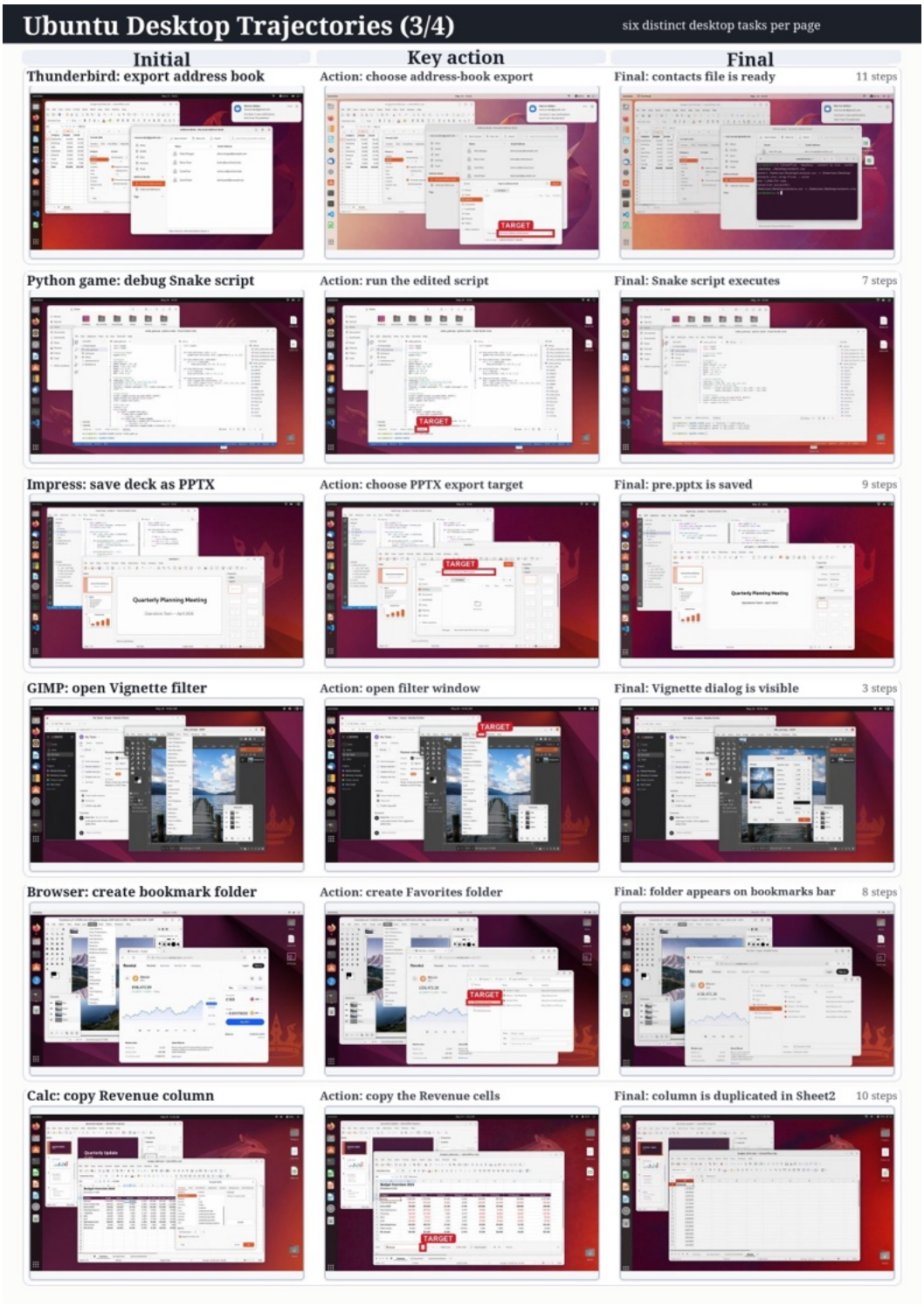

Figure 19: Ubuntu desktop trajectories (3/4).

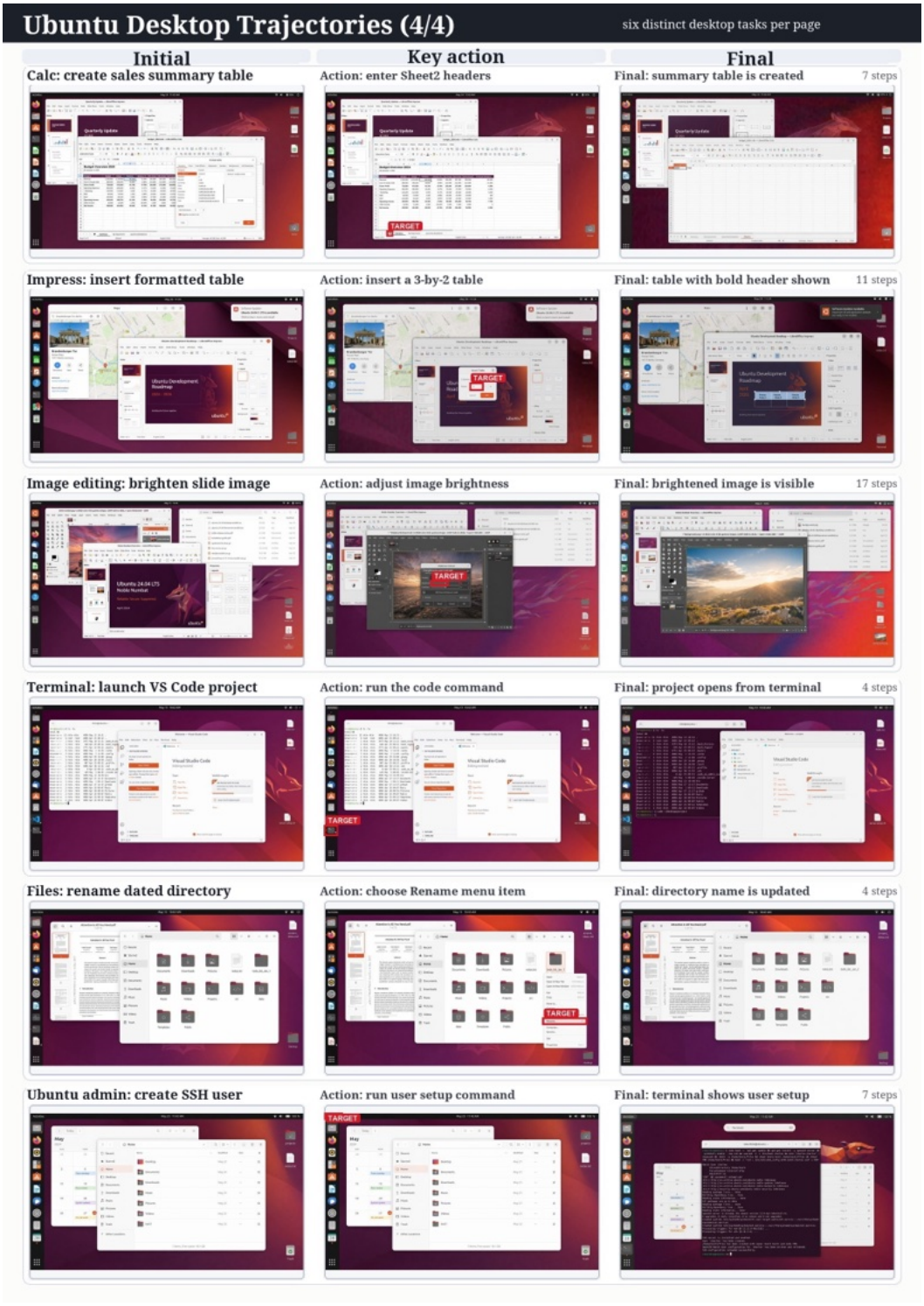

Figure 20: Ubuntu desktop trajectories (4/4).

H.2 WINDOWS 11

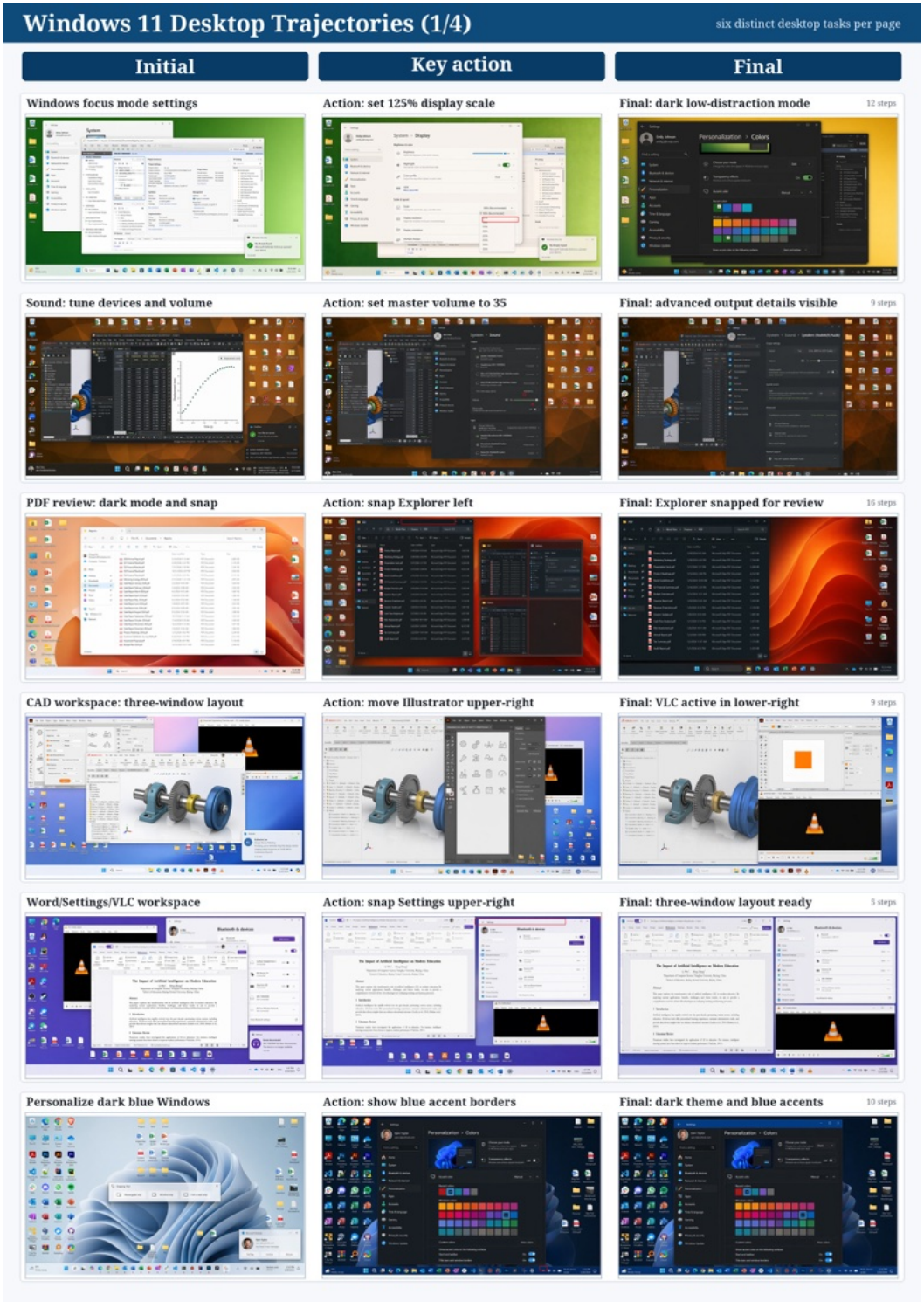

Figure 21: Windows 11 trajectories (1/4).

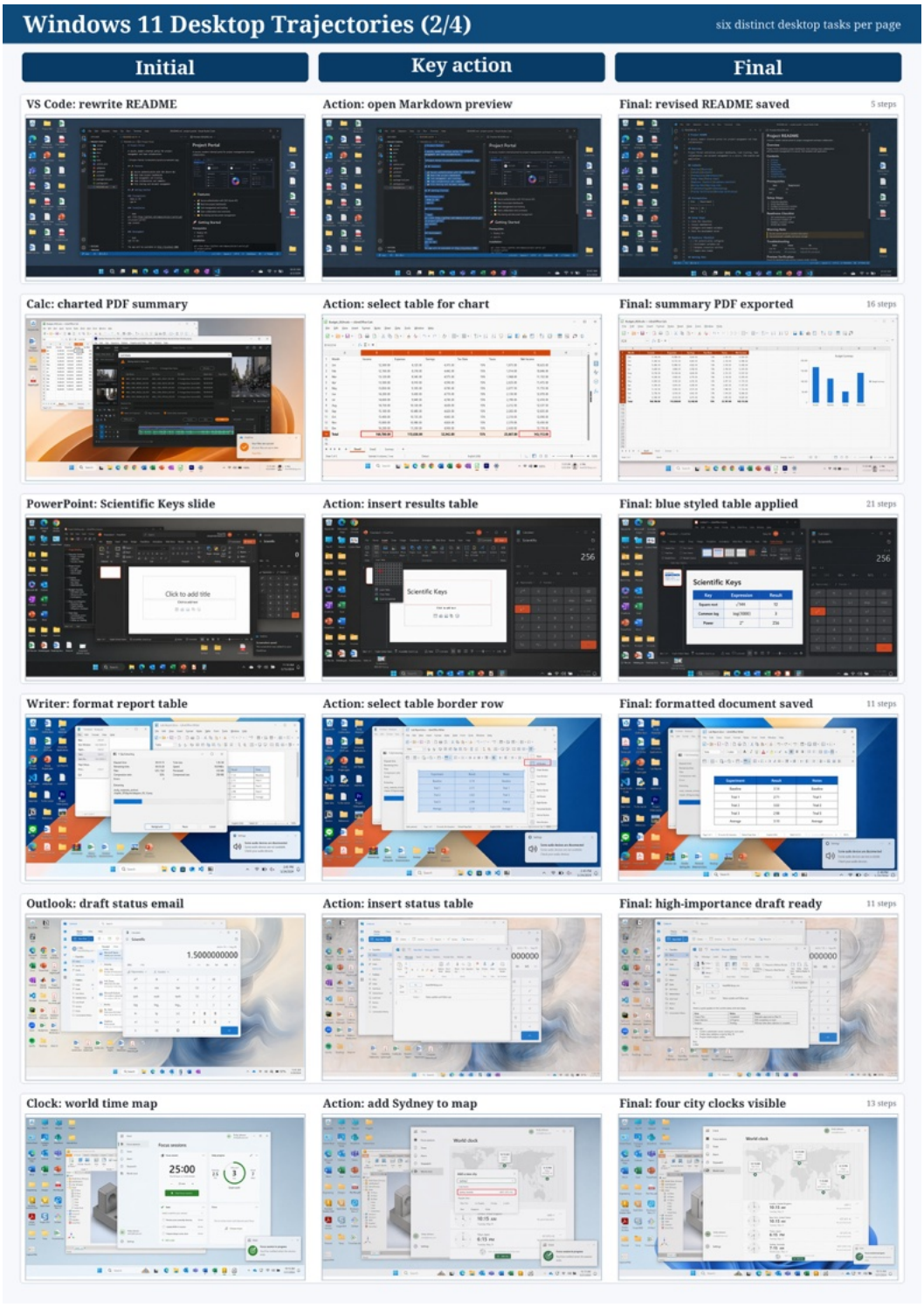

Figure 22: Windows 11 trajectories (2/4).

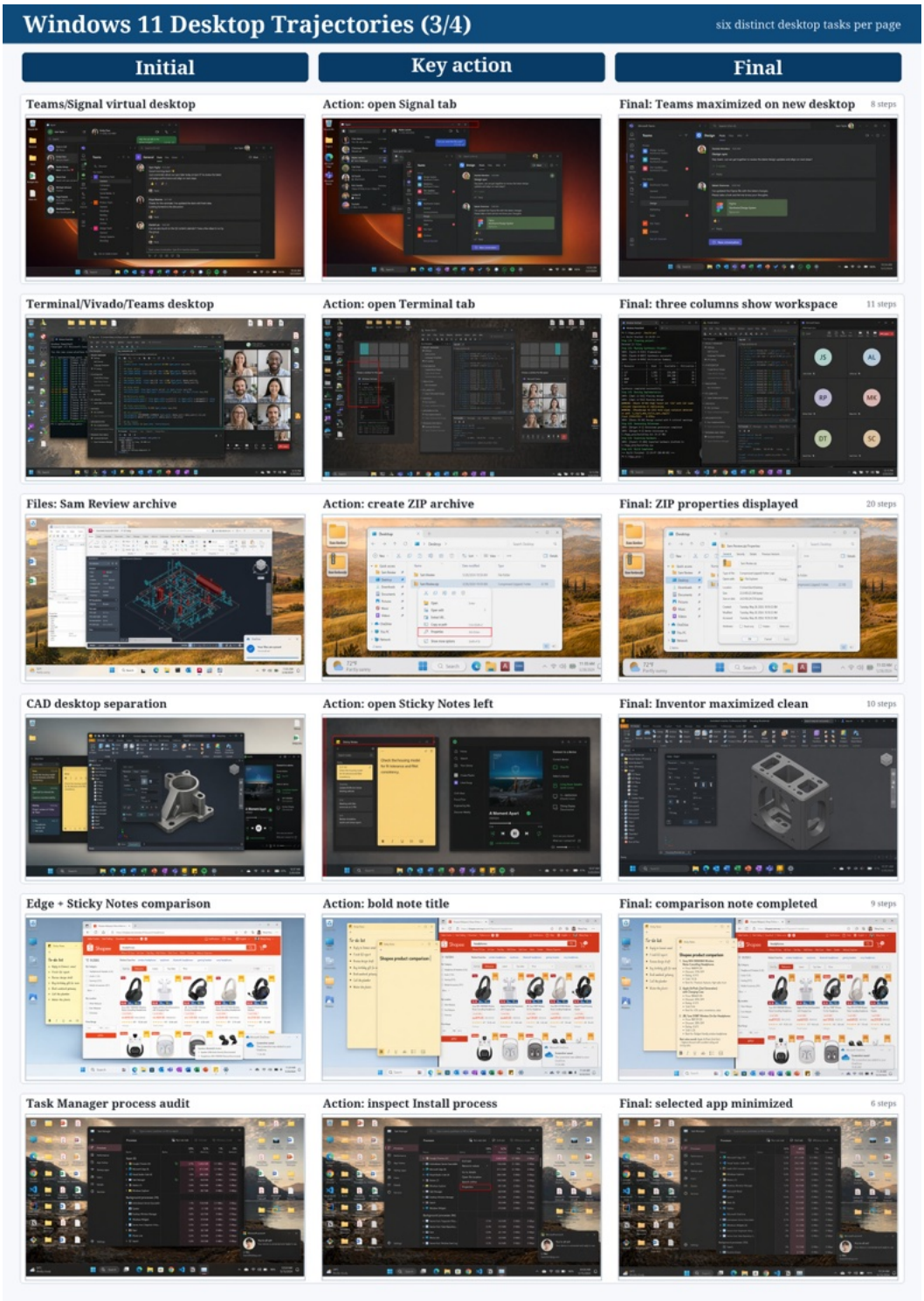

Figure 23: Windows 11 trajectories (3/4).

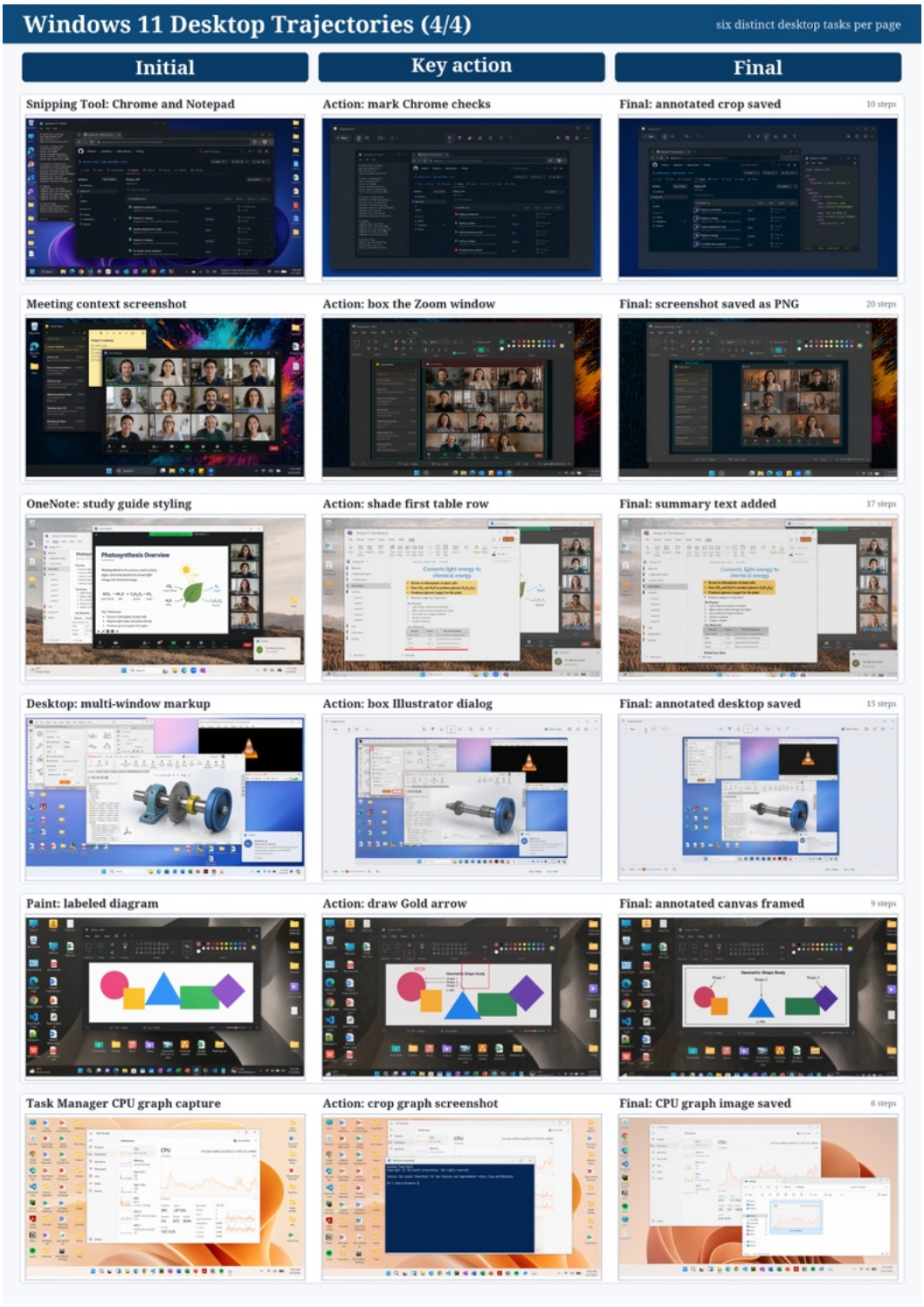

Figure 24: Windows 11 trajectories (4/4).

H.3 MACOS

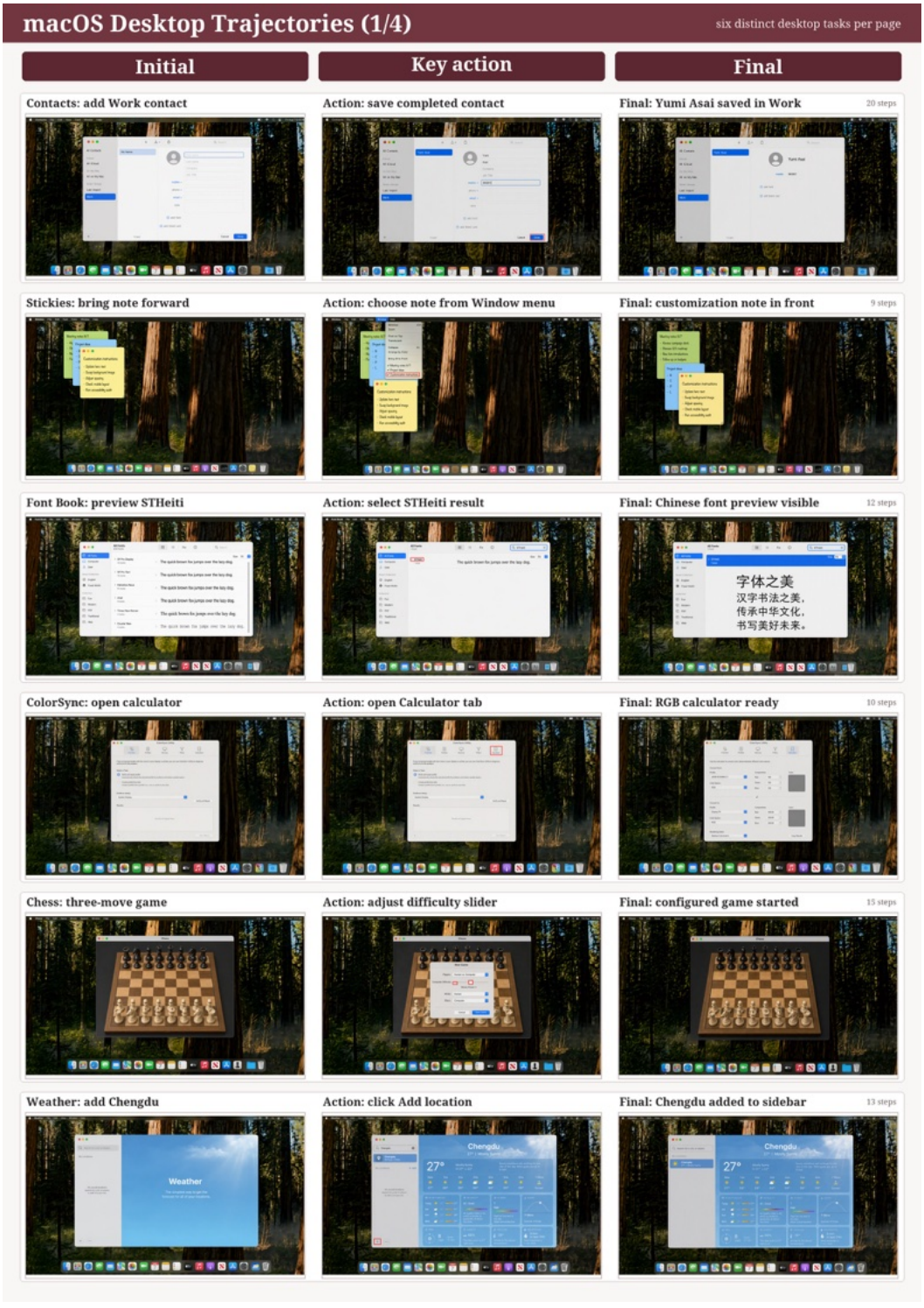

Figure 25: macOS trajectories (1/4).

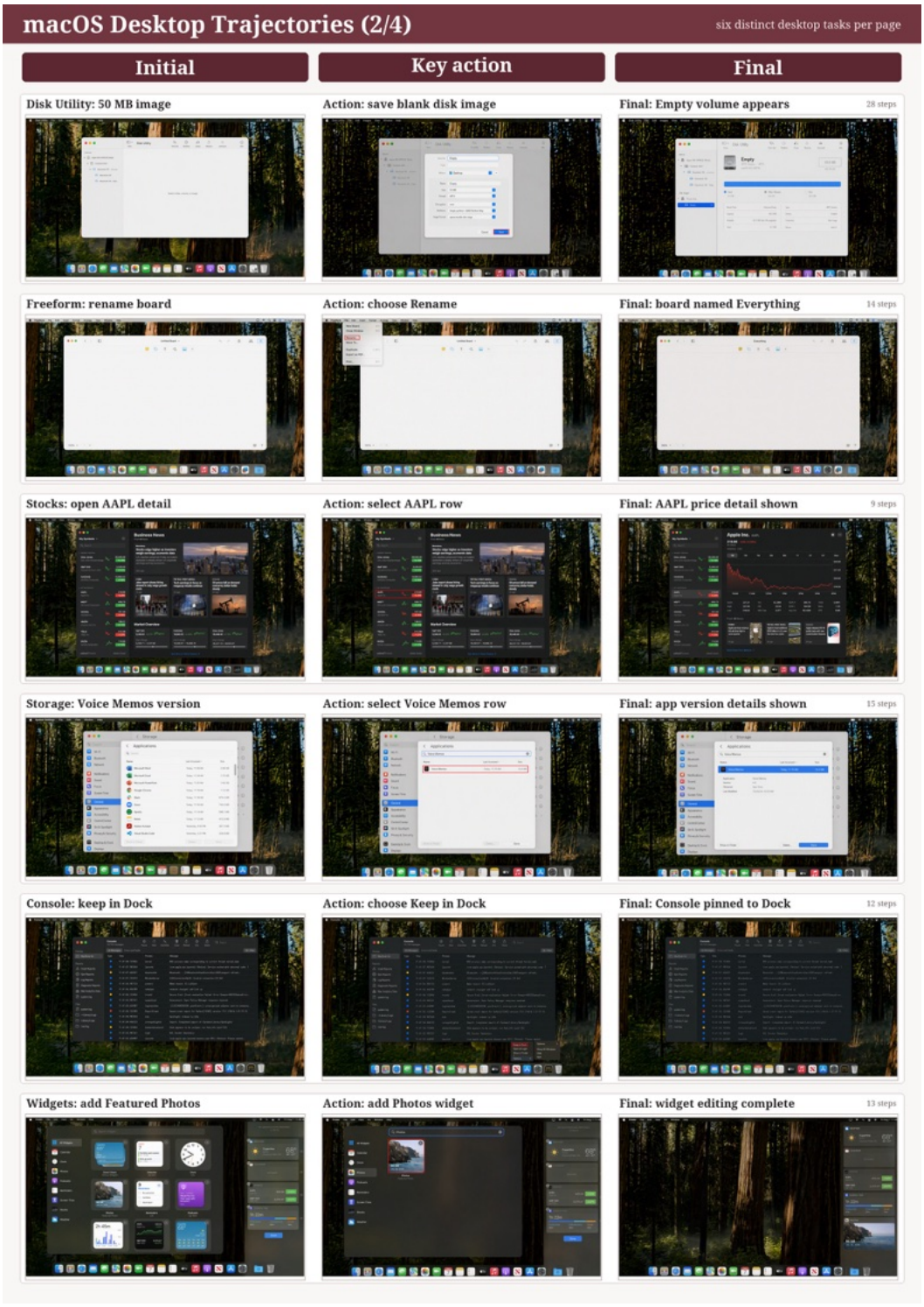

Figure 26: macOS trajectories (2/4).

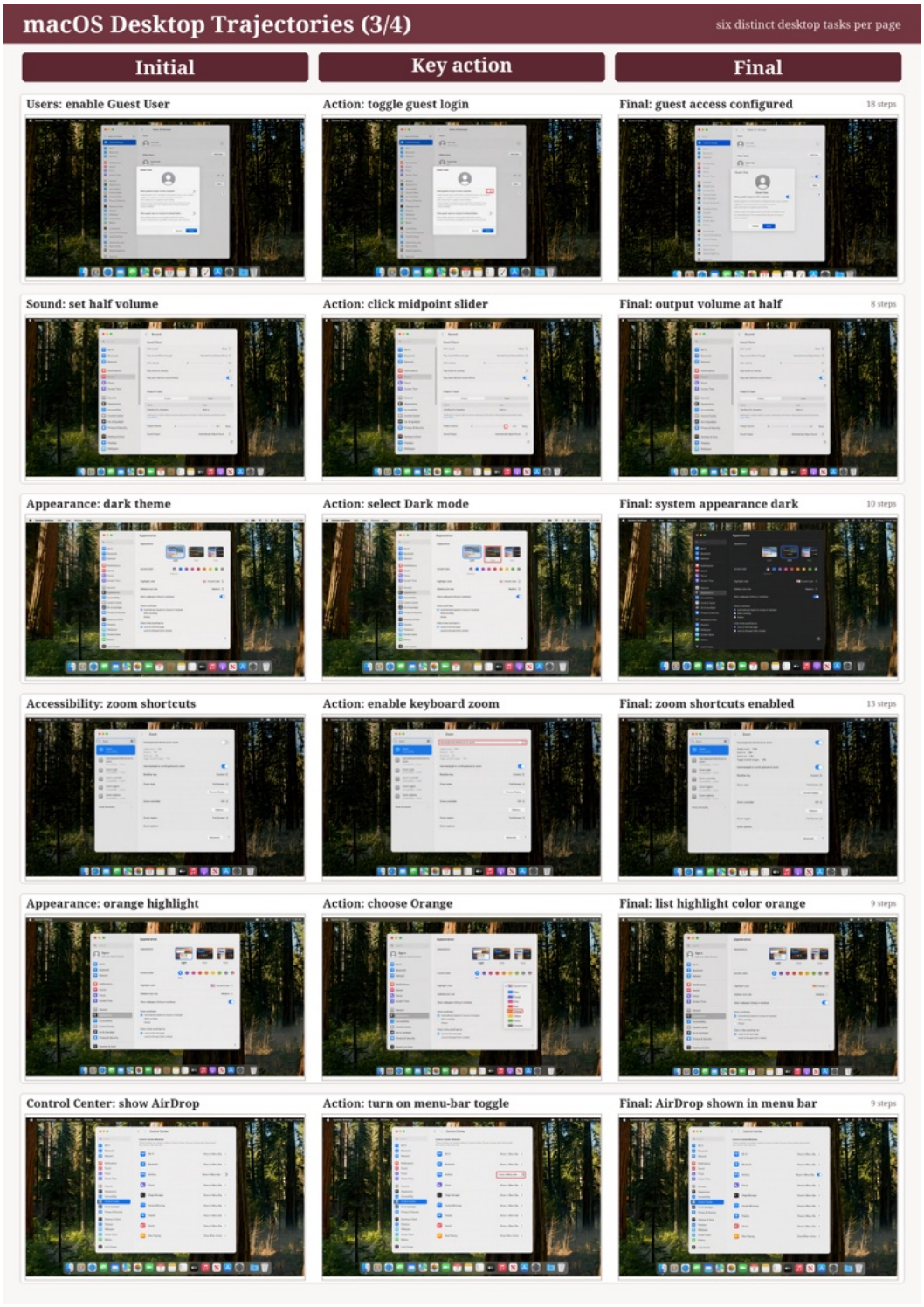

Figure 27: macOS trajectories (3/4).

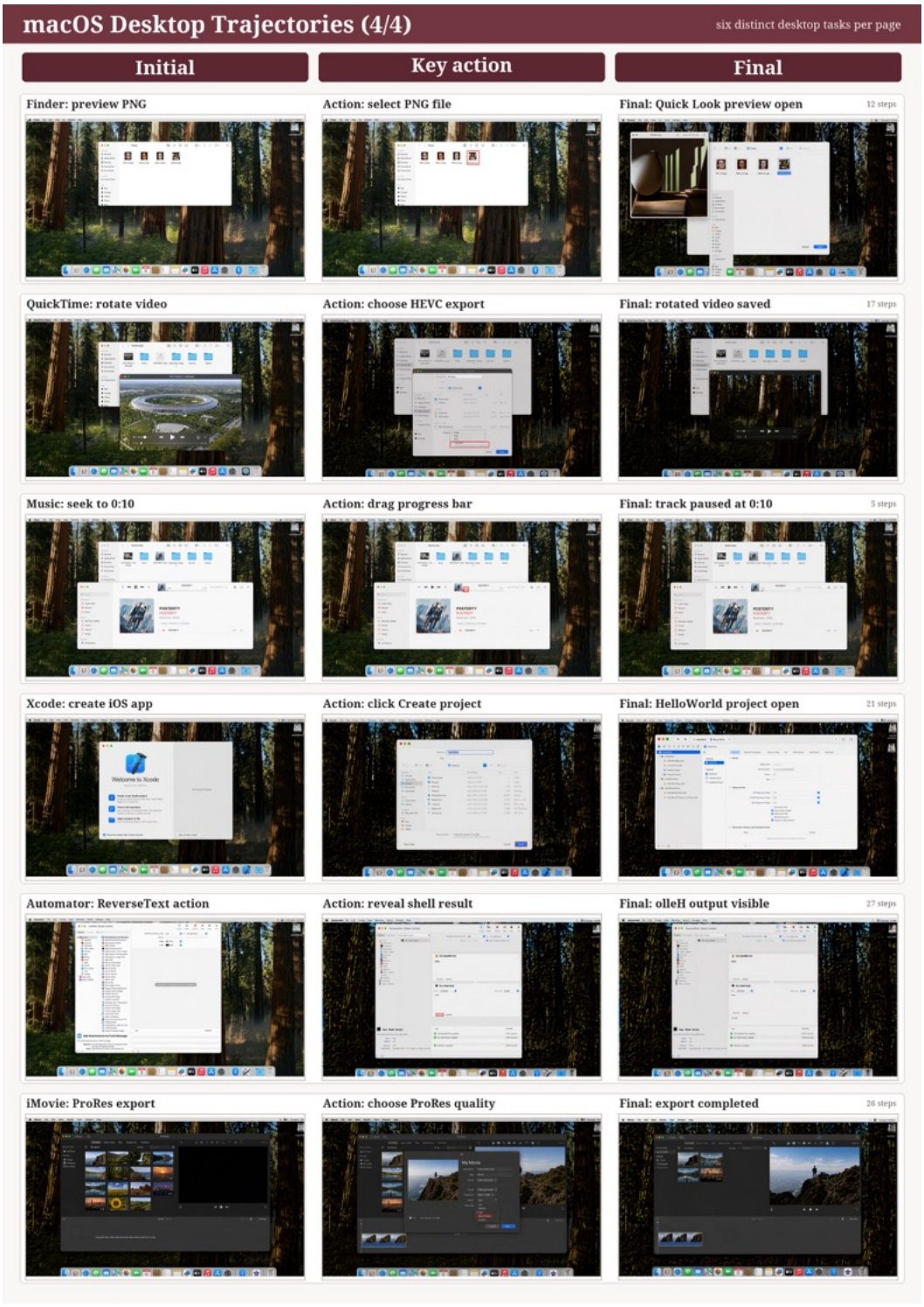

Figure 28: macOS trajectories (4/4).

H.4 CHROME

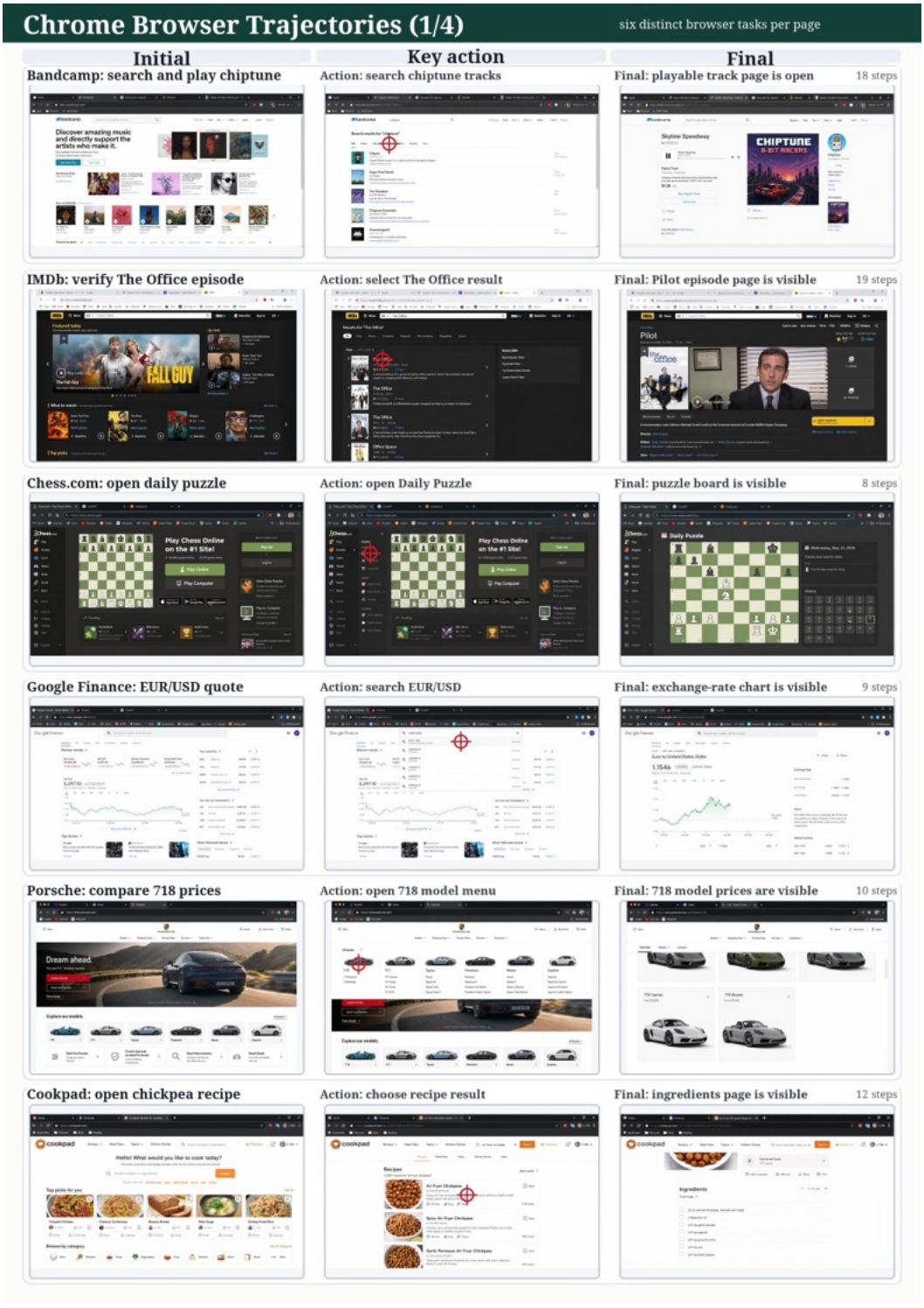

Figure 29: Chrome trajectories (1/4).

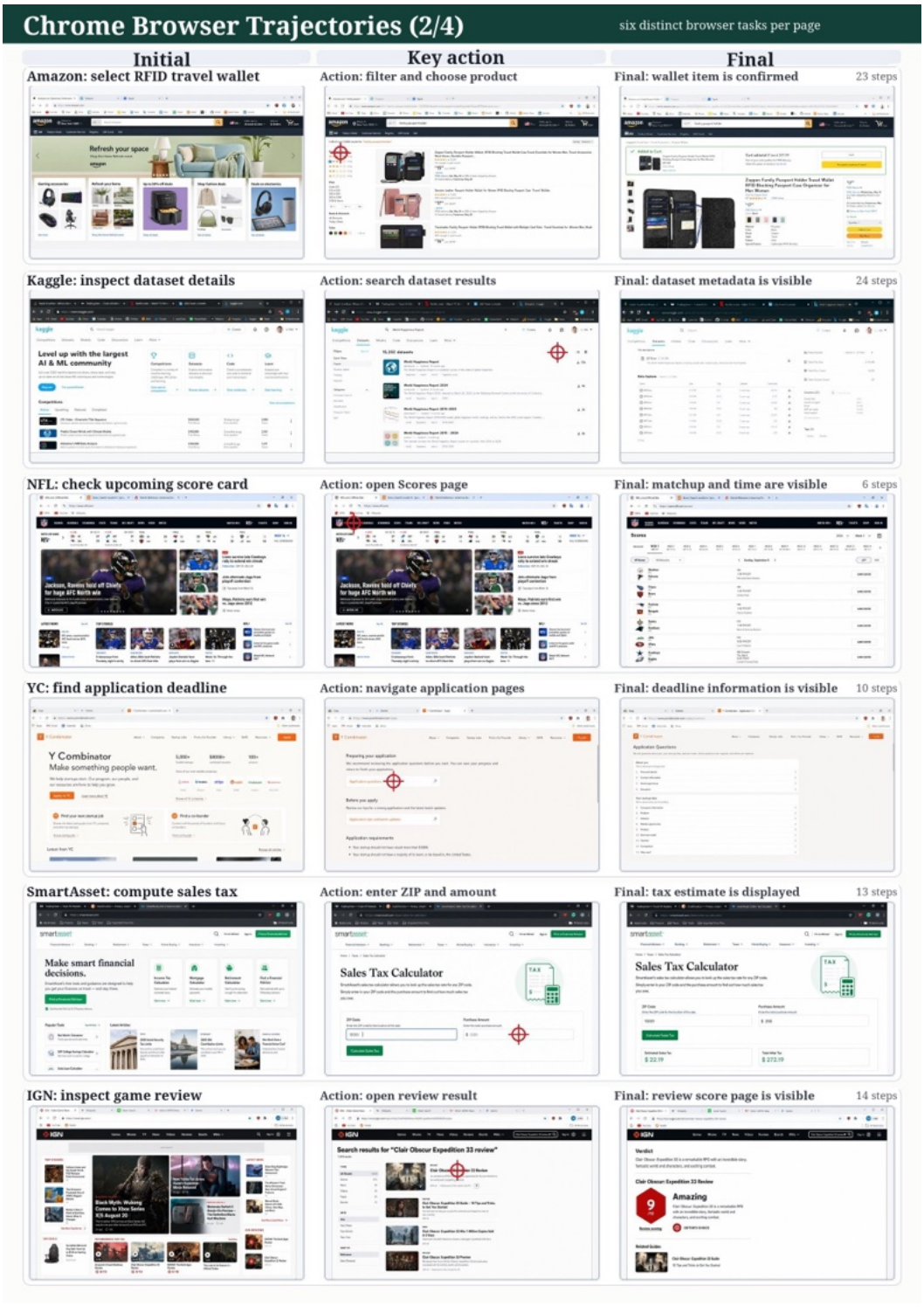

Figure 30: Chrome trajectories (2/4).

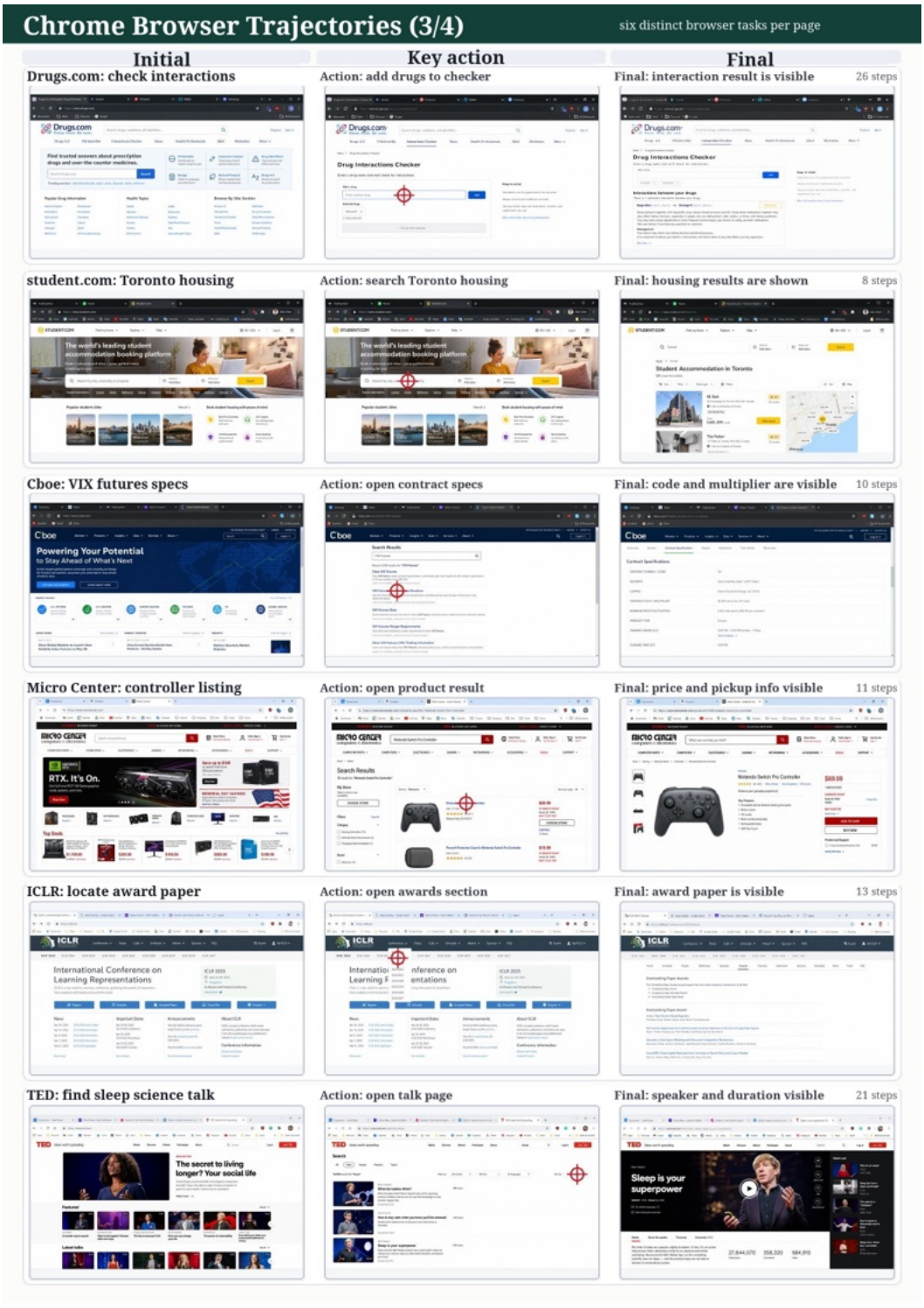

Figure 31: Chrome trajectories (3/4).

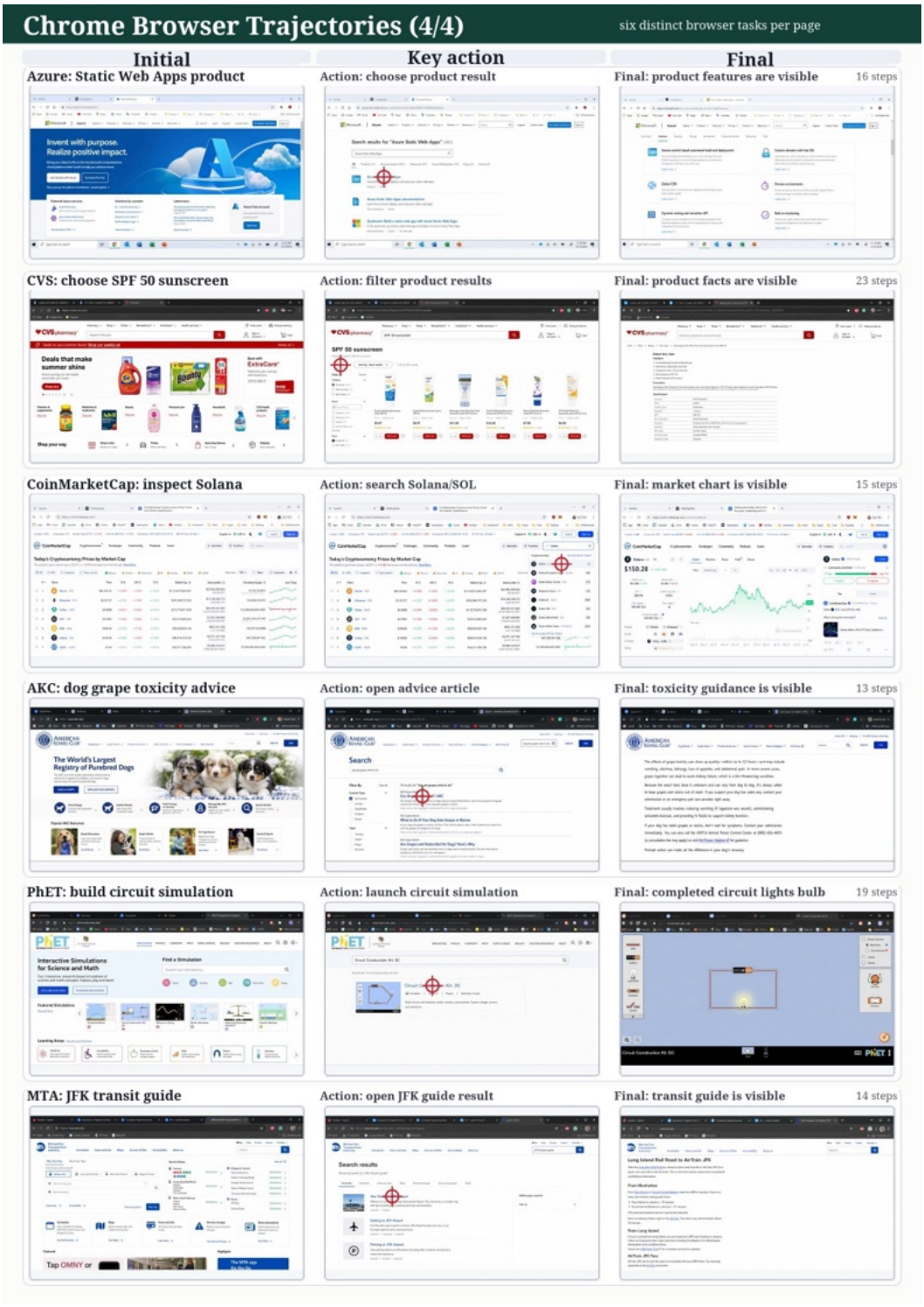

Figure 32: Chrome trajectories (4/4).

I PROFESSIONAL APPLICATION WORKFLOWS

We further examine whether AutoGUIWorld can produce coherent workflows in dense professional interfaces spanning seven complementary domains. The coverage includes engineering authoring, visual design, quantitative analysis, electronic design, audio production, scientific research, and development automation. The following pages present representative professional outcomes within each branch of this hierarchy.

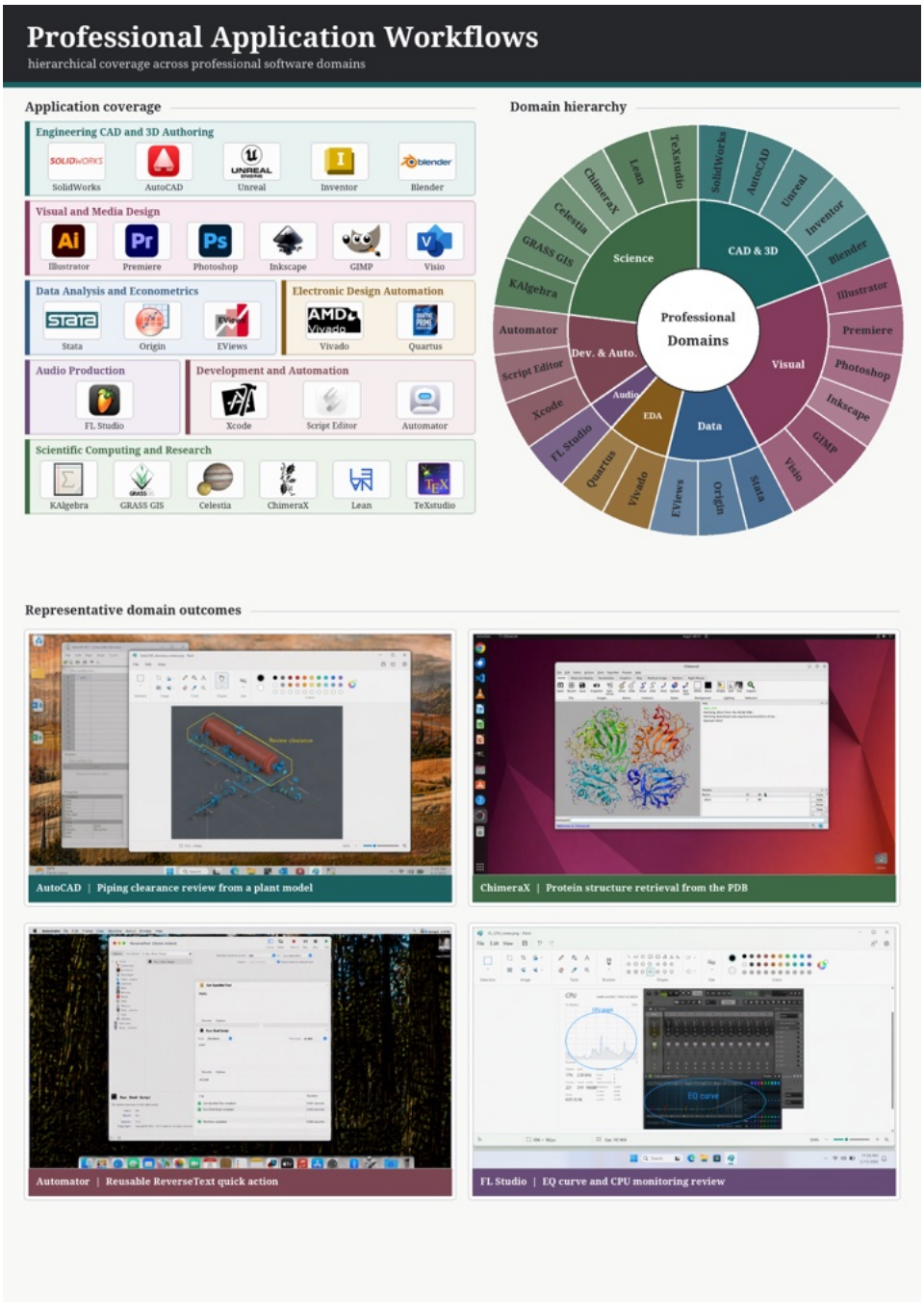

Figure 33: Hierarchical coverage of professional application domains.

I.1 Engineering CAD and 3D Authoring

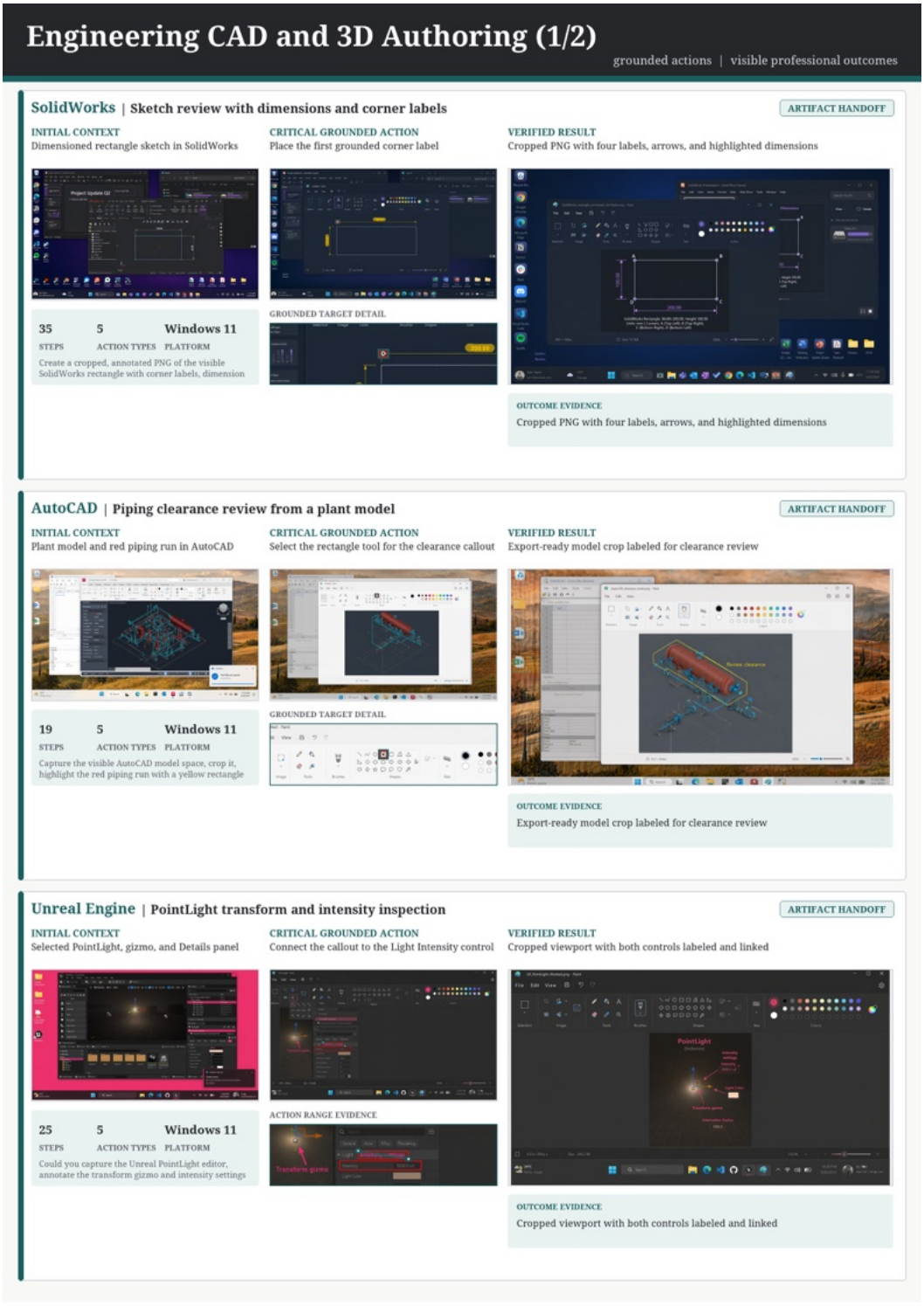

Figure 34: Engineering CAD and 3D authoring workflows (1/2).

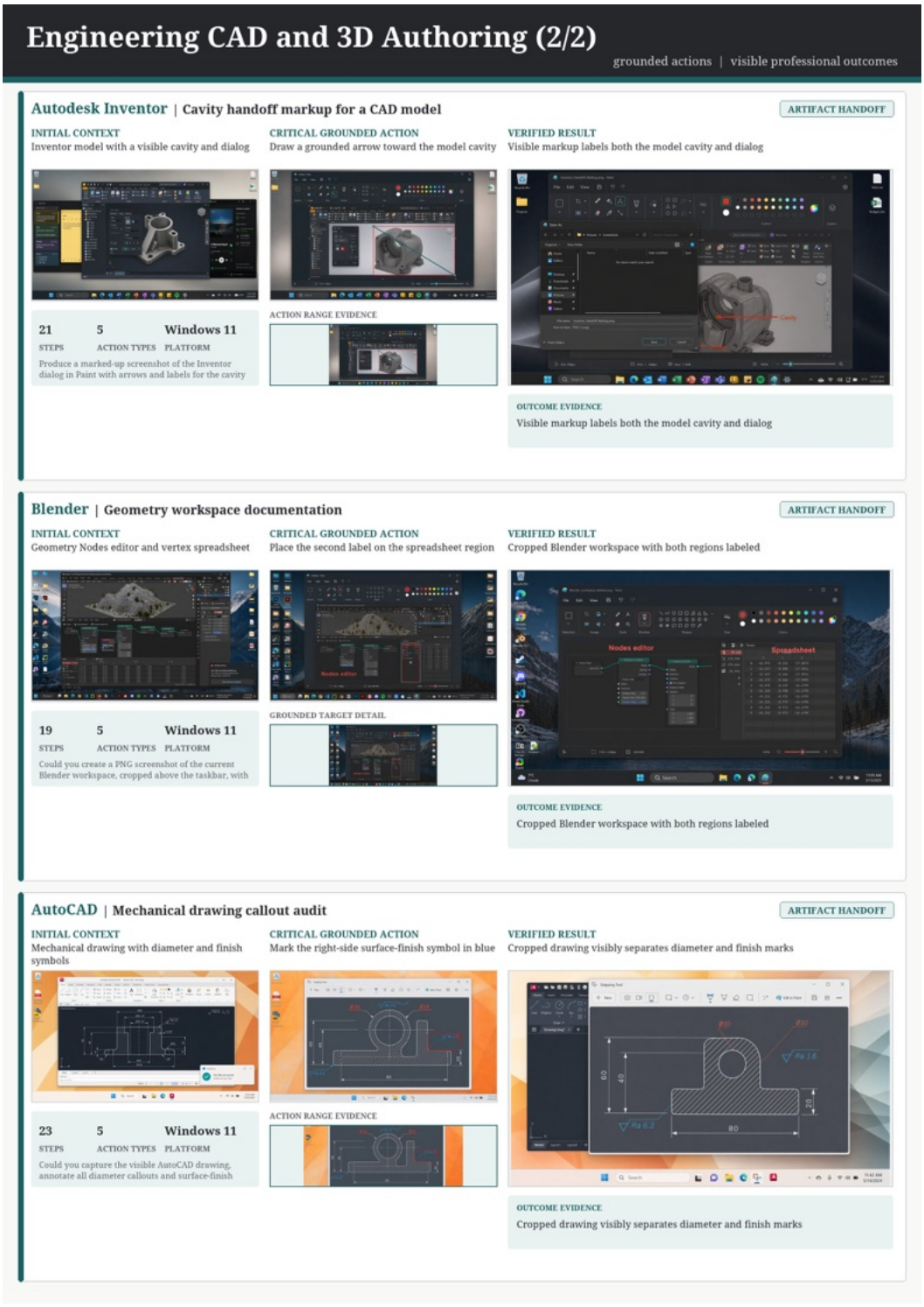

Figure 35: Engineering CAD and 3D authoring workflows (2/2).

I.2 Visual and Media Design



Figure 36: Visual and media design workflows (1/2).

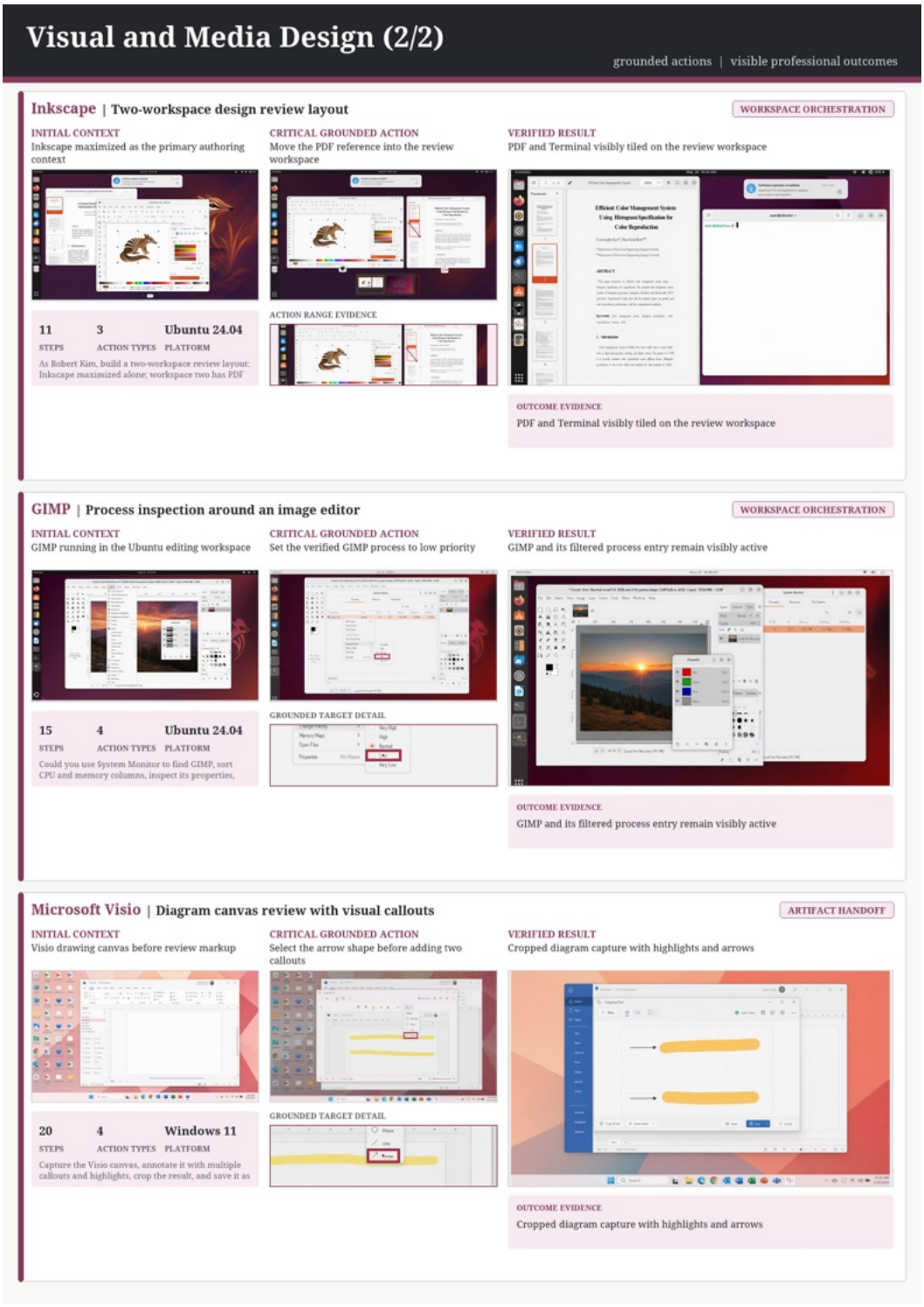

Figure 37: Visual and media design workflows (2/2).

I.3 Data Analysis and Econometrics

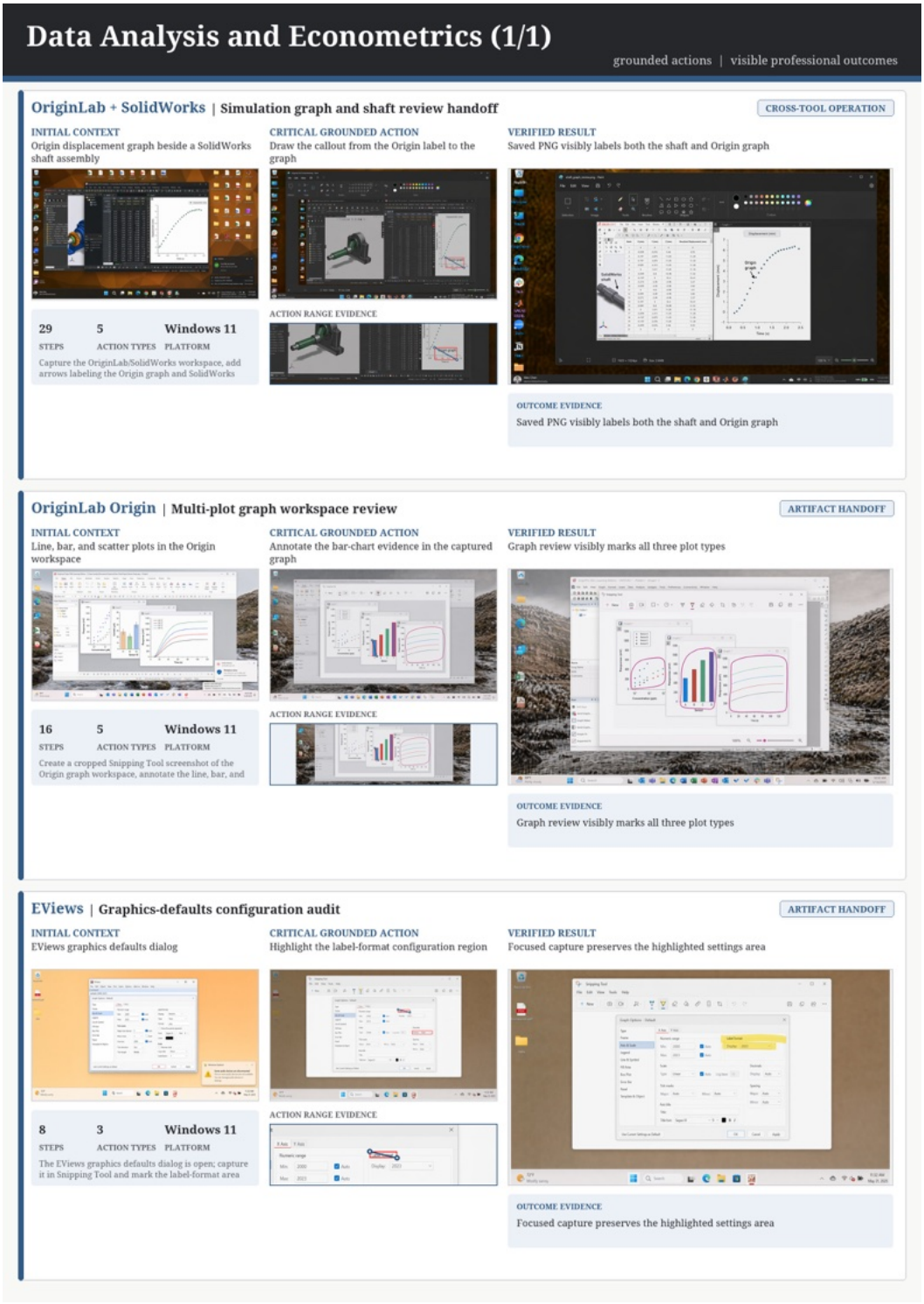

Figure 38: Data analysis and econometrics workflows.

I.4 Electronic Design Automation



Figure 39: Electronic design automation workflows.

I.5 AUDIO PRODUCTION

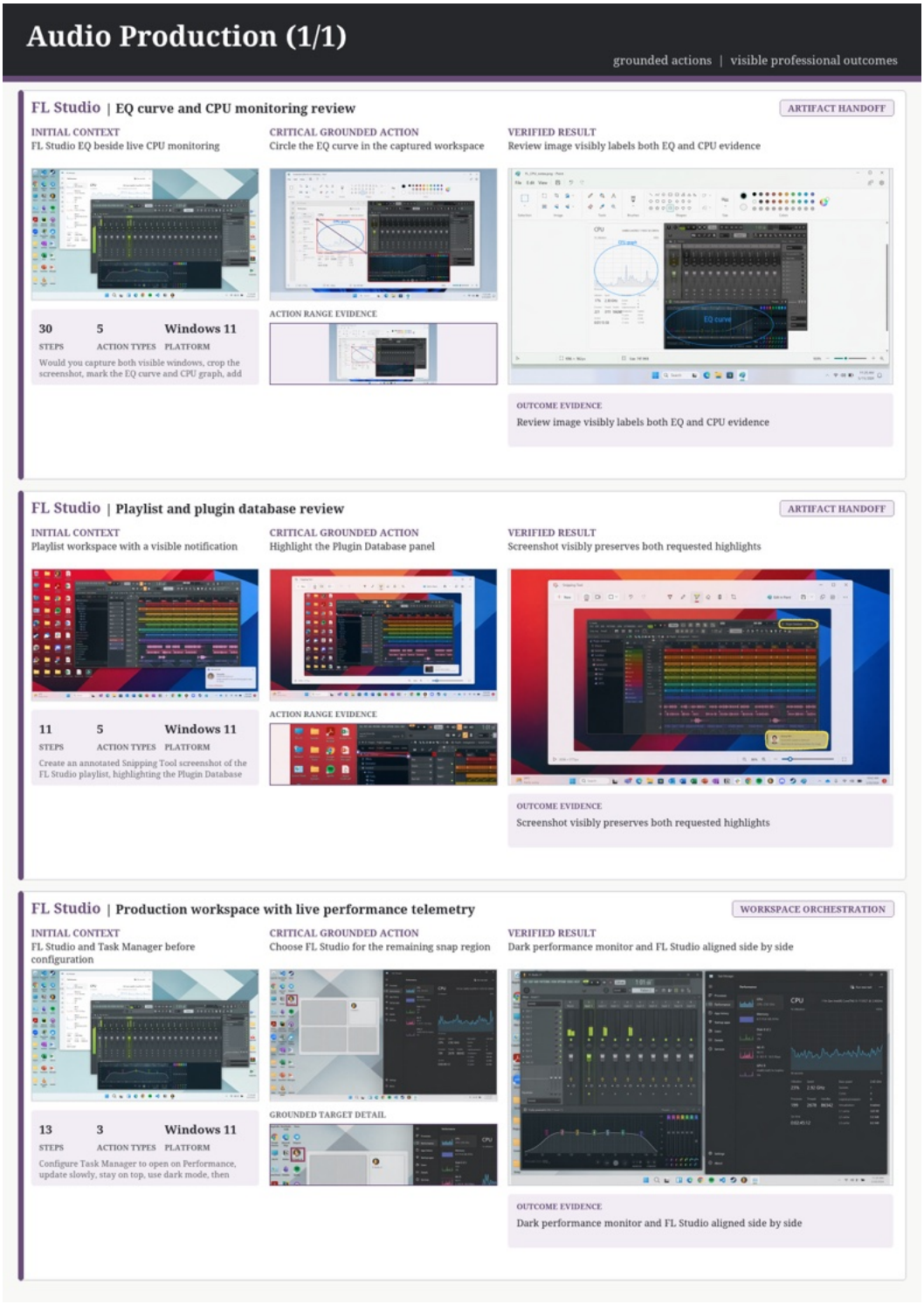

Figure 40: Audio production workflows.

I.6 Scientific Computing and Research

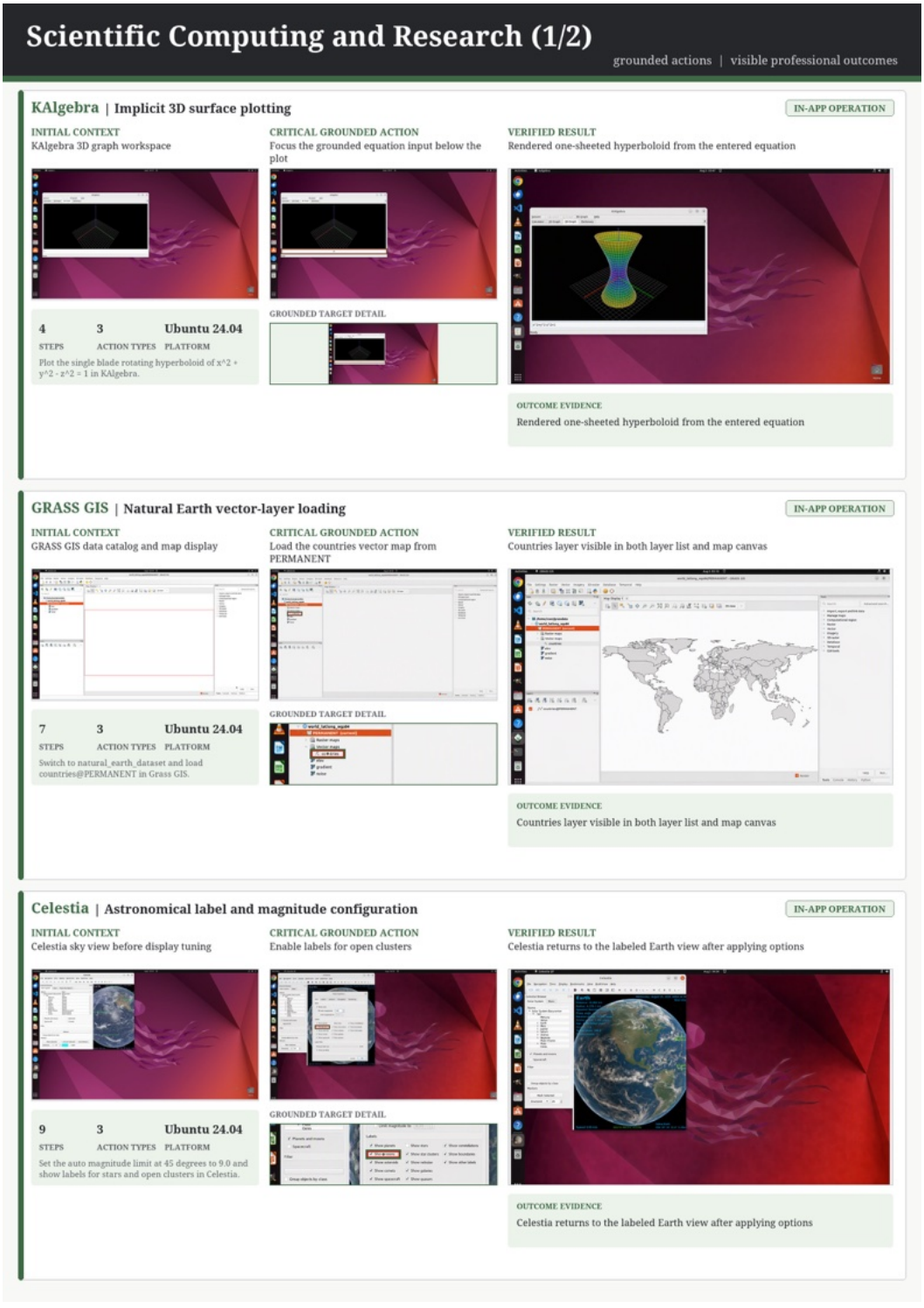

Figure 41: Scientific computing and research workflows (1/2).

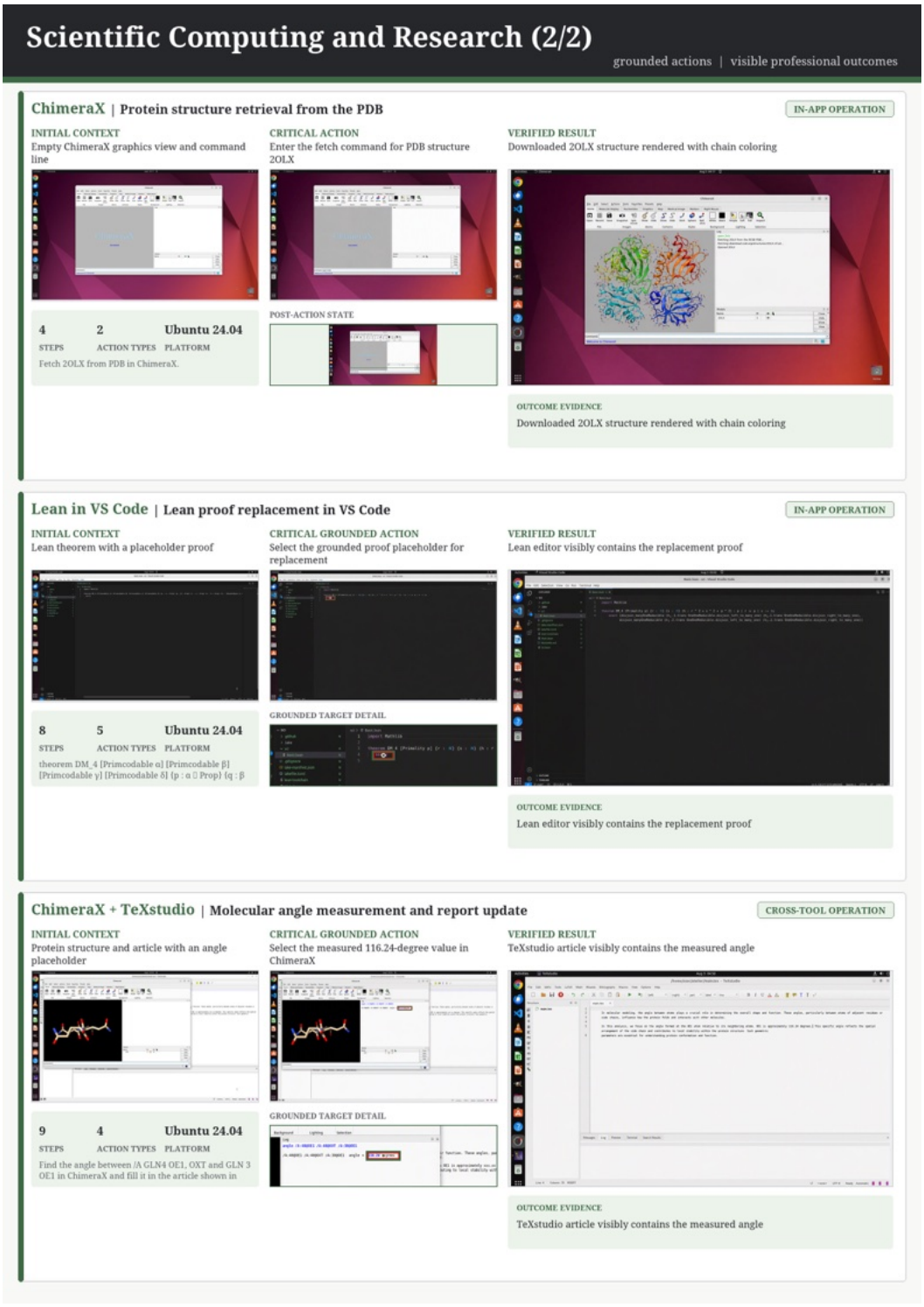

Figure 42: Scientific computing and research workflows (2/2).

I.7 Development and Automation

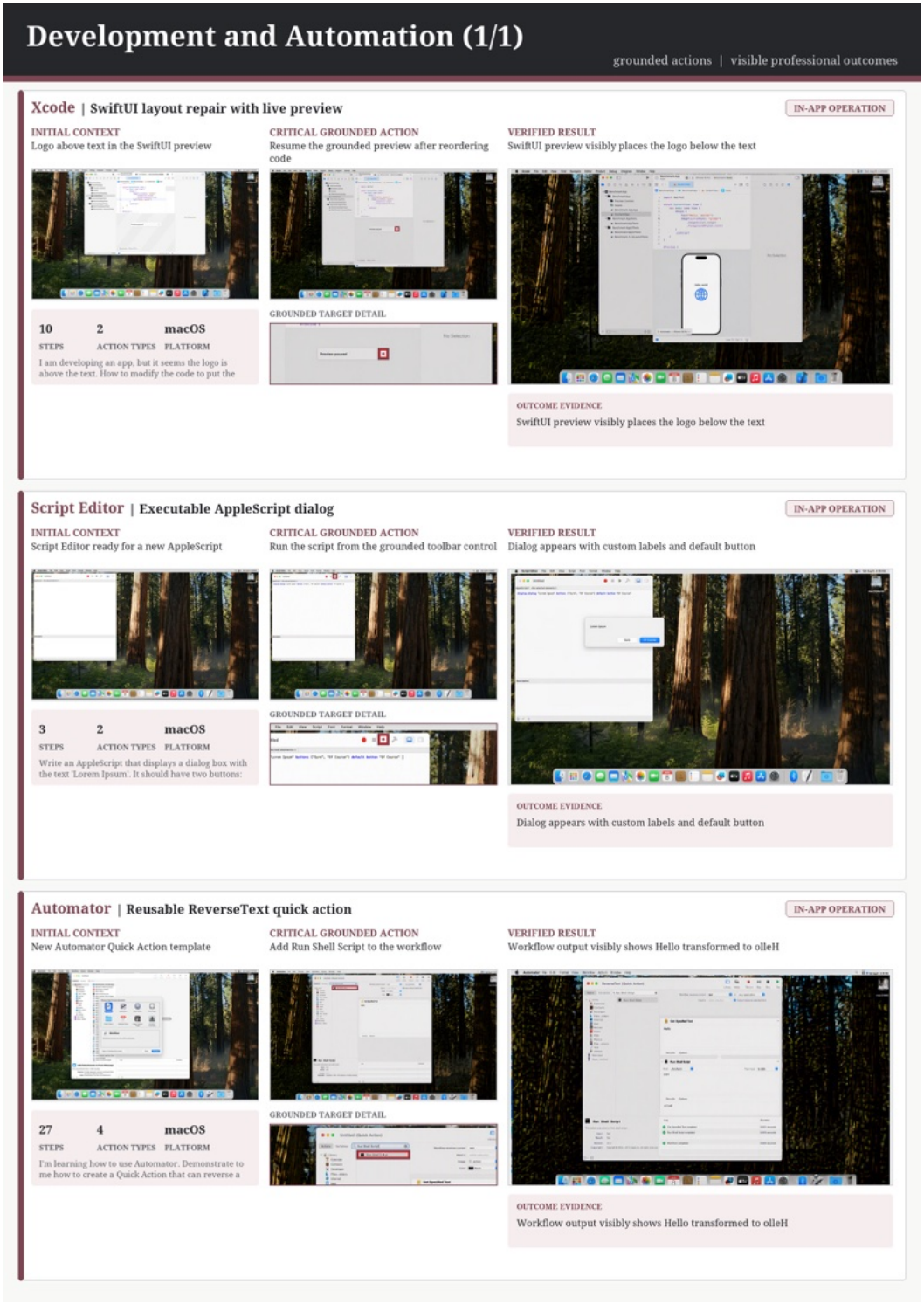

Figure 43: Development and automation workflows.

J QUALITATIVE SEED EXAMPLES

These galleries show the initial screenshots used to condition task generation. Each plate retains the original full-screen views and example labels; the trajectory galleries in Appendix H show subsequent actions and outcomes.

J.1 UBUNTU DESKTOP

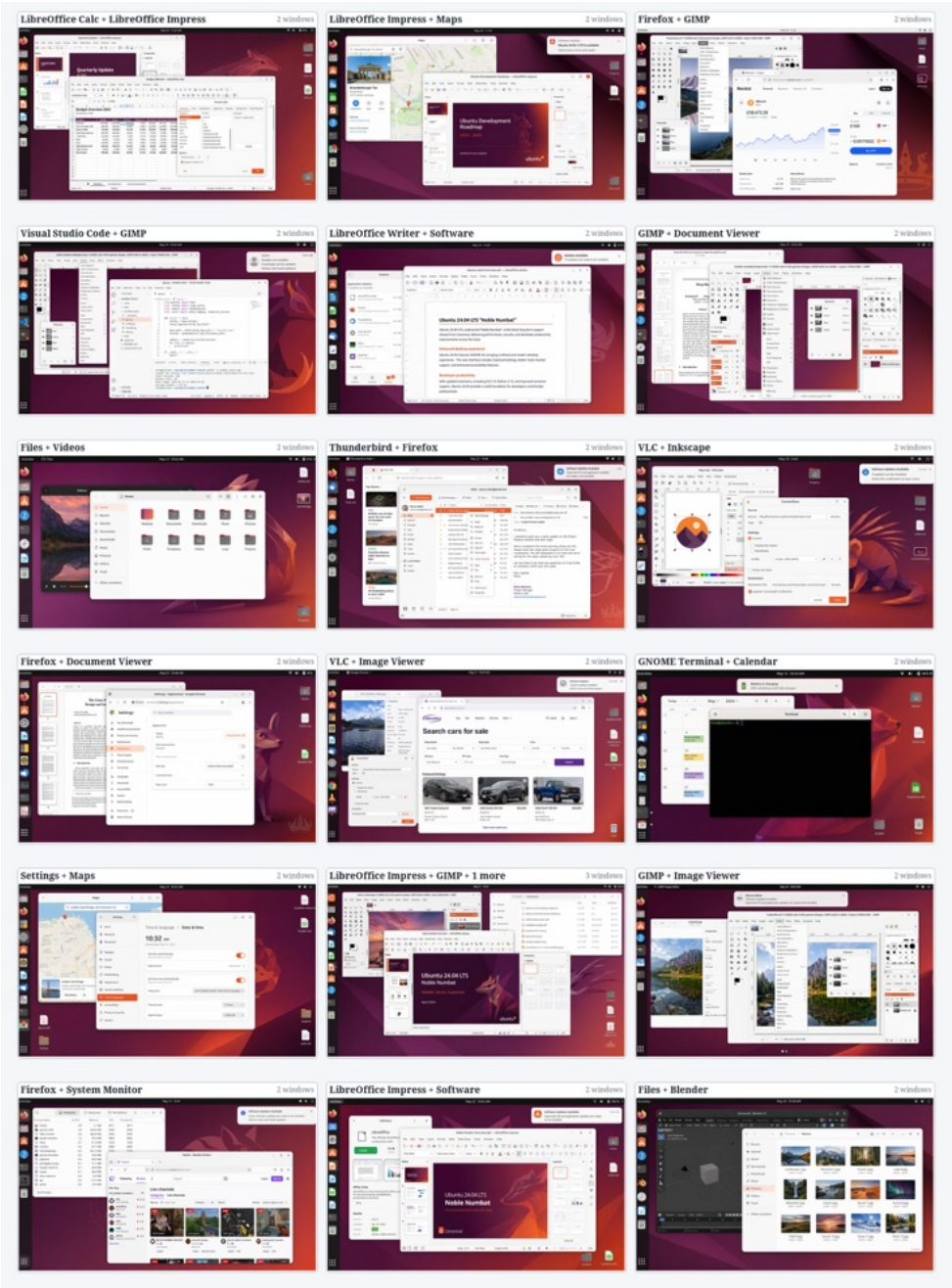

Figure 44: Ubuntu desktop seed screenshots (1/3).

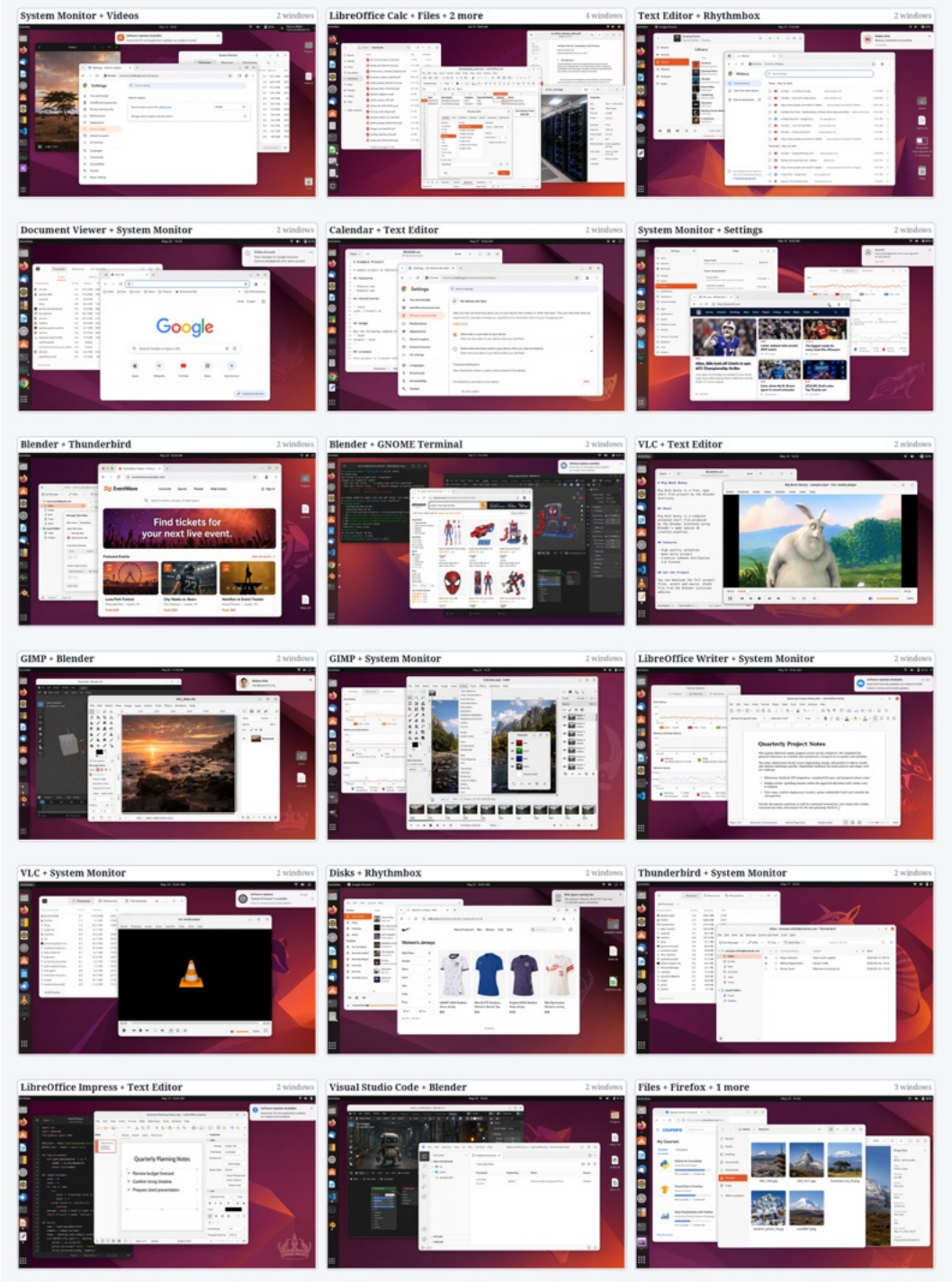

Figure 45: Ubuntu desktop seed screenshots (2/3).

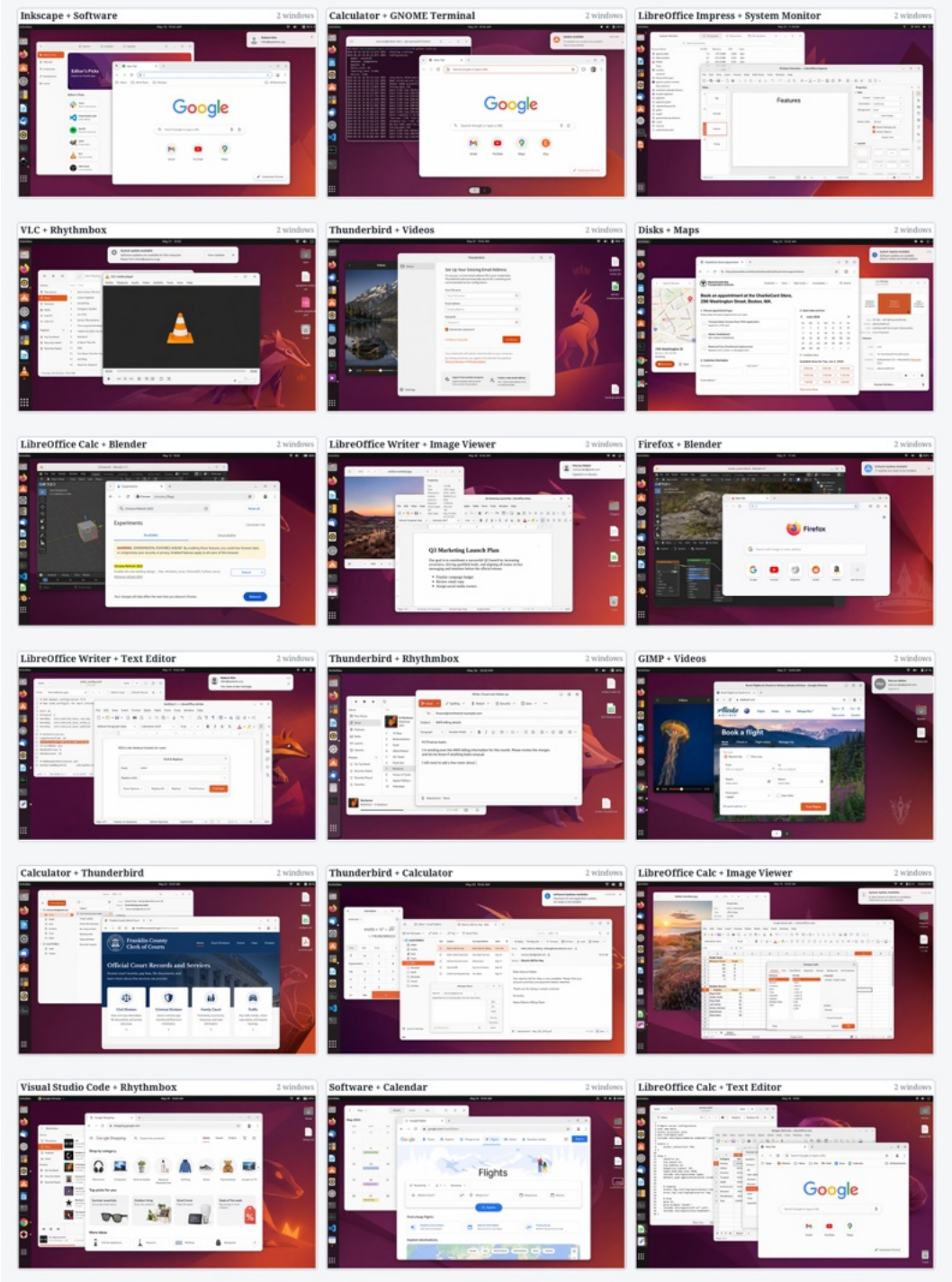

Figure 46: Ubuntu desktop seed screenshots (3/3).

J.2 WINDOWS 11

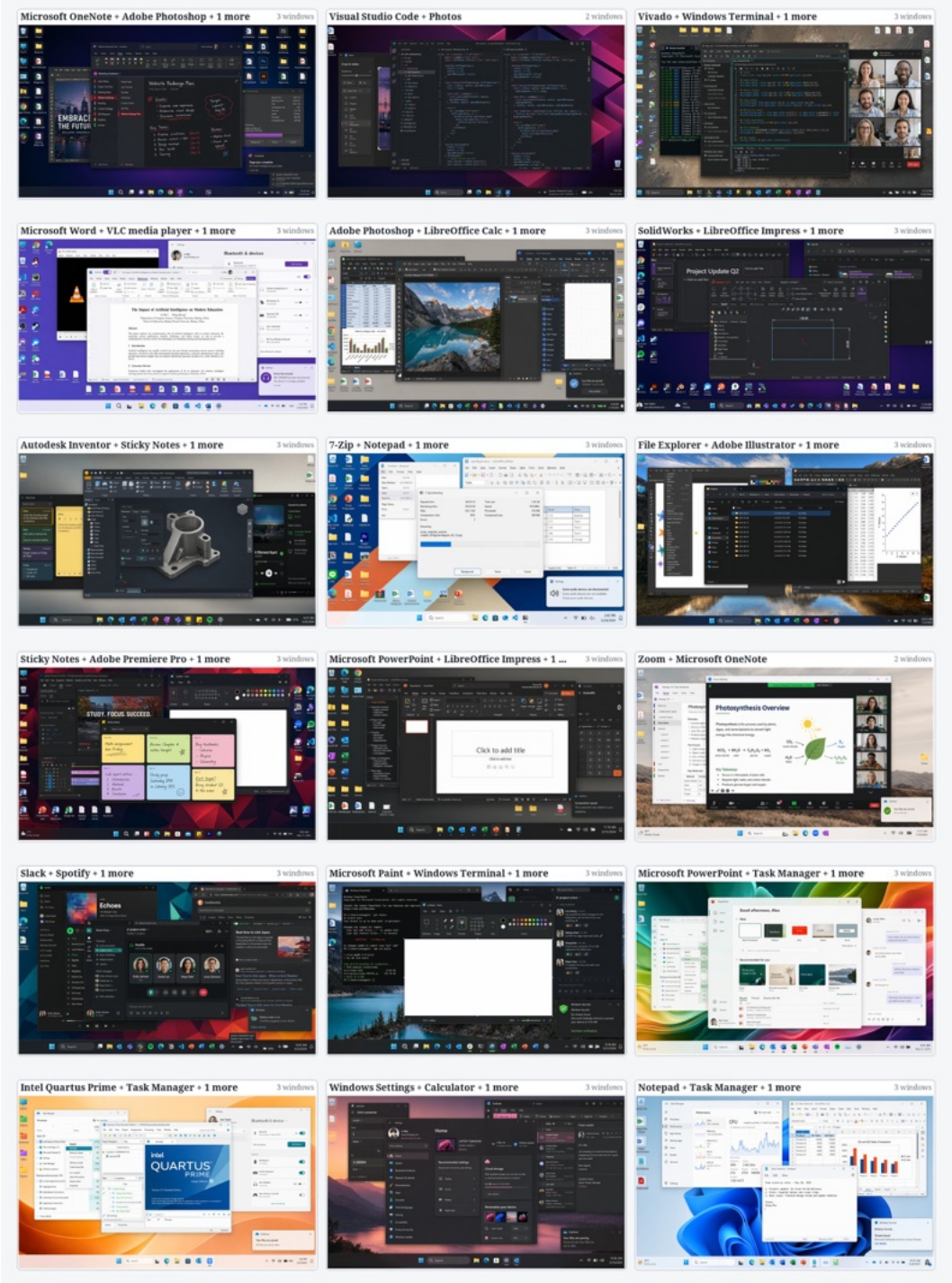

Figure 47: Windows 11 seed screenshots (1/3).

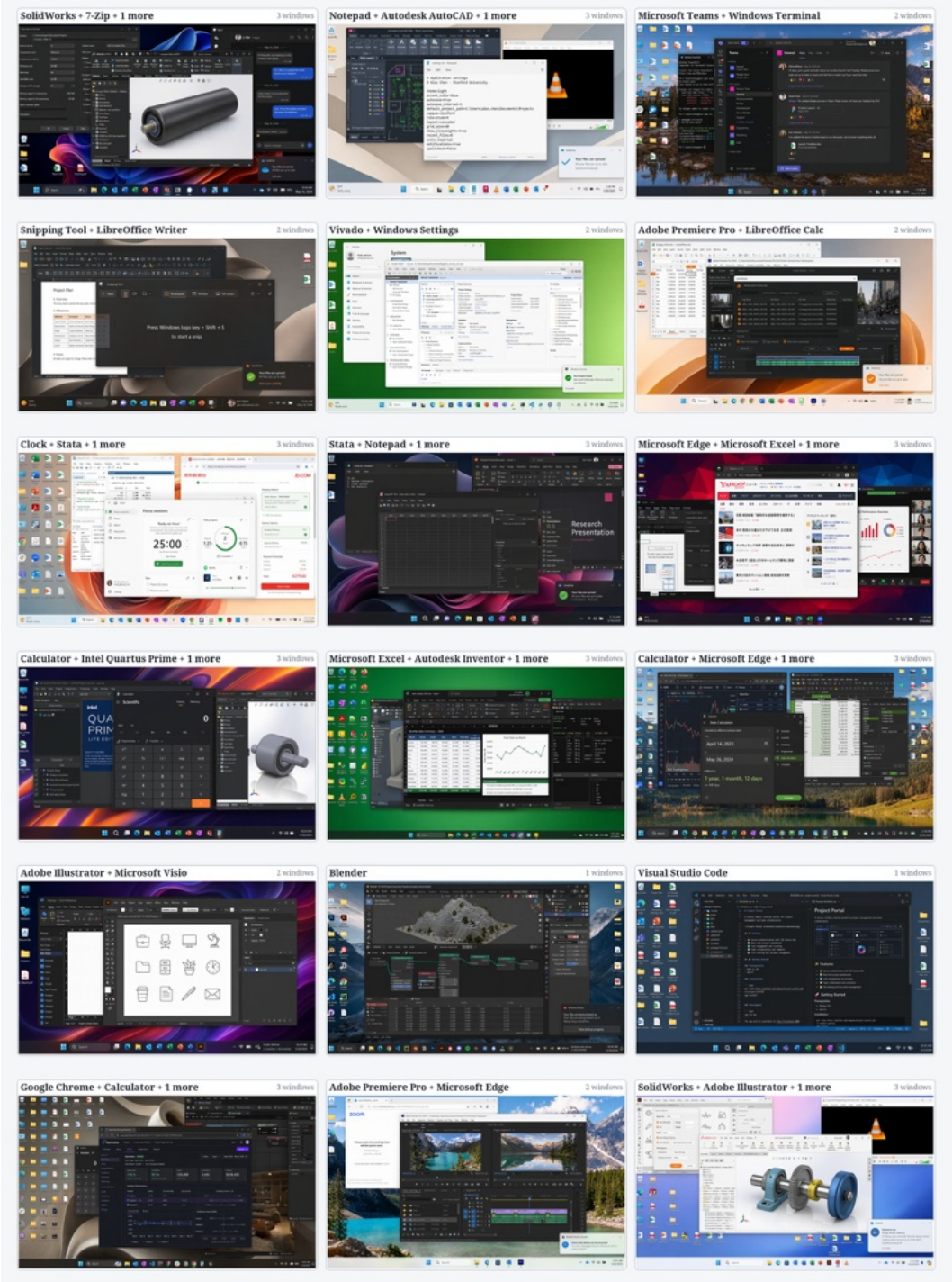

Figure 48: Windows 11 seed screenshots (2/3).

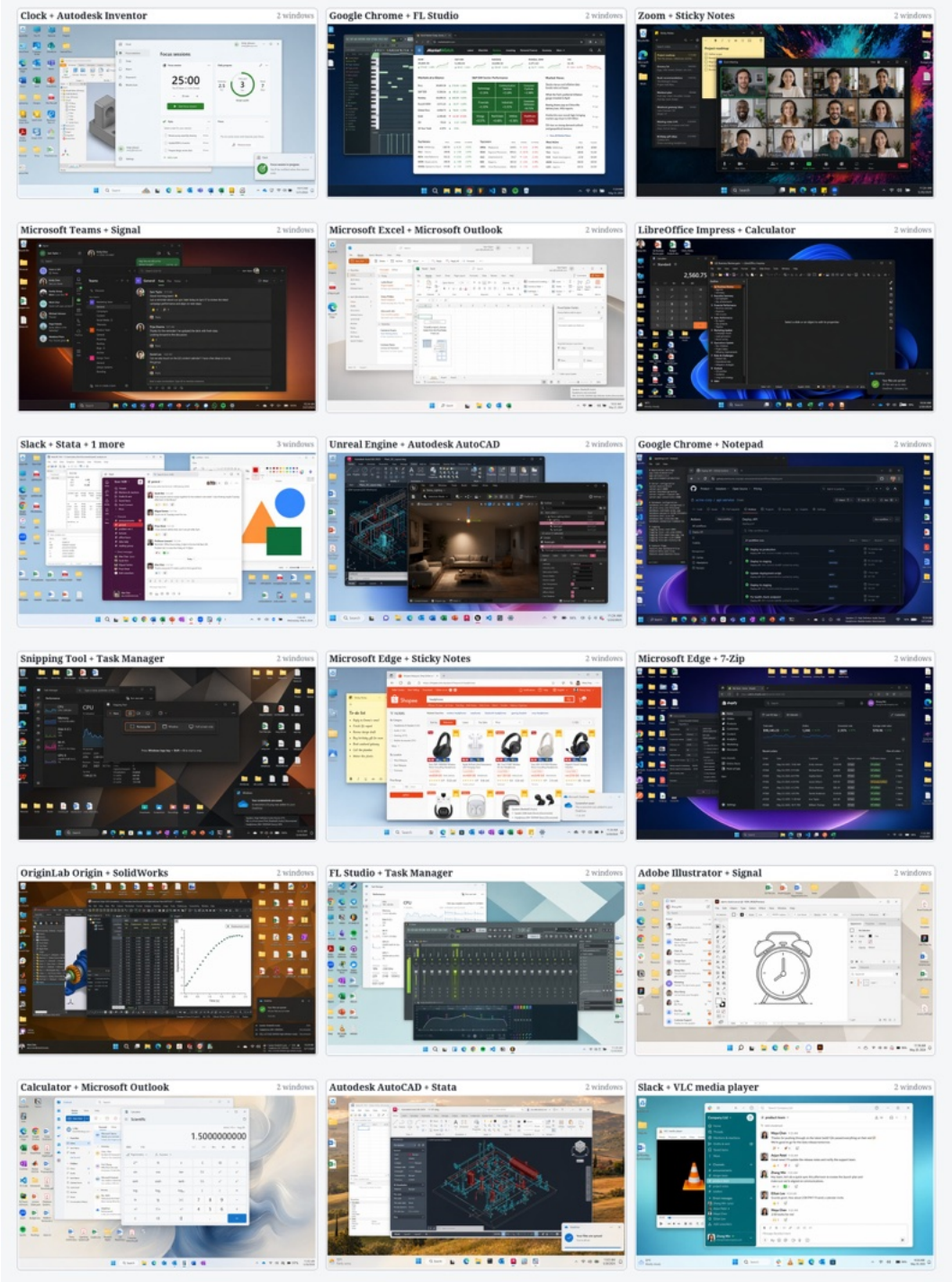

Figure 49: Windows 11 seed screenshots (3/3).

J.3 MACOS

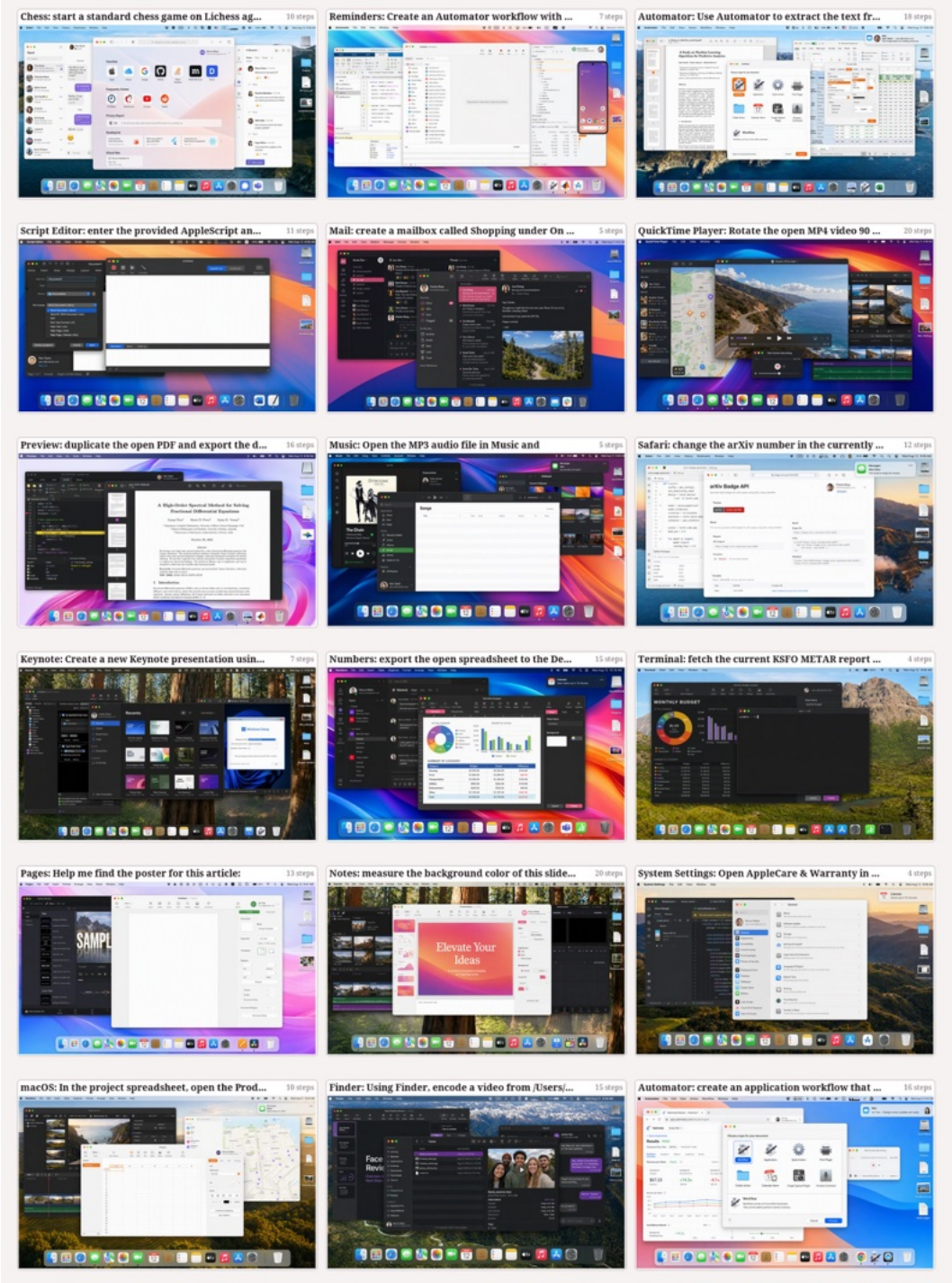

Figure 50: macOS seed screenshots (1/3).

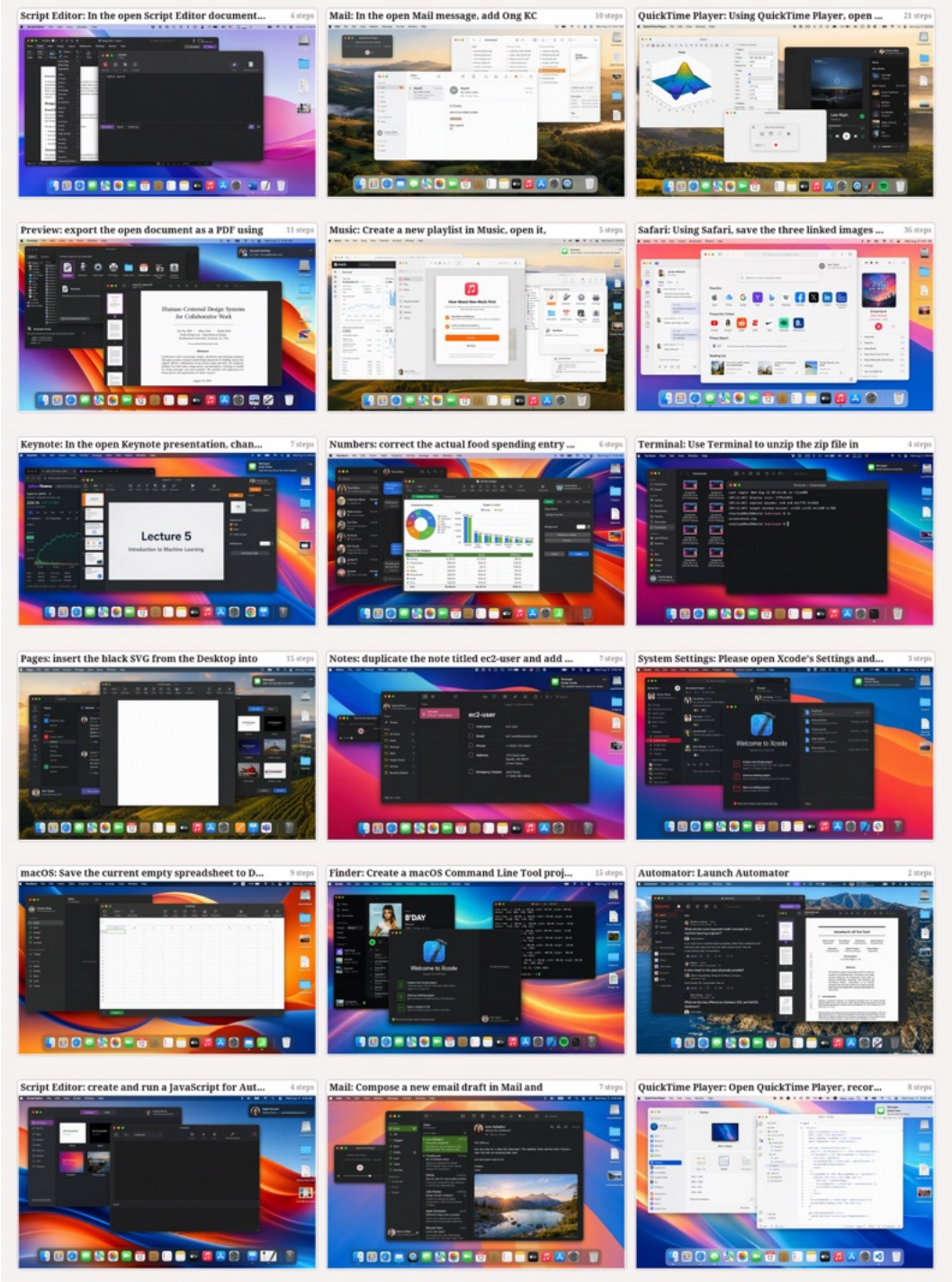

Figure 51: macOS seed screenshots (2/3).

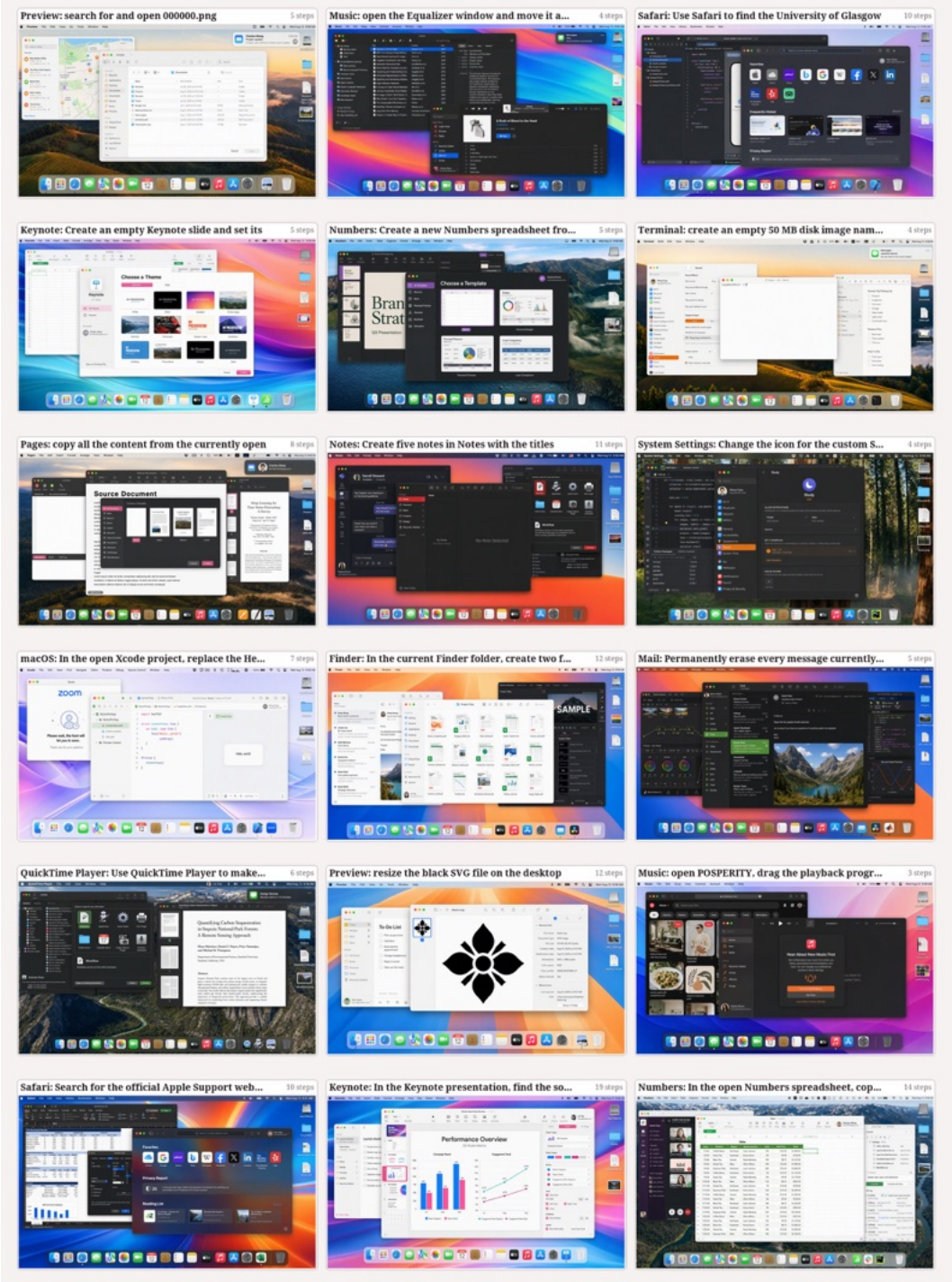

Figure 52: macOS seed screenshots (3/3).

J.4 CHROME

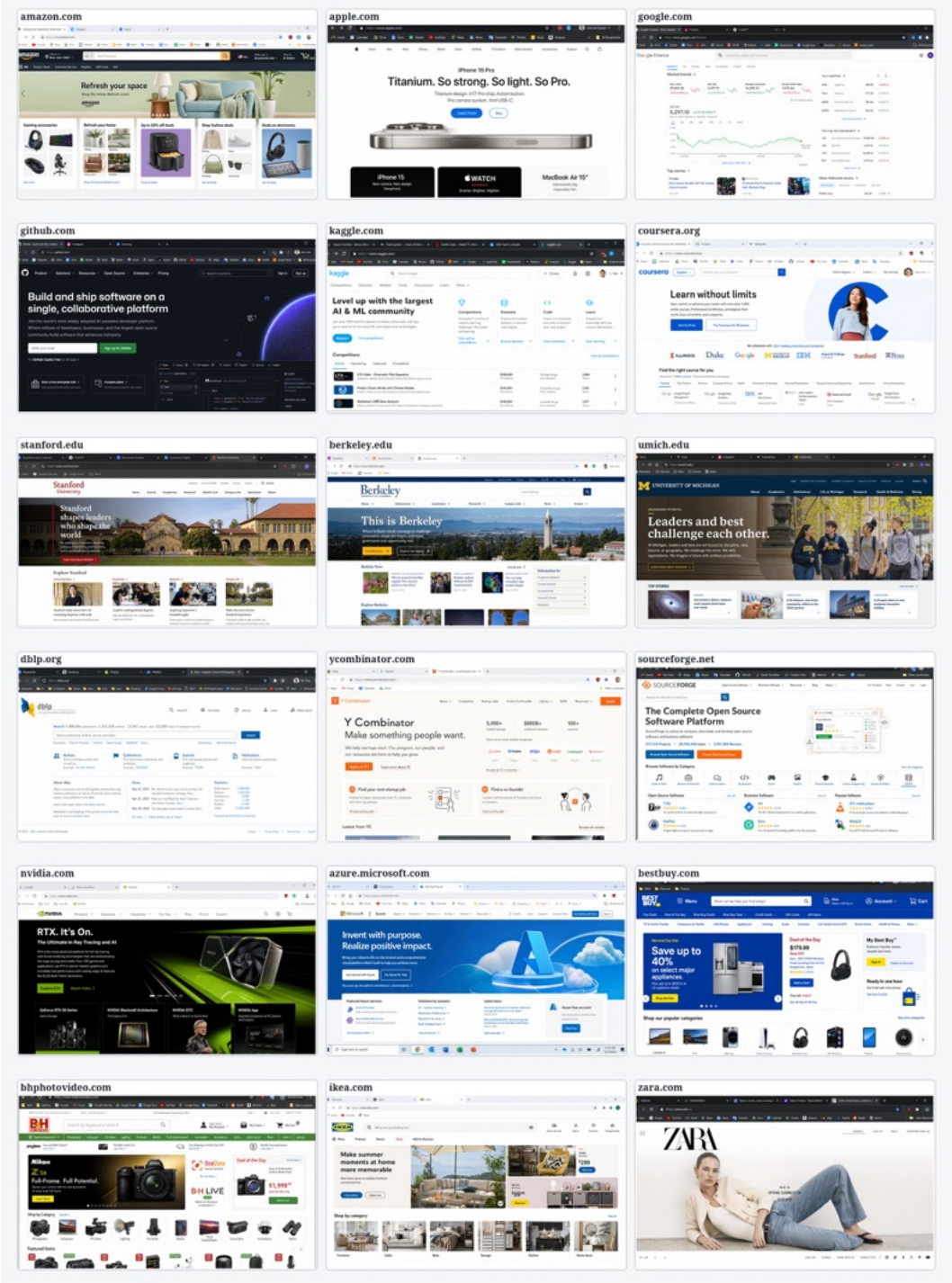

Figure 53: Chrome seed screenshots (1/3).

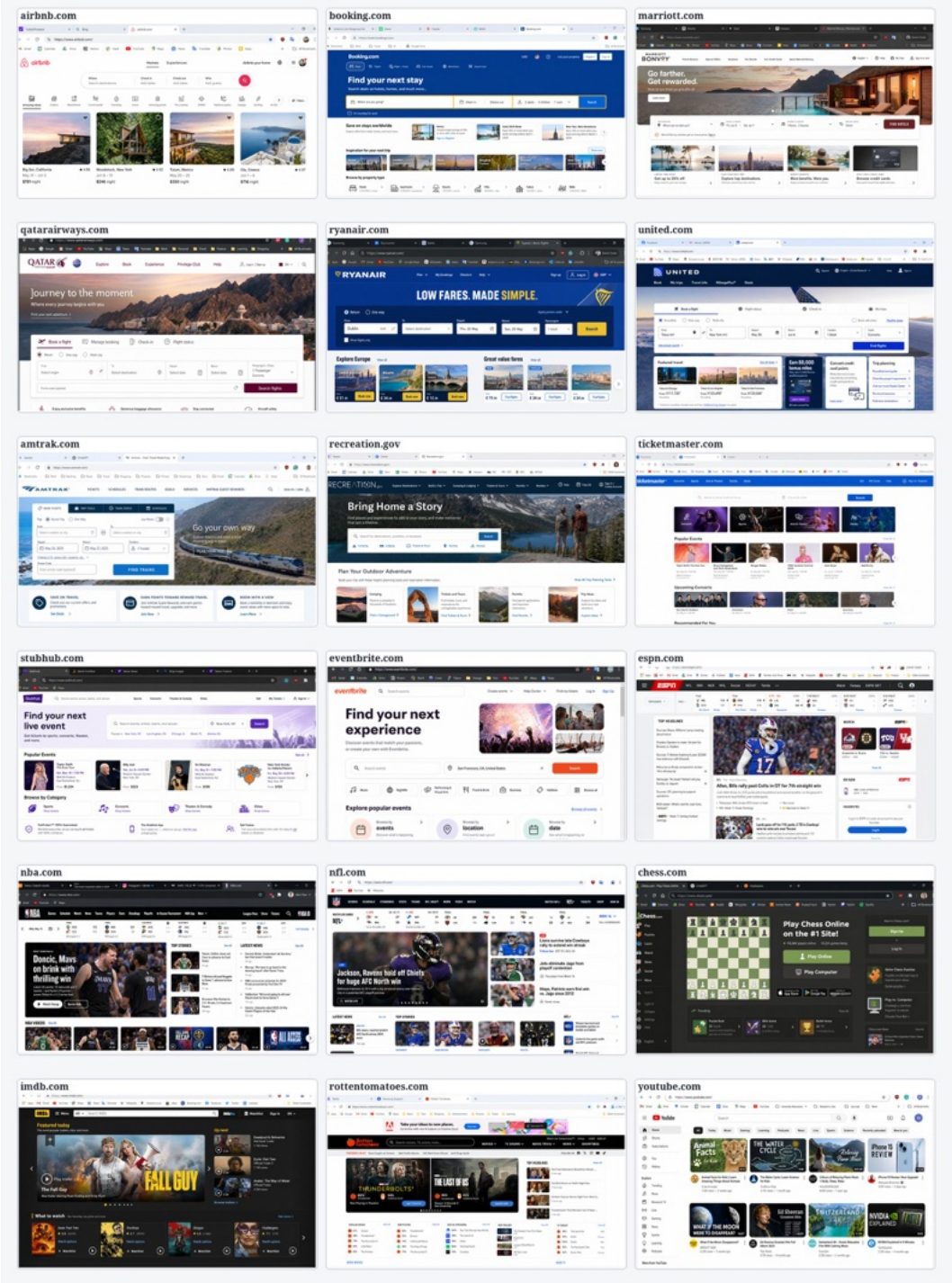

Figure 54: Chrome seed screenshots (2/3).

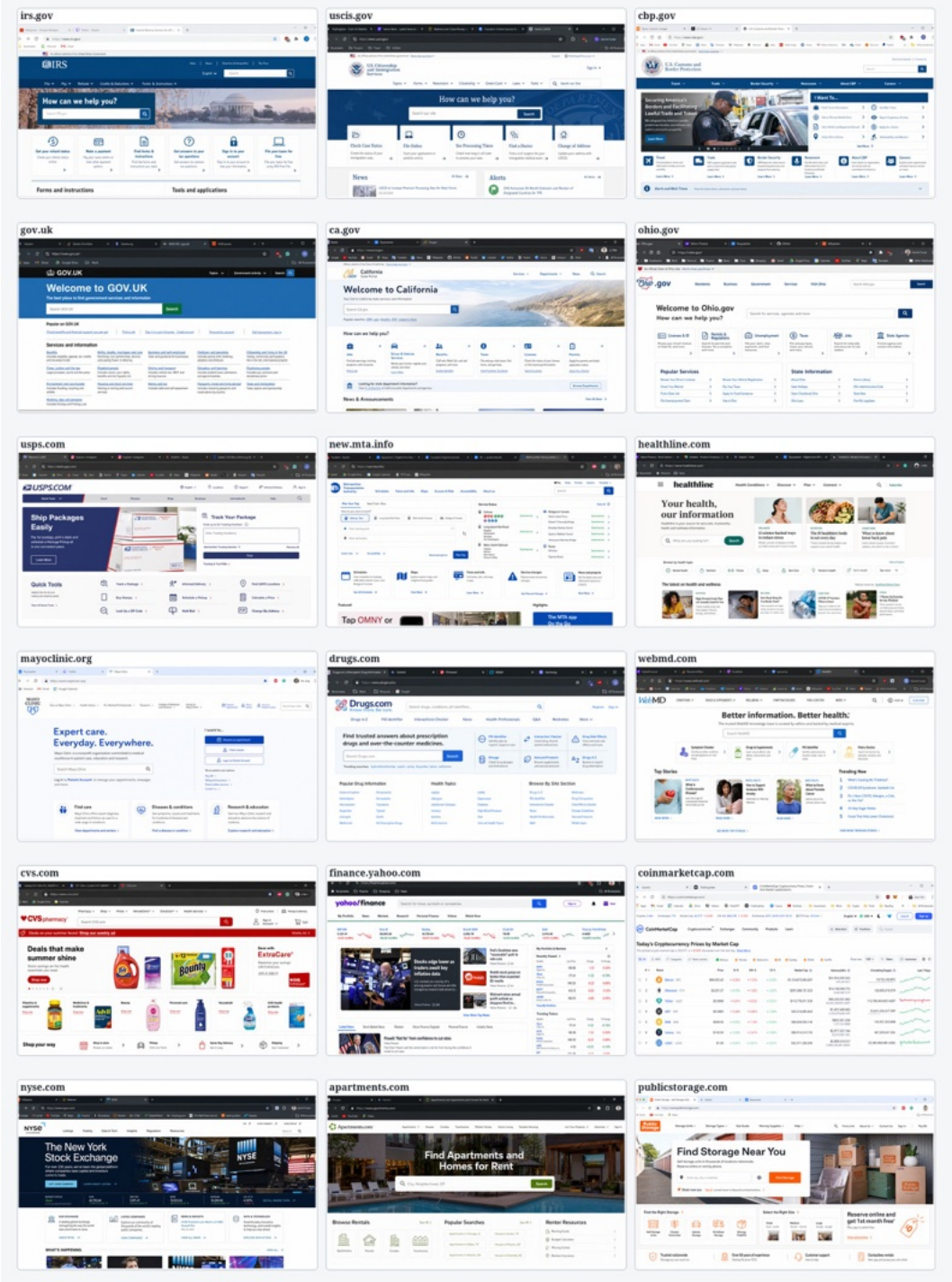

Figure 55: Chrome seed screenshots (3/3).