# DRAGFORGE: PROCEDURAL EVALUATION OF DRAG FOR GUI AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Drag actions combine object selection, a cursor trajectory, and release in a valid state. Endpoint grounding alone cannot evaluate interactions whose outcome depends on the intervening path. We introduce DragForge, a procedural benchmark with 18 task families and 60 configurations spanning placement, path-constrained manipulation, shape production, selection, and dynamic control. Policies observe screenshots and execute bounded waypoint sequences; evaluators measure the resulting task state. Privileged references solve all 180 instances used in the policy evaluation. In paired experiments on 117 development and 117 evaluation instances, replacing successful paths with straight segments while preserving their endpoints retains all 45 placement successes but only one of 72 successes in path-dependent families, in both splits. We evaluate five general-purpose screenshot-conditioned vision-language policies on the same 180 instances under one protocol: four open-weight models and Claude Opus 5.5. Family-balanced task quality ranges from 6.8 to 70.1 on a 0–100 scale. Among the four open-weight models, quality rises descriptively with size, from 6.8 for a dense 27B model that scores 0.8 on placement to 60.8 for a 2.8T-parameter model, although size is not isolated from training. The strongest policy, Claude Opus 5.5, solves every placement family but scores 78.4 on geometry and 32.0 on dynamic control. Recorded static prefixes improve with early calls and saturate as episodes terminate under native limits. In track, maze and curve tracing, replayed failures of the stronger policies usually press and release at the correct locations but violate the path, although this replay does not separate execution from perception of path geometry. DragForge supplies executable tasks, state-based scoring, and controlled trajectory interventions for studying drag behavior beyond endpoint prediction.

## 1 INTRODUCTION

Dragging a file into a folder and drawing a selection around several objects both begin with a press and end with a release. Their requirements between those events differ. For placement, many paths lead to the same result. For a corridor, trace, or enclosing region, intermediate positions determine whether the action works. A GUI agent can therefore predict plausible endpoints without producing a successful interaction, yet GUI agents are mostly trained and evaluated on exactly this kind of point prediction (Cheng et al., 2024; Gou et al., 2025; Wu et al., 2025; Li et al., 2025).

Measuring this difference requires executing the action. Recent drag benchmarks score source and target regions or distance to a recorded demonstration (Bout et al., 2026; Liao et al., 2026; Hu et al., 2026). Distance to a demonstration measures agreement with that path, but an interface may admit several valid paths or reject a nearby path crossing a boundary. The resulting task state provides a complementary measure, as in execution-based agent benchmarks (Zhou et al., 2024; Xie et al., 2024): whether the object was placed, the region selected, or the required shape produced.

DragForge makes these conditions explicit in procedurally generated browser environments. First, it provides 18 families with executable success predicates and bounded task-quality measures under a waypoint action interface. Second, reference controllers and endpoint-preserving interventions test feasibility and establish when interior path geometry matters. Third, it evaluates five screenshot-conditioned policies on the same instances, action interface and scoring, from a dense 27B open-weight model to a proprietary frontier model (Qwen Team, 2026; MiniMax, 2026; Kimi Team,



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.

2026a;b; Anthropic, 2026), with per-episode evidence and family-level analysis. The benchmark is intended for controlled diagnosis of drag behavior; transfer to full desktop workflows remains an open question.

## 2 Related work

**Interactive agent benchmarks.** WebArena, VisualWebArena, WorkArena, OSWorld and AndroidWorld evaluate task completion in realistic web, desktop and mobile environments (Zhou et al., 2024; Koh et al., 2024; Drouin et al., 2024; Xie et al., 2024; Rawles et al., 2025), while Mind2Web scores actions against recorded demonstrations (Deng et al., 2023). BrowserGym provides a common interface across web-agent benchmarks (Le Sellier de Chezelles et al., 2025). DragForge adopts execution-based evaluation but narrows the scope to cursor trajectories. MiniWoB++ already contains drag environments (Shi et al., 2017; Liu et al., 2018), and pixel-based agents have been trained on the suite (Shaw et al., 2023); our contribution is a broader procedural suite with common budgets, state-based measures and trajectory interventions. As in Procgen (Cobbe et al., 2020), procedural generation separates development instances from evaluation instances.

**GUI grounding.** Screenshot-conditioned agents map pixels to actions, and recent models are trained for element grounding and multi-step interaction (Hong et al., 2024; Cheng et al., 2024; Gou et al., 2025; Wu et al., 2025; Lin et al., 2025; Xu et al., 2025; Qin et al., 2025). Grounding benchmarks such as ScreenSpot-Pro score whether a predicted point falls inside a target element (Li et al., 2025). These measures concern a single point and do not test the path between press and release, the variable DragForge controls.

**Drag interactions.** DragOn evaluates source and target grounding across four domains (Bout et al., 2026). GUI-Drag and its text-dragging ScreenDrag benchmark study action triggering and text-span grounding (Liao et al., 2026). ShowUI-$\pi$ introduces a benchmark also named ScreenDrag and studies continuous drag trajectories (Hu et al., 2026). Its online protocol approximates transitions using retrieved recorded states; DragForge instead executes actions against procedural environment state. DragForge differs in two ways: it scores the resulting state across heterogeneous tasks, and it changes intermediate geometry while holding action endpoints fixed. We do not assume that a single demonstrated trajectory defines valid actions.

Table 1 distinguishes the evaluation object in related work. Endpoint-region correctness, agreement with a demonstrated trajectory, and resulting task state answer different questions. DragForge focuses on the last, while exposing the trajectory as a controlled experimental variable. Its synthetic scope makes exact reset and intervention practical, but sacrifices the visual diversity and workflow realism of application-derived tasks. In human–computer interaction, Fitts' law models pointing time and the steering law models movement through constrained paths (MacKenzie, 1992; Accot & Zhai, 1997). Our corridor and tracing families impose steering-like constraints, but we score task outcomes rather than movement time.

Table 1: Complementary evaluation targets in drag benchmarks. The comparison concerns reported evaluation protocols.

| Benchmark | Evaluation object | Main measurements |
|---|---|---|
| DragOn (Bout et al., 2026) | Source and target grounding across four interaction domains | Region-based success and spatial-tolerance accuracy |
| ScreenDrag / ShowUI-$\pi$ (Hu et al., 2026) | Continuous trajectories across five application domains | Offline trajectory error and endpoint accuracy; data-driven online rollout success |
| DragForge | State transitions across 18 procedural task families | Task quality, strict completion, reference calibration, and endpoint-preserving interventions |

| Task demand | Families | Configs. |
|---|---|---|
| Assignment | File sorting, luggage, funnel, pharmacy, blood bank | 15 |
| Constrained transport | Track following, maze | 12 |
| Shape reproduction | Glyph drawing | 3 |
| Coverage | Painting, curve tracing | 6 |
| Region selection | Lasso | 3 |
| Sequential puzzle | Water sorting | 3 |
| Dynamic control | Flappy, ping-pong, volley, fruit slicing, hoop flicking, free kick | 18 |
| Total | 18 families | 60 |

Table 2: Benchmark scope. Families differ in what makes a drag useful: destination, intermediate geometry, selected region, action sequence, or interaction with moving state.

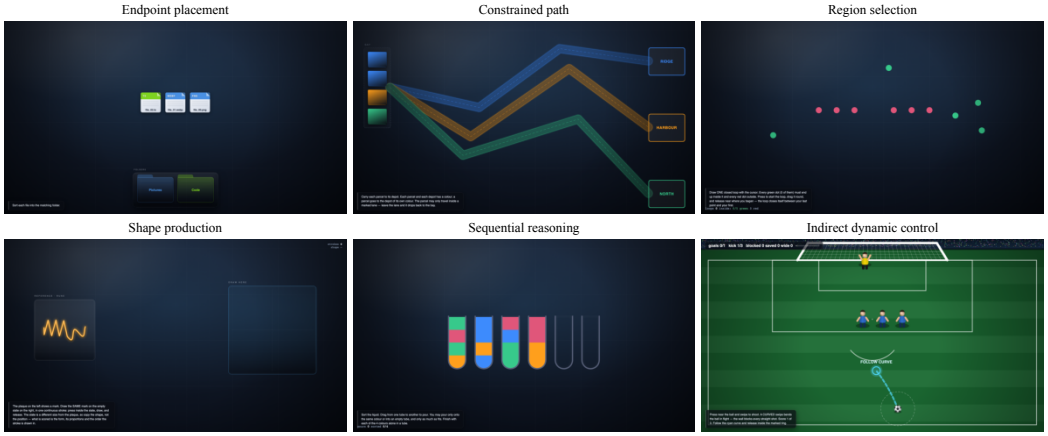


Figure 1: Initial observations from six families on an instance outside all reported splits: placement, corridor following, region selection, shape reproduction, sequential pouring, and dynamic control.

# 3 The DragForge benchmark

## 3.1 Task families and procedural instances

The suite contains twelve static and six dynamic families (Table 2). Most families have three configurations; track following has nine, giving 60 configurations. A configuration fixes generator parameters and, for dynamic tasks, a planning cadence. A seed determines an instance. Figure 1 illustrates six interfaces.

**Instance splits.** Every configuration has three disjoint sets of instances. Three *development* instances per configuration served for pilot runs, metric design and the audits in Section 4. Five *feasibility* instances per configuration are used only to test reference controllers. Three *evaluation* instances per configuration, 180 in total, are used for the policy study and its normalization anchors. Seed values appear in Appendix A.3.

## 3.2 Observations and executable actions

A policy receives a screenshot, task brief, viewport dimensions, and planning horizon. Each request omits earlier conversational turns. The evaluator inspects privileged state to score episodes but withholds it from the policy. The environment viewport is $1920\times1080$; image processing appears in Appendix A.1.

An action contains at most 64 cursor waypoints,

$$a = ((t_i, x_i, y_i, b_i))_{i=1}^{K}, \qquad K \leq 64, \tag{1}$$

where $t_i$ is a relative simulation tick, $(x_i, y_i)$ is a pixel position, and $b_i$ is a button state. Static execution presses at the first point, traverses the path, and releases at the last. Interior button states do not create separate drags. Every vertex is preserved and intermediate mouse events limit adjacent positions to four pixels, independently of waypoint timing; in a single-seed check, reducing the bound to 16 waypoints barely changes the strongest policy (Appendix A.7). The parser can append a terminal release pose.

Dynamic families instead apply tick-wise cursor updates through a simulation interface. Simulation time pauses during inference, so API latency does not consume the simulated control horizon.

Static tasks allow two decisions per reported object, with explicit family overrides and a cap of 60. Dynamic tasks use configuration-specific cadences and at most 60 decisions. Inert phases can advance without another model decision. An episode ends at success, a terminal state, or exhausting its decision or simulation budget. All configurations and predicates appear in the appendix.

### 3.3 What a screenshot reveals

A source and logged-prompt audit distinguishes rendered cues from privileged simulator state (Table 7 in Appendix A.1). Several dynamic tasks expose recent motion within a single image: flappy draws a velocity-dependent tail, volley draws ball history and a hand-velocity vector, and ping-pong draws ball history and an incoming-interception marker. Fruit slicing instead draws the blade history, without a projectile-position trail. Numeric simulator parameters are not supplied to the policy. Task briefs state each family's rules, for example that the defensive wall blocks straight free kicks, but no brief contains coordinates or a path. These cues change what a policy must infer before choosing a trajectory.

Some conditions provide strong guidance or additional rules. Easy hoop and free-kick tasks display solved action guides, so their scores do not measure the same unaided physical inference as harder settings. Free kick imposes a minimum-curvature rule in easy; a rejected straight shot need not have collided with a drawn defender. Non-actionable flight phases advance without new model decisions.

### 3.4 Task quality and aggregation

We measure bounded task quality $q \in [0, 1]$ from the resulting state. Count-based tasks use completed-subgoal fractions; painting and tracing use mask intersection-over-union (IoU); lasso uses target-set IoU; glyph uses a tolerance-scaled shape score with validity checks. Maze retains binary completion because its available distance proxy is not validated as geodesic progress. Appendix A.10 gives the exact definitions. These measures credit partial accomplishment; we report strict success as a supplementary outcome.

To avoid giving track's nine configurations three times the influence of an ordinary family, we average configurations within each family and then weight families equally:

$$Q_m = \frac{100}{|F|} \sum_{f \in F} \frac{1}{|C_f|} \sum_{c \in C_f} \bar{q}_{m,f,c}. \tag{2}$$

Here $\bar{q}_{m,f,c}$ is the mean task quality of model $m$ on configuration $c$ of family $f$, averaged over its three evaluation instances. We also report a reference-normalized score, following the affine calibration convention used in D4RL (Fu et al., 2020):

$$N_{m,f,c} = 100 \frac{\bar{q}_{m,f,c} - \bar{q}_{\text{random},f,c}}{\bar{q}_{\text{reference},f,c} - \bar{q}_{\text{random},f,c}}, \qquad N_m = \frac{1}{|F|} \sum_{f \in F} \frac{1}{|C_f|} \sum_{c \in C_f} N_{m,f,c}. \tag{3}$$

Random corresponds to zero and the privileged reference to 100; the reference is not a proven optimum. Normalized scores are unclipped and may be negative or exceed 100. Small reference–random gaps make this measure sensitive to anchor estimation, so we present it alongside $Q_m$ rather than as the sole summary.

Quality definitions were developed on pilot rollouts on development instances and frozen before the evaluation runs. Every policy is scored with the same definitions, configuration weights and instance weights.

# 4 ENVIRONMENT VALIDATION

## 4.1 FEASIBILITY AND PREDICATE CHECKS

Privileged reference controllers solve all 180 evaluation instances within the episode decision limits and the 64-waypoint action bound. Static controllers produce bounded paths, water sorting uses a dedicated solver, and dynamic controllers read simulation state. The same controllers also solve all 180 development instances and all 300 feasibility instances, 660 of 660 in total. Because the controllers read state that policies never see, this shows feasibility, not that every instance can be solved from screenshots.

Two further checks on the development instances test the success predicates. With no input, no instance succeeds, even when dynamic environments run through their full simulation horizon; this is a check of the predicates, not a budget-matched baseline. A random controller that samples press and release positions uniformly succeeds on four of the 180 instances, all in dynamic families (Appendix A.3). On the evaluation instances, the same random controller succeeds on two of 180 instances, both easy hoop flick. These evaluation reference and random outcomes are the normalization anchors of Section 3.4.

## 4.2 BEHAVIORAL CHECKS OF TASK QUALITY

A four-condition audit covers all 60 configurations at each of the three development seeds: no input, a reference-action prefix, the full reference, and its horizontal reflection. All 180 full-reference replays reproduce their recorded scores and success predicates. No no-input episode succeeds, although nine dynamic cases reach quality between 0.14 and 0.5 as the environment evolves. Among the 135 shortenable reference sequences, 110 prefixes produce intermediate quality, six produce zero, and 19 retain full quality; none exceeds its full reference. Quality therefore registers partial progress that strict completion misses.

Twelve geometry stress cases test excessive coverage, distractor selection, and invalid shape production. None succeeds and all score below the matched reference. Nonzero overlap can still earn partial quality: selecting every lasso point scores 45.5–55.6 because target-set IoU penalizes, rather than discards, false positives. These checks do not establish general exploit resistance or agreement with human judgments. Full procedures and native diagnostics appear in Appendix A.4.

## 4.3 SENSITIVITY TO PATH GEOMETRY

To test whether task outcomes depend on the path rather than its endpoints, we replay successful reference sequences on the 39 static configurations outside water sorting, first on their 117 development instances and then, as a confirmatory replication, on their 117 evaluation instances. We replace each path by a straight segment or perturb its interior positions while preserving the initial press and final release. The subset comprises five placement and six path-dependent families.

On both splits, replacing the path with a straight segment preserves all 45 placement successes but only one of 72 path-dependent successes (Table 3). Small perturbations preserve all but at most one success; larger perturbations retain 15 of 72 path-dependent successes on both splits. Path-dependent families therefore test trajectory execution and not only endpoint placement. Section 5.2 shows that failures of the stronger policies are consistent with this dependence. The perturbation procedure and family-level counts appear in Appendix A.8.

# 5 SCREENSHOT-CONDITIONED POLICY EVALUATION

We evaluate Claude Opus 5.5 (Anthropic, 2026), Kimi K3 (Kimi Team, 2026b), Kimi K2.6 (Kimi Team, 2026a), MiniMax M3 (MiniMax, 2026) and Qwen3.8-27B (Qwen Team, 2026). All five share the instances, observations, action interface, decoder and scoring. We do not evaluate specialized GUI agents. Their action spaces either predict a single coordinate (Gou et al., 2025; Wu et al., 2025) or specify a drag by its start and end points alone (Qin et al., 2025), so they cannot express the multi-waypoint trajectories that path-dependent families require. The straight-path condition of Table 3 shows what one such drag per object can achieve: with perfect press and release points,

Table 3: Effect of changing the path while keeping both endpoints fixed, on development instances and, as a replication, on the evaluation instances used for the policy study. Entries count completed instances.

| | | Original | Straight | Path perturbation | |
|---|---|---|---|---|---|
| Instances | Task group | path | path | $\pm 5$ px | $\pm 20$ px |
| Development | Placement | 45/45 | 45/45 | 45/45 | 45/45 |
| Development | Path-dependent | 72/72 | 1/72 | 72/72 | 15/72 |
| Evaluation | Placement | 45/45 | 45/45 | 45/45 | 45/45 |
| Evaluation | Path-dependent | 72/72 | 1/72 | 71/72 | 15/72 |

straight drags solve one of 72 path-dependent reference instances; adapting trajectory-generating models such as ShowUI-$\pi$ (Hu et al., 2026) to our interface is left to future work, so our conclusions concern general-purpose policies. All policies receive $960\times540$ screenshots, use the same terminal-action decoder, and have the same 64-waypoint and 60-request limits, 24,000-completion-token cap, 360-second request timeout and medium reasoning effort (Appendix A.1).

The evaluation instances were not used for pilot runs or metric design, but they were used for renderer and reference validation, so they are not a blind holdout for every engineering decision.

Task quality $Q$ and reference-normalized progress $N$ are the primary aggregates, with the same configuration and instance weights for all policies; family-level quality and strict completion supplement them. Invalid or truncated responses are scored from the resulting state, not as zero (Appendix A.10, Table 15).

Table 4: Task quality and normalized progress on the complete evaluation matrix. Each policy uses the same 180 instances, with equal weights for the 18 families. Normalization sets the matched random controller to zero and the privileged reference to 100. Brackets give 95% bootstrap intervals over instances within each configuration; intervals for $N$ hold the reference and random anchors fixed, and Appendix A.9 reports sensitivity to the random anchor. Parameter counts are from the official model cards; n/a marks an undisclosed size. Best value in each column is bold.

| | | Equal-weight mean over 18 families [95% CI] | |
|---|---|---|---|
| Policy | Params (total/active) | Task quality $Q \uparrow$ | Normalized progress $N \uparrow$ |
| Claude Opus 5.5 | n/a | **70.1** [67.4, 72.9] | **70.6** [67.7, 73.5] |
| Kimi K3 | 2.8T/104B | 60.8 [58.0, 63.5] | 60.4 [57.6, 63.2] |
| Kimi K2.6 | 1T/32B | 32.7 [30.2, 35.2] | 31.4 [28.8, 33.9] |
| MiniMax M3 | 428B/23B | 29.4 [26.8, 31.9] | 27.6 [25.0, 30.3] |
| Qwen3.8-27B | 27B/27B (dense) | 6.8 [4.6, 9.1] | 2.6 [$-0.7$, 6.5] |

Claude Opus 5.5 has the highest family-balanced quality, 70.1, followed by Kimi K3 at 60.8, Kimi K2.6 at 32.7, MiniMax M3 at 29.4 and Qwen3.8-27B at 6.8. Reference normalization gives the same ordering, with scores of 70.6, 60.4, 31.4, 27.6 and 2.6. Every value uses all 180 instances per policy. Bootstrap 95% intervals, which resample instances within each configuration, have half-widths of 2.2 to 2.8 points in $Q$. Paired intervals separate Claude Opus 5.5 from Kimi K3 (9.4 points, 95% CI [4.9, 13.7]), Kimi K3 from Kimi K2.6 and MiniMax M3 from Qwen3.8-27B, with two-sided bootstrap $p < 0.001$ each, whereas the 3.3-point lead of Kimi K2.6 over MiniMax M3 is marginal (95% CI [0.0, 6.6], $p = 0.05$), at the boundary under a Holm correction over these four comparisons. Qwen3.8-27B is close to the random controller on $N$ (2.6; 5.4 with 30-draw anchors, Appendix A.9). We treat only these adjacent comparisons as inferential; family-level differences are descriptive. The reference has raw task quality 98.3 on these layouts, so even the strongest policy is far from it.

**Placement and path execution.** Claude Opus 5.5 scores 100 on placement, 78.4 on geometry and 32.0 on dynamic control, and Kimi K3 scores 98.6, 57.1 and 30.1. MiniMax M3 and Kimi K2.6 show a larger placement–geometry gap: 73.5 versus 11.7, and 72.5 versus 16.9. The groups differ in

| | Task quality $Q \uparrow$ by task demand | | | |
|---|---|---|---|---|
| Policy | Placement | Geometry | Puzzle | Dynamic |
| Claude Opus 5.5 | 100.0 | 78.4 | 100.0 | 32.0 |
| Kimi K3 | 98.6 | 57.1 | 77.8 | 30.1 |
| Kimi K2.6 | 72.5 | 16.9 | 25.7 | 16.4 |
| MiniMax M3 | 73.5 | 11.7 | 0.0 | 15.2 |
| Qwen3.8-27B | 0.8 | 5.9 | 0.0 | 13.9 |

Table 5: **Quality across task demands.** Each column averages families within that group. The overall score additionally weights groups by their number of families: five placement (assignment), six geometry (the path-dependent families of Table 3), one puzzle, and six dynamic families.

more than their trajectory requirements, so this comparison is descriptive and does not isolate motor precision. The dynamic group includes easy configurations with solved action guides. Together with the endpoint-preserving intervention, it shows that placement performance is a poor proxy for path execution.

**Family profiles.** Claude Opus 5.5 reaches 100 on all placement families, lasso and water sorting, but only 10.5 on flappy and 62.1 on curve tracing, below Kimi K3 on flappy and fruit slicing. Kimi K3 scores 11.1 on maze and zero on ping-pong, and the three weaker policies score zero on track and maze. Qwen3.8-27B earns more quality in dynamic control than in placement. Most cells have nine episodes and wide intervals (Table 6).

**Scale among open-weight models.** Ordered by size, Qwen3.8-27B (27B dense), MiniMax M3 (428B total, 23B active), Kimi K2.6 (1T, 32B active) and Kimi K3 (2.8T, 104B active) score 6.8, 29.4, 32.7 and 60.8; by active parameters, only MiniMax M3 (23B) and the dense 27B model swap. The gap opens before path geometry matters: the 27B model scores 0.8 on placement, where every larger model exceeds 72. Kimi K3 completes 93 of 180 episodes, while no smaller open-weight model completes more than 25. Claude Opus 5.5 scores highest, but its size is undisclosed, so we do not place it on this axis. Four models from three developers cannot separate size from training data or architecture, so this trend is descriptive.

### 5.1 Progress under additional policy calls

We truncate the recorded trajectories at fixed policy-call budgets without changing the prompt or generation settings. For the twelve static families, saved post-action native states determine quality at each budget. Already-terminated episodes retain their final quality. Every point uses the same 126 instances per policy, with equal weights for families, configurations and instances. Native episode caps remain in force; larger plotted budgets do not extend them.

Additional calls raise static-task quality substantially at small budgets. From one to eight calls, Claude Opus 5.5 rises from 42.2 to 82.0, Kimi K3 from 31.4 to 68.9, MiniMax M3 from 9.7 to 32.8 and Kimi K2.6 from 12.0 to 35.0, while Qwen3.8-27B moves only from 2.3 to 3.1.

Beyond 16 calls the gains are small: 0.5 quality points for Claude Opus 5.5, 0.8 for Kimi K3, 0.8 for MiniMax M3, 1.9 for Kimi K2.6 and none for Qwen. Only 3–15 of 126 episodes per policy are still running after sixteen calls, so the plateau mostly reflects episodes that have already ended under native caps. It says nothing about larger budgets under a different planning policy. Final static-only scores are 89.2, 76.1, 40.8, 36.5 and 3.3 for Claude Opus 5.5, Kimi K3, Kimi K2.6, MiniMax M3 and Qwen; they differ from the full-suite aggregates because dynamic families are excluded.

### 5.2 Where path-dependent tasks fail

**Correct endpoints, failed paths.** We replayed every executed action and tested its press and release against each family's acceptance geometry (Appendix A.5). In track, maze and curve tracing, 37–82% of the failed actions of Claude Opus 5.5, Kimi K3, Kimi K2.6 and MiniMax M3 grab the correct object and release it inside the correct destination. The share grows with policy strength:

Table 6: Task quality by family under the common evaluation settings. Each cell covers all three evaluation instances of every configuration: 27 episodes for track and nine for every other family, so one episode can move a nine-episode cell by up to 11 points. Small numbers give 95% bootstrap half-widths over instances within each configuration; ±0.0 means every resample agrees. Values use a 0–100 scale.

| Family | Claude Opus 5.5 | Kimi K3 | Kimi K2.6 | MiniMax M3 | Qwen3.8-27B |
|---|---|---|---|---|---|
| *Placement* | | | | | |
| File sorting | 100.0±0.0 | 93.0±5.6 | 62.6±6.9 | 54.6±15.7 | 0.6±0.8 |
| Luggage | 100.0±0.0 | 100.0±0.0 | 68.9±11.1 | 86.7±11.1 | 0.0±0.0 |
| Funnel | 100.0±0.0 | 100.0±0.0 | 94.1±6.7 | 100.0±0.0 | 2.2±3.3 |
| Pharmacy | 100.0±0.0 | 100.0±0.0 | 74.1±10.2 | 57.4±12.0 | 0.0±0.0 |
| Blood bank | 100.0±0.0 | 100.0±0.0 | 63.1±16.1 | 68.8±9.8 | 1.2±1.9 |
| *Geometry* | | | | | |
| Track following | 88.9±7.4 | 74.3±6.7 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Glyph drawing | 68.7±8.8 | 45.4±10.4 | 5.4±8.0 | 0.0±0.0 | 0.0±0.0 |
| Maze | 66.7±22.2 | 11.1±16.7 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Painting | 84.2±1.6 | 72.3±7.4 | 39.7±6.5 | 26.2±12.9 | 18.4±10.6 |
| Lasso | 100.0±0.0 | 100.0±0.0 | 47.9±7.2 | 38.1±16.1 | 10.4±9.3 |
| Curve tracing | 62.1±11.9 | 39.5±6.6 | 8.5±2.7 | 5.6±1.8 | 6.6±2.3 |
| *Puzzle* | | | | | |
| Water sorting | 100.0±0.0 | 77.8±16.7 | 25.7±18.0 | 0.0±0.0 | 0.0±0.0 |
| *Dynamic* | | | | | |
| Flappy | 10.5±10.0 | 37.4±14.1 | 19.6±16.3 | 2.2±3.3 | 6.3±7.6 |
| Ping-pong | 52.2±17.6 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 9.3±10.2 |
| Volley | 25.9±18.5 | 17.0±18.9 | 26.7±17.8 | 18.5±18.5 | 13.3±17.8 |
| Fruit slicing | 40.4±20.2 | 66.7±22.2 | 3.7±5.6 | 33.5±23.5 | 43.2±22.2 |
| Hoop flick | 40.7±7.4 | 37.0±2.8 | 37.0±5.6 | 37.0±5.6 | 11.1±16.7 |
| Free kick | 22.2±16.7 | 22.2±16.7 | 11.1±16.7 | 0.0±0.0 | 0.0±0.0 |

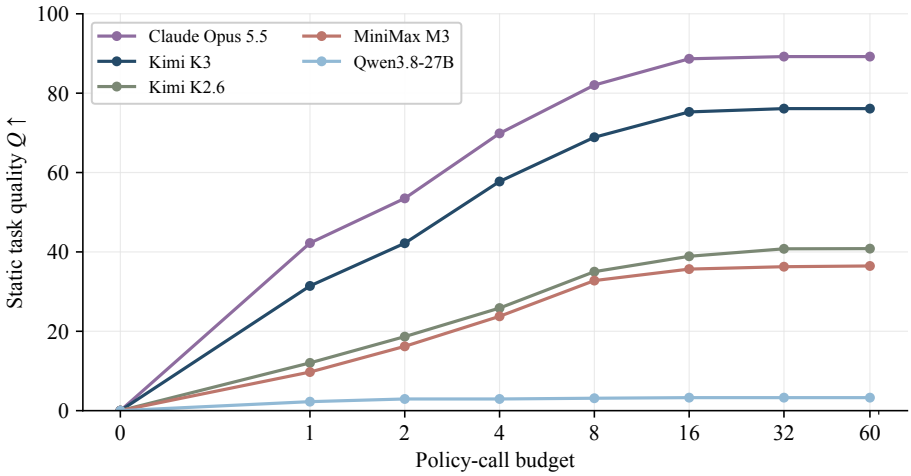


Figure 2: **Task quality versus policy-call budget.** Curves truncate recorded trajectories for the twelve static families. All five policies use the same instances and image resolution. Terminal outcomes are retained, including policy-output errors; quality is not replaced by a running maximum. These paired prefixes measure progress with additional opportunities to act, rather than independent runs with budget-aware planning.

82% for Claude Opus 5.5 and 60% for Kimi K3, against 37% for Kimi K2.6 and MiniMax M3. For Kimi K3, Kimi K2.6 and MiniMax M3, every endpoint-correct track failure left its corridor and every endpoint-correct maze failure struck a wall, so these failures are consistent with the path dependence that the intervention establishes, although this observational replay does not separate

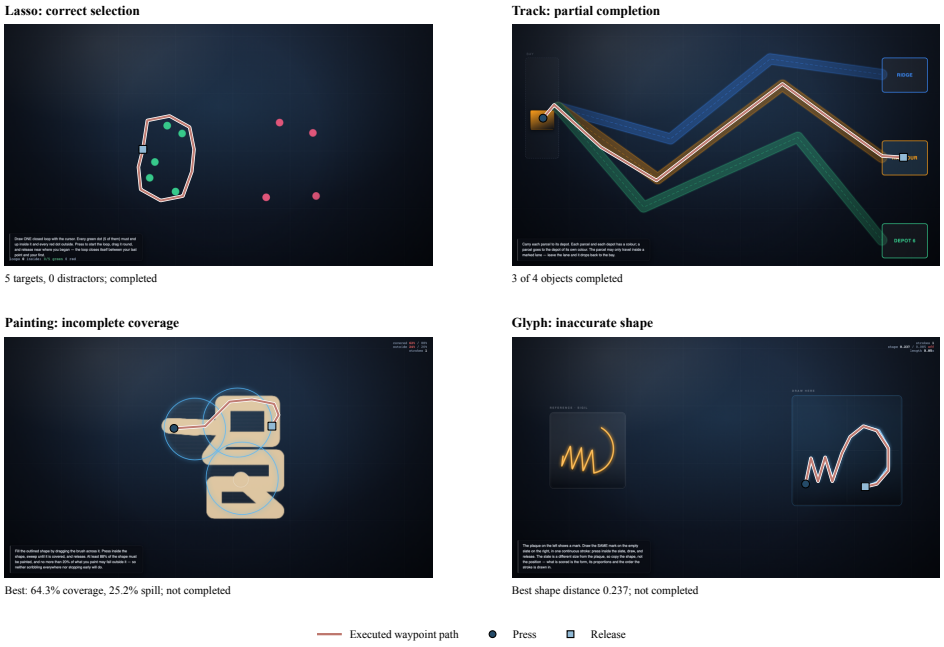


Figure 3: Recorded Kimi K3 trajectories: successful lasso selection, incomplete track transport, incomplete painting, and inaccurate glyph reproduction. Muted red shows the final executed path; circles and squares mark press and release. Backgrounds are the input observations before those actions. Annotations report episode outcomes; painting and glyph metrics describe the best recorded attempt, which need not be the action shown.

perception of path geometry from execution. Qwen3.8-27B has no such actions because it rarely grabs an object at all.

Figure 3 shows four selected Kimi K3 episodes; Appendix A.6 shows a random sample.

## 6 Limitations

DragForge measures synthetic interactions, including game-like dynamics, rather than arbitrary desktop workflows. Three evaluation instances per configuration provide limited evidence about procedural variability, and reference success on 660 instances does not establish universal solvability. Neither the difficulty annotations nor the benchmark isolate perception, reasoning and motor control as causes of failure. Dynamic simulation pauses during inference and omits real-time latency effects. Observations are downscaled to $960\times540$, which halves small cues such as 16-pixel labels, so perception contributes to some placement failures. Without a human baseline, the reference is a feasibility ceiling, not a human standard.

Reference normalization depends on the chosen controllers and finite random samples.

## 7 Conclusion

Across 18 procedural drag families, five vision-language policies reach task quality from 6.8 to 70.1 against 98.3 for the privileged reference; placement competence does not extend to geometry or dynamic control, and stronger policies often reach correct endpoints yet fail along the path. Human baselines and transfer to real interfaces remain open.

## REPRODUCIBILITY STATEMENT

Evaluation scripts record seeds, source hashes, prompts, actions, outcomes, and request metadata. Verification tools check logged prompts and screenshot integrity. A local archive includes source, selected records, request logs, and all final-study input images. From a clean extraction, offline checks validate all 900 episodes and regenerate the result tables byte-for-byte. Requests use temperature 1, top-p 0.99 and a fixed sampling seed; because hosted model versions can change, this guarantees re-analysis of the recorded episodes rather than identical re-collection, and the logs retain returned model identifiers and provider fingerprints. The complete benchmark code, including environments, reference controllers, the evaluation harness and analysis scripts, will be publicly released upon acceptance.

## AI USE STATEMENT

AI assistance was used to inspect code, implement evaluation tooling, analyze results, and draft text. Authors are responsible for checking the experiments, references, and claims.

## ETHICS STATEMENT

**Data and participants.** All environments are synthetic and procedurally generated. They contain no personal data, real user accounts or documents, and the study involved no human participants. Task assets are generated by the benchmark code.

**Dual use.** Reliable drag control is a general capability of computer-use agents. It could help accessibility tools and routine automation, but it could also help automate interactions that sites use to tell humans from bots, such as slider or drag-and-drop challenges. DragForge does not contain such challenges and does not target any third-party system, and we report aggregate capability rather than techniques for bypassing interaction checks. Agents that act on real interfaces should run with the permission of the system owner and with human oversight.

**Interpretation of results.** Scores measure bounded synthetic tasks under a fixed protocol. They should not be read as evidence that an agent is safe or reliable in real desktop workflows, and a high score does not license unsupervised deployment. The limitations in Section 6 apply to any such use.

**Model access and compute.** Models were accessed through hosted inference services under their terms of use, and no model was fine-tuned. Episode logs contain only synthetic observations and model outputs.

## REFERENCES

Johnny Accot and Shumin Zhai. Beyond Fitts' law: Models for trajectory-based HCI tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*, pp. 295–302, 1997.

Anthropic. Introducing Claude Opus 5.5. `https://www.anthropic.com/claude-opus-5-5`, 2026.

Nathan Bout, Maxime Langevin, and Ronan Riochet. DragOn: A benchmark and dataset for drag-based GUI interactions. In *ICML 2026 Workshop on Multimodal AI Agents (SCALE)*, 2026. arXiv:2606.06322.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 2048–2056, 2020.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pp. 28091–28114, 2023.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 11642–11662, 2024.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. CogAgent: A visual language model for GUI agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14281–14290, 2024.

Siyuan Hu, Kevin Qinghong Lin, and Mike Zheng Shou. ShowUI-$\pi$: Flow-based generative models as GUI dexterous hands. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8130–8140, 2026. arXiv:2512.24965.

Kimi Team. Kimi K2.6: Advancing open-source coding. `https://www.kimi.com/blog/kimi-k2-6`, 2026a.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026b.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The BrowserGym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. Expert Certification.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8778–8786, 2025.

Zeyi Liao, Yadong Lu, Boyu Gou, Huan Sun, and Ahmed Hassan Awadallah. Beyond clicking: A step towards generalist GUI grounding via text dragging. In *ICLR 2026 Workshop on Agents in the Wild: Safety, Security, and Beyond*, 2026. arXiv:2601.06031.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. ShowUI: One vision-language-action model for GUI visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 19498–19508, 2025.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*, 2018.

I. Scott MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human–Computer Interaction*, 7(1):91–139, 1992.

MiniMax. MiniMax M3: Frontier coding, 1M context, native multimodality — all in one model. `https://www.minimax.io/blog/minimax-m3`, 2026.

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.

Qwen Team. Qwen3.8-27B. `https://huggingface.co/Qwen/Qwen3.8-27B`, 2026.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina N. Toutanova. From pixels to UI actions: Learning to follow instructions via graphical user interfaces. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34354–34370, 2023.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 3135–3144, 2017.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-ATLAS: Foundation action model for generalist GUI agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 52040–52094, 2024.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 69772–69805, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

# A Protocol details

## A.1 Model evaluation settings

In the common-setting evaluation, all five policies receive 960×540 images resized from the 1920×1080 environment viewport. Coordinates are scaled back by two before execution. The common terminal-action decoder extracts the final raw or fenced JSON array after a reasoning boundary or at the end of the response. Each request has a 24,000-completion-token cap and a 360-second timeout; each episode permits at most 60 requests and each action at most 64 emitted waypoints.

Every request uses temperature 1, top-p 0.99 and a fixed sampling seed. After collecting the 900 primary episodes, we made one full-reset reattempt for each of 43 prespecified transient infrastructure failures: 36 Kimi K3, four MiniMax M3, two Kimi K2.6 and one Qwen3.8-27B episode, all transport failures that occurred before a scoreable policy output. Claude Opus 5.5 had no such failures. Excluding the 41 instances with any reattempt from every policy changes $Q$ by at most 1.3 points

and leaves the ordering unchanged. Every reattempt replaced its primary outcome irrespective of success; policy-output failures were not retried.

All five policies request medium reasoning effort. Providers may not honor these options, so we retain requested and returned model identifiers, usage and any provider fingerprints.

Table 7: Observation cues in the evaluated dynamic interfaces, established by code inspection. These are available cues, not evidence that a human or model can recover the full state.

| Family | Visible cues | Additional inference or limitation |
|---|---|---|
| Flappy | Corridor, craft and velocity-dependent tail | Indirect lift, damping and scrolling dynamics |
| Ping-pong | Ball history, shadow and predicted incoming intercept | Return mechanics; gold far-side region specifies the landing target |
| Volley | Ball history and engaged hand-velocity vector | Contact timing and velocity-to-return mapping |
| Fruit slicing | Fruit/bomb locations and blade history | Projectile motion and future spawns |
| Hoop flick | Rim, ball, phase; solved release guide on easy | Unguided launch dynamics on medium/hard |
| Free kick | Goal, defenders and keeper; solved curve on easy | Curvature and keeper dynamics; extra curvature rule |

Ping-pong displays two distinct regions: a gold far-side polygon specifies the accepted return zone, while a cyan near-side ellipse predicts where to intercept the incoming ball. The landing target follows the same generated width and perspective projection as the scoring predicate. Reading an incoming-interception aid does not by itself specify the bat contact offset needed for a valid return.

### A.2 Interpretation of recorded-prefix budgets

Only 3, 6, 7, 15, and 13 of the 126 static episodes per policy continue beyond sixteen calls for Claude Opus 5.5, Kimi K3, MiniMax M3, Qwen3.8-27B, and Kimi K2.6, respectively. Beyond 32 calls, the counts are 0, 5, 3, 5, and 4. Continuing episodes beyond sixteen calls belong only to file sorting, blood bank, and water sorting; Kimi K3 has no continuing blood-bank episode. The late plateau primarily describes a mixture of already terminated episodes under native caps. It does not establish that larger budgets with a different planning policy would be ineffective. All curve points retain the same denominator and family weights.

### A.3 CONTROL AUDIT

| Family | Instances | Reference | Random | No input |
|---|---|---|---|---|
| file_sort | 9 | 9 | 0 | 0 |
| luggage | 9 | 9 | 0 | 0 |
| funnel | 9 | 9 | 0 | 0 |
| pharmacy | 9 | 9 | 0 | 0 |
| blood_bank | 9 | 9 | 0 | 0 |
| track | 27 | 27 | 0 | 0 |
| water_sort | 9 | 9 | 0 | 0 |
| glyph | 9 | 9 | 0 | 0 |
| maze | 9 | 9 | 0 | 0 |
| paint | 9 | 9 | 0 | 0 |
| flappy | 9 | 9 | 0 | 0 |
| lasso | 9 | 9 | 0 | 0 |
| canvas_trace | 9 | 9 | 0 | 0 |
| ping_pong | 9 | 9 | 1 | 0 |
| volley | 9 | 9 | 0 | 0 |
| fruit_slice | 9 | 9 | 2 | 0 |
| hoop_flick | 9 | 9 | 1 | 0 |
| free_kick | 9 | 9 | 0 | 0 |
| Total | 180 | 180 | 4 | 0 |

Table 8: Strict completion in the control audit on development seeds 1000–1002. Privileged references establish feasibility; random policies provide a baseline; no-input execution checks for spontaneous completion over the full simulation horizon. These serve different validation purposes.

Development, feasibility and evaluation instances use seeds 1000–1002, 2000–2004 and 3000–3002; Figure 1 uses seed 17. Reference controllers additionally solve all 300 feasibility and all 180 evaluation instances. Only the evaluation controls enter normalization, and policy aggregates use evaluation instances only.

### A.4 BEHAVIORAL VALIDATION OF TASK QUALITY

We test the utility definitions with a prespecified four-condition audit on all 60 configurations at development seed 1000, and replicate it unchanged at seeds 1001 and 1002. The conditions execute no actions, a prefix of the saved reference actions, the full reference sequence, or a horizontal reflection of that sequence. Remaining episode time advances under the standard loop. Reflection is a geometric perturbation, not a guaranteed wrong-target label: a symmetric layout may admit a reflected solution, and some dynamics do not depend on the reflected coordinate.

At seed 1000, all 60 full-reference replays reproduce their saved native score, strict success, and utility, and so do all 120 at the two replication seeds. No no-action episode strictly succeeds. Two dynamic cases at seed 1000 nevertheless obtain quality 0.2 without input, flappy/easy and ping-pong/hard, and seven more at the replication seeds reach 0.14 to 0.5, mostly in ping-pong. Progress can therefore accumulate from environment evolution alone, which is why we report both the strict predicate and calibrated quality. Zero input is not assumed to imply zero utility.

For 15 configurations the reference has a single action, so its nonempty prefix is identical to the full sequence. We exclude these cases only from the claim about shortened sequences, retaining them in the audit records. Of the remaining 45 prefixes, 36 give quality strictly between zero and the full-reference value, two give zero, and seven retain the full-reference value. No prefix exceeds its matching full-reference utility. Reflected sequences succeed on seven configurations. At seeds 1001 and 1002 the prefixes split 38/1/6 and 36/3/6 into intermediate, zero and full quality, again with none exceeding its reference, and reflections succeed on five and eight configurations. Reflections therefore cannot be assigned zero utility without execution. These checks validate replay consistency and several coarse behavioral expectations, rather than proving cross-family cardinal comparability or agreement with human judgments. The seven shortened sequences retaining full quality at seed 1000 also show that reducing the number of reference actions does not necessarily reduce task ac-

complishment. The frozen case construction and complete native metrics accompany the evaluation evidence.

A separate prespecified stress audit targets four geometry-based scorers at all three configurations, again at development seed 1000. A broad raster adds irrelevant pixels in painting and tracing; a viewport-enclosing rectangle selects targets together with distractors; a doubled reference shape tests glyph validity. None of the twelve cases meets strict success, and every utility is below its matched correct reference (Table 9).

Table 9: Geometry-specific stress cases. Quality ranges show the minimum and maximum across three configurations, not statistical intervals.

| Family | Stress action | Completed | Task quality |
|---|---|---|---|
| Painting | Broad raster | 0/3 | 7.2–10.9 |
| Curve tracing | Broad raster | 0/3 | 2.8–6.2 |
| Lasso selection | Enclose all points | 0/3 | 45.5–55.6 |
| Glyph drawing | Double-scale reference | 0/3 | 0.0 |

Selecting every lasso point still earns 45.5–55.6 task-quality points because it includes all targets together with false positives. This is a consequence of target-set IoU, not successful selection. Similarly, broad coverage receives small nonzero mask IoU. These results separate partial overlap from solved tasks. They support resistance to these specific shortcuts at the strict-success level, but do not establish universal exploit resistance or justify treating quality as a success percentage.

### A.5 Endpoint attribution of model failures

We replay the logged, executed plans of all 900 evaluation episodes without model calls and recover the state before and after each action; every replay reproduces its saved success and score. A failed action is endpoint-correct when its press and release satisfy the family's acceptance geometry. In track, the press grabs an undelivered parcel and the release lies inside the depot that parcel must reach given current deliveries and capacity. In maze, the press lies within the page's grab radius of the token and the release inside the goal cell. In curve tracing, press and release lie within half the road width of the two road ends, in either order. Lasso, glyph and painting are excluded because their endpoints carry no target. Counts are actions, so repeated attempts of the same failing route are counted separately; the episode-level column is less sensitive to repetition.

Table 10: Endpoint attribution of failed actions. Path-dependent columns cover track, maze and curve tracing: failed actions, those whose press and release both satisfy the family's acceptance geometry, and failed episodes containing at least one such action out of all failed episodes. Placement columns split failed placement actions into presses that grab no object and releases outside the correct destination.

| | Path-dependent | | | Placement | | |
|---|---|---|---|---|---|---|
| Policy | Failed | Correct ends | Episodes | Failed | Press missed | Wrong/no target |
| Claude Opus 5.5 | 39 | 32 (82%) | 7/10 | 0 | 0 | 0 |
| Kimi K3 | 144 | 87 (60%) | 25/32 | 71 | 63 | 8 |
| Kimi K2.6 | 186 | 69 (37%) | 30/45 | 298 | 193 | 105 |
| MiniMax M3 | 147 | 55 (37%) | 18/45 | 292 | 197 | 95 |
| Qwen3.8-27B | 268 | 0 (0%) | 0/45 | 537 | 481 | 56 |

### A.6 Randomly sampled trajectories

Figure 4 complements the illustrative Figure 3 with six episodes drawn uniformly at random, with a fixed generator, from all track, maze and curve-tracing episodes of the five policies. Each panel shows the last executed action on its input observation. Track dominates the sample because it contributes 27 of the 45 such episodes per policy.

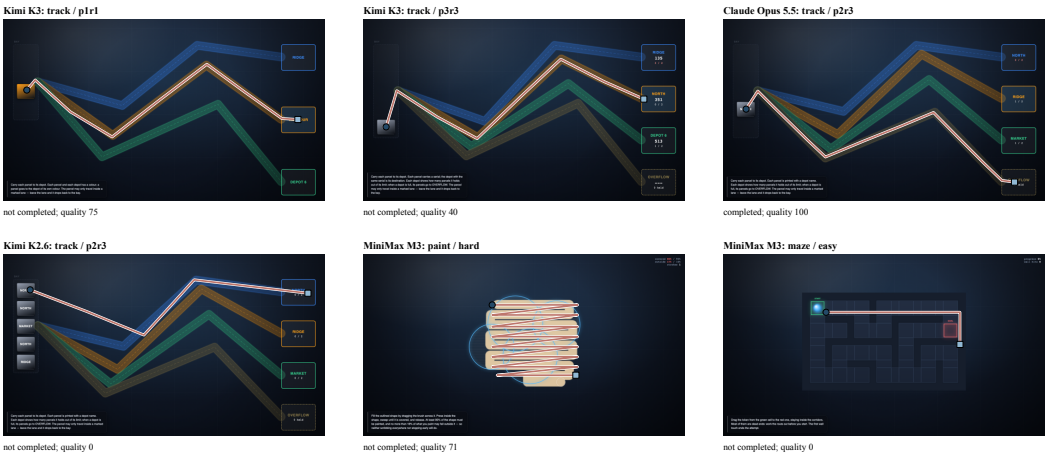


Figure 4: Randomly sampled track, maze and curve-tracing episodes from the five policies, with the last executed action and the episode outcome.

### A.7 Action-budget ablation

To test whether the 64-waypoint bound limits path execution, we reran Claude Opus 5.5 with a 16-waypoint bound on the 24 path-dependent static configurations at evaluation seed 3000, with the prompt stating the new bound and everything else unchanged. Strict completions change from 20 to 18 of 24; both lost episodes end with an invalid action. On the 22 instances where both runs end normally, mean task quality is 81.5 with 64 waypoints and 80.7 with 16. The 64-waypoint bound is therefore not the limiting factor for the strongest policy. This single-seed, single-model check does not rule out an effect for other policies.

### A.8 Path intervention details

All 117 original reference sequences succeed on both the development (seeds 1000–1002) and evaluation (seeds 3000–3002) instances. Each intervention is replayed from a fresh copy of the same instance, using the original action sequence without replanning after failure. Straight-path replacement joins the original endpoints. For perturbations, collinear midpoints are first inserted along the longest segments until the path has 63 spatial vertices; this preserves its geometry while allowing initially straight paths to be perturbed. Only interior coordinates are changed, independently by at most five or twenty pixels per axis. Initial and final positions remain fixed.

Painting retains nine of nine successes under the larger perturbation, whereas track and maze retain none. These differences reflect the task predicates; success under one perturbation magnitude is not a general measure of geometric tolerance.

### A.9 Reference calibration

Reference normalization requires a positive gap between independently measured reference and random means on the same configuration and evaluation layouts. A small gap amplifies anchor estimation error; unclipped scores can be negative or exceed 100. Task quality $Q$ does not depend on anchor estimation. The controller is a reference implementation, not a demonstrated optimum.

The random controller independently samples press and release coordinates uniformly over the viewport and uses a duration of $\max(2, \min(h - 1, 10))$ ticks for planning horizon $h$. It follows the same episode loop and decision limits. One RNG stream is evaluated per layout, seeded by its layout seed, giving three outcomes per configuration and 180 overall. These finite empirical anchors are held fixed for every policy. All reference–random gaps are positive on the final split; the smallest is $1/3$ for easy hoop flick. Three draws per configuration provide limited precision, so we ran 30 additional random draws on each of the 12 configurations whose random quality is nonzero; the other 48 score zero in every draw. The extra draws lower the random mean of easy hoop flick from 0.67 to

Table 11: Completed development instances under endpoint-preserving path changes, by family. Table 3 gives the evaluation-instance totals.

| | Original | Straight | Path perturbation | |
|---|---|---|---|---|
| Family | path | path | $\pm 5$ px | $\pm 20$ px |
| *Placement* | | | | |
| File sorting | 9/9 | 9/9 | 9/9 | 9/9 |
| Luggage routing | 9/9 | 9/9 | 9/9 | 9/9 |
| Funnel sorting | 9/9 | 9/9 | 9/9 | 9/9 |
| Pharmacy sorting | 9/9 | 9/9 | 9/9 | 9/9 |
| Blood-bank allocation | 9/9 | 9/9 | 9/9 | 9/9 |
| *Path-dependent* | | | | |
| Track following | 27/27 | 0/27 | 27/27 | 0/27 |
| Glyph drawing | 9/9 | 1/9 | 9/9 | 1/9 |
| Maze navigation | 9/9 | 0/9 | 9/9 | 0/9 |
| Painting | 9/9 | 0/9 | 9/9 | 9/9 |
| Lasso selection | 9/9 | 0/9 | 9/9 | 2/9 |
| Curve tracing | 9/9 | 0/9 | 9/9 | 3/9 |

0.12 and change the other anchors by at most 0.07. Recomputing $N$ with these anchors changes it by at most 0.5 points for four policies and from 2.6 to 5.4 for Qwen3.8-27B, and leaves the ordering unchanged.

Table 12: All active configurations. Cadence is simulation ticks per model decision; a dash denotes static execution. Parameters not listed use generator defaults in the source snapshot.

| Family | Config. | Generator overrides | Cadence |
|---|---|---|---|
| file_sort | easy | n_files=3, n_folders=2 | – |
| file_sort | medium | n_files=6, n_folders=3 | – |
| file_sort | hard | n_files=20, n_folders=5 | – |
| luggage | easy | n_bags=5, n_chutes=3, tag_px=22, confusable=False | – |
| luggage | medium | n_bags=5, n_chutes=6, tag_px=16, confusable=False | – |
| luggage | hard | n_bags=5, n_chutes=6, tag_px=16, confusable=True | – |
| funnel | easy | n_balls=2 | – |
| funnel | medium | n_balls=3 | – |
| funnel | hard | n_balls=5 | – |
| pharmacy | easy | n_boxes=4, dose_px=22, confusable=False | – |
| pharmacy | medium | n_boxes=6, dose_px=16, confusable=False | – |
| pharmacy | hard | n_boxes=6, dose_px=16, confusable=True | – |
| blood_bank | easy | n_bags=5, n_groups=2, capacity=2, group_px=22 | – |
| blood_bank | medium | n_bags=6, n_groups=3, capacity=2, group_px=16 | – |
| blood_bank | hard | n_bags=9, n_groups=4, capacity=2, group_px=16 | – |
| track | p1r1 | p_level=1, r_level=1, n_parcels=4, n_depots=3 | – |
| track | p1r2 | p_level=1, r_level=2, n_parcels=4, n_depots=3 | – |
| track | p1r3 | p_level=1, r_level=3, n_parcels=5, n_depots=3 | – |
| track | p2r1 | p_level=2, r_level=1, n_parcels=4, n_depots=3 | – |
| track | p2r2 | p_level=2, r_level=2, n_parcels=4, n_depots=3 | – |
| track | p2r3 | p_level=2, r_level=3, n_parcels=5, n_depots=3 | – |
| track | p3r1 | p_level=3, r_level=1, n_parcels=4, n_depots=3 | – |
| track | p3r2 | p_level=3, r_level=2, n_parcels=4, n_depots=3 | – |
| track | p3r3 | p_level=3, r_level=3, n_parcels=5, n_depots=3 | – |
| water_sort | easy | n_colors=3, spare=3 | – |
| water_sort | medium | n_colors=4, spare=2 | – |
| water_sort | hard | n_colors=5, spare=1 | – |
| glyph | easy | n_strokes=1, tolerance=0.115 | – |
| glyph | medium | n_strokes=2, tolerance=0.085 | – |
| glyph | hard | n_strokes=3, tolerance=0.07 | – |
| maze | easy | cols=7, rows=4, cell=132, braid=0.05 | – |
| maze | medium | cols=9, rows=6, cell=118, braid=0.12 | – |

| Family | Config. | Generator overrides | Cadence |
|---|---|---|---|
| maze | hard | cols=12, rows=7, cell=96, braid=0.2 | – |
| paint | easy | n_lobes=2, brush=40.0, coverage=0.84, max_spill=0.26 | – |
| paint | medium | n_lobes=3, brush=34.0, coverage=0.88, max_spill=0.2 | – |
| paint | hard | n_lobes=4, brush=26.0, coverage=0.9, max_spill=0.16 | – |
| flappy | easy | corridor=300.0, wind_amp=105.0, wind_freq=0.58, speed=260.0, n_sections=5 | 12 |
| flappy | medium | corridor=200.0, wind_amp=165.0, wind_freq=0.3, speed=325.0, n_sections=6 | 12 |
| flappy | hard | corridor=140.0, wind_amp=135.0, wind_freq=0.3, speed=385.0, n_sections=7 | 12 |
| lasso | easy | layout_kind=cluster | – |
| lasso | medium | layout_kind=bridge | – |
| lasso | hard | layout_kind=double, n_decoys=6 | – |
| canvas_trace | easy | shape=wave, stroke_width=44 | – |
| canvas_trace | medium | stroke_width=28 | – |
| canvas_trace | hard | shape=spiral, stroke_width=16 | – |
| ping_pong | easy | mode=serve, n_serves=4, ball_speed=0.85, required_frac=0.5, target_half=0.72, max_seconds=20 | 30 |
| ping_pong | medium | mode=serve, n_serves=10, ball_speed=1.35, required_frac=0.6 | 15 |
| ping_pong | hard | mode=serve, n_serves=12, ball_speed=1.45, required_frac=0.42 | 15 |
| volley | easy | target_score=1, opp_speed=420, hands_r=72, grab_radius=150, contact_gain=1.35, up_bias=-520, net_h=240, max_touches=4, opp_reach=160, max_seconds=5 | 30 |
| volley | medium | target_score=3, opp_speed=760 | 14 |
| volley | hard | target_score=5, opp_speed=980 | 14 |
| fruit_slice | easy | n_fruits=6, n_bombs=1, required_frac=0.5, spawn_seconds=22.0 | 24 |
| fruit_slice | medium | n_fruits=12, n_bombs=3, required_frac=0.75, spawn_seconds=22.0 | 24 |
| fruit_slice | hard | n_fruits=20, n_bombs=6, required_frac=0.85, spawn_seconds=22.0 | 24 |
| hoop_flick | easy | n_shots=3, required_frac=0.34, moving_hoop=False, rim_w=260, aim_guide=True, aim_assist=True, aim_assist_angle_deg=12.0, aim_assist_speed_frac=0.5 | 15 |
| hoop_flick | medium | n_shots=5, required_frac=0.6, moving_hoop=False | 15 |
| hoop_flick | hard | n_shots=8, required_frac=0.75, moving_hoop=True | 15 |
| free_kick | easy | n_kicks=3, required_frac=0.34, wall_blocking=True, goal_half=420, wall_cover_half=240, wall_y=450, keeper_reach=28, keeper_speed=240, keeper_commit_y=760, keeper_curve_gain=0.0, curve_guide=True, curve_assist=False, min_goal_spin=0.015 | 15 |
| free_kick | medium | n_kicks=5, required_frac=0.6, wall_blocking=True | 15 |
| free_kick | hard | n_kicks=8, required_frac=0.75, wall_blocking=True | 15 |

### A.10 TASK-PROGRESS DEFINITIONS

For count-based families, utility is $\min(1, k/k_{\text{required}})$, where $k$ is the native correct count and the denominator is the generated task requirement (number of files for file sorting, target score for volley). Fruit slicing receives zero utility after a bomb hit. Capacity, crash and other fault diagnostics remain distinct from subgoal progress. For paint and canvas trace, let $r$ be target coverage and $p$ be one minus spill/off-path fraction; mask IoU is $rp/(r + p - rp)$, with zero for no intersection. Lasso utility is $\text{TP}/(n_{\text{targets}} + \text{FP})$. Glyph utility is $\exp(-d/\tau)$ using the environment's normalized

shape distance and configured tolerance, gated by the existing inside-slate and allowed-length-ratio checks; no stroke receives zero. Maze uses binary completion. These definitions do not claim that every family has a continuous score.

The evaluator records the resulting native state before closing an episode, including when a policy emits an invalid action. A formatting failure can occur after useful earlier actions; its task quality is therefore evaluated from the resulting state rather than inferred from the response status. Recorded strict policy-episode completion additionally requires no terminal output failure: an invalid, truncated or refused action counts as unsuccessful even when the retained state satisfies the task predicate. The retained state predicate is stored separately in such cases. No privileged state is supplied to the policy.

### A.11 TASK-STATE SUCCESS PREDICATES

These predicates are read from the executed environment. A valid final release is not a universal requirement: some tasks succeed during movement, and others retain the best completed stroke. Budget exhaustion with no success is a task failure. The source-linked supplement identifies the predicate implementations.

| Family | Success condition |
|---|---|
| file_sort | Every file assignment equals its generated destination folder. |
| luggage | Every suitcase assignment equals its generated destination chute. |
| funnel | Every ball assignment equals its generated color-matched chute. |
| pharmacy | Every carton assignment equals its generated dose-based tray. |
| blood_bank | Every group fills its own fridge up to the smaller of group count and capacity; all remaining units are in overflow, with no foreign-group units in a fridge. |
| track | All parcels are assigned to their home depots; reasoning level 3 additionally requires exact home-capacity and overflow counts. Leaving all lanes while holding drops the parcel, so route validity affects state transitions. |
| water_sort | The number of completed monochromatic full tubes equals the number of colors. |
| glyph | A completed stroke lies entirely on the slate; its normalized, 128-sample shape distance is within the configured tolerance, and its length ratio lies between the reciprocal and direct configured maximum. Either drawing direction is accepted. |
| maze | The token reaches the goal cell through valid collision-checked movement. Success is set on reaching the goal, not on a final release. |
| paint | A completed stroke covers at least the configured fraction of the target mask, with painted area outside the mask no greater than the configured spill fraction. |
| flappy | All tunnel sections are cleared and the craft has not crashed. |
| lasso | A completed loop contains every target dot and no distractor dot under even-odd containment. The best accepted loop is retained. |
| canvas_trace | Target-path pixel coverage is at least 75%, and at most 30% of painted pixels are off the target path. |
| ping_pong | In serve mode, the number of placed serves reaches the required count; otherwise the agent score reaches the target score. |
| volley | The agent score reaches the configured target score. |
| fruit_slice | At least the required number of fruit have been sliced and no bomb has been hit. |
| hoop_flick | The number of made baskets reaches the required count. |
| free_kick | The number of scored goals reaches the required count. |

### A.12 EXACT POLICY PROMPTS

The shared system prompt below is for a static call. Dynamic calls substitute the current planning horizon. The user message contains one PNG observation and the instance instruction, without conversational history.

> You control a mouse cursor on a 960x540 pixel screen. The world is PAUSED while you decide and resumes only when you answer, so take your time — but it will then advance 256 simulation ticks at 60 ticks per second, and your answer must cover that whole span.
>
> Your action is a single drag: the button goes down, the cursor travels, the button comes up.
>
> Output exactly one JSON array on a single line:
>
> [{"t": INT, "x": INT, "y": INT, "btn": 0 or 1}, ...]

- ‘t‘ is a tick offset from now; the first must be 0 and ticks must increase.
- ‘x‘, ‘y‘ are absolute pixels, integers, origin (0,0) top-left.
- ‘btn‘ is 1 while the button is held, 0 when released.
- Between two waypoints the cursor moves in a straight line at constant speed, so the SHAPE of the path is set by how many waypoints you give and where.
- Use at most 64 points.
- Objects in the scene may be MOVING. Aim at where a thing will be when you get there, not at where it is now.
- Reply with the JSON only: no markdown fences, no commentary before or after it.

This is one decision in a multi-step episode. Perform only ONE drag now. After execution you will receive an updated screenshot and can act again. Do not try to finish all subgoals in this reply.

The task examples below use the first configuration of each family at seed 3000. The machine-readable evidence contains all 180 instance instructions.

**file_sort** GOAL. Every file icon on the desktop must end up inside the folder whose label matches that file’s kind. The episode succeeds only if all of them do.

CONTROL. Press on a file icon, hold, move to a folder, release. Pressing anywhere that is not a file does nothing.

ENCODING. The drag is a direct move: the file goes where you release it.

SCORING. One drag per file; you may take them in any order.

**luggage** GOAL. Every suitcase must end up in the chute whose number matches the number printed on that suitcase’s tag. The episode succeeds only if all of them do.

CONTROL. Press on a suitcase, hold, move to a chute, release. Pressing anywhere that is not a suitcase does nothing.

ENCODING. The drag is a direct move: the suitcase goes where you release it.

SCORING. The suitcases are identical apart from their tags — shape, colour and size carry no information at all. Several chute numbers may use the SAME DIGITS in a different order, so reading which digits a tag contains is not enough; the order has to match too.

THIS EPISODE. Every suitcase is identical apart from its flight number. Drag each one into the chute whose number matches that flight number.

**funnel** GOAL. Every ball must end up in the chute of its own colour.

CONTROL. Press on a ball, hold, move to a chute, release.

ENCODING. The drag is a direct move: the ball goes where you release it.

SCORING. All balls must be placed correctly.

THIS EPISODE. Drag each ball into the chute of the same colour.

**pharmacy** GOAL. Every carton must end up in the LOW tray if its printed dose is 10 mg or less, and in the HIGH tray if the dose is more than 10 mg.

CONTROL. Press on a carton, hold, move to a tray, release.

ENCODING. The drag is a direct move: the carton goes where you release it.

SCORING. The cartons are identical apart from the printed dose. READ THE DECIMAL POINT: doses such as 5.0 mg and 50 mg use the same digits in the same order and fall on opposite sides of the threshold.

THIS EPISODE. Every carton is identical apart from the printed dose. Put each one in the LOW tray if its dose is 10 mg or less, and in the HIGH tray if it is more.

**blood_bank** GOAL. Every bag must end up in a fridge that is legal for it.

CONTROL. Press on a bag, hold, move to a fridge, release.

ENCODING. The drag is a direct move: the bag goes where you release it.

THE RULE. A bag belongs in the fridge labelled with its own blood group. Each of those fridges holds a limited number of units and displays how many it currently holds out of that limit. Once a group's fridge is full, every remaining bag of that group goes in the OVERFLOW fridge instead.

SCORING. So the right answer for a bag DEPENDS ON WHAT YOU HAVE ALREADY FILED. A bag may not go to OVERFLOW while its own fridge still has room, and may never go to another group's fridge. The bags are identical apart from the printed group, and the + and - signs matter.

THIS EPISODE. Each fridge holds 2 units and shows how many it has. Put every bag in the fridge for its own blood group; once that fridge is full, put the rest in OVERFLOW.

**track** GOAL. Every parcel must be carried from the bay to its own depot.

CONTROL. Press on a parcel, hold, move to a depot, release.

ENCODING. THE ROUTE IS SCORED, NOT JUST THE ENDPOINTS. While you are holding a parcel the cursor must stay inside a marked lane — the faint striped channels painted from the bay to each depot. The moment the cursor leaves every lane, the parcel drops and returns to the bay. The lanes bend, so a straight drag from the bay to a depot leaves the lane and delivers nothing. Emit enough intermediate points to follow the bend.

SCORING. Leaving the lane costs the attempt, not the episode: the parcel goes back to the bay and can be picked up again.

THIS EPISODE. Carry each parcel to its depot. Each parcel and each depot has a colour; a parcel goes to the depot of its own colour. The parcel may only travel inside a marked lane — leave the lane and it drops back to the bay.

**water_sort** GOAL. Finish with every colour alone in a tube.

CONTROL. Drag from one tube to another to pour. Nothing travels along the path — the drag names a source and a destination and nothing else.

ENCODING. You may pour only onto the same colour or into an empty tube, and only as much as fits. The whole top run of one colour moves at once.

SCORING. LOOK AHEAD. Every pour is reversible in principle and irreversible in practice: legal pours lead to positions from which the puzzle can no longer be solved, and nothing on screen tells you which ones. An empty tube is the scarcest thing you have.

THIS EPISODE. Sort the liquid. Drag from one tube to another to pour. You may pour only onto the same colour or into an empty tube, and only as much as fits. Finish with each of the 3 colours alone in a tube.

**glyph** Complete the task shown on screen.

THIS EPISODE. The plaque on the left shows a mark. Draw the SAME mark on the empty slate on the right, in one continuous stroke: press inside the slate, draw, and release. The slate is a different size from the plaque, so copy the shape, not the position — what is scored is the form, its proportions and the order the stroke is drawn in.

**maze** Complete the task shown on screen.

THIS EPISODE. Drag the token from the green cell to the red one, staying inside the corridors. Most of them are dead ends: work the route out before you start. The first wall touch ends the attempt.

**paint** Complete the task shown on screen.

THIS EPISODE. Fill the outlined shape by dragging the brush across it. Press inside the shape, sweep until it is covered, and release. At least 84% of the shape must be painted, and no more than 26% of what you paint may fall outside it — so neither scribbling everywhere nor stopping early will do.

**flappy** GOAL. Keep the craft inside the tunnel until every section is cleared.

CONTROL. Press and HOLD. While you hold, the craft climbs or dives toward the cursor's height; release and it falls.

ENCODING. THE CURSOR IS A SETPOINT, NOT A POSITION. The craft accelerates toward the height you are holding and lags behind it, so aim above where you want it and let it catch up — and aim at where the tunnel WILL be, not where it is, because holding the current centreline is always one lag too late.

SCORING. The world scrolls at a fixed rate and does not wait for you. There is one tunnel and no choice of route: touching either wall ends the run, and the score is how far you got.

THIS EPISODE. Hold the craft and steer it up and down. While you hold, it climbs or dives toward the cursor; let go and it falls. The tunnel scrolls at you and does not wait. Keep the craft inside it for all 5 sections — touching a wall ends the run.

**lasso** GOAL. Draw ONE closed loop that has every green dot inside it and every red dot outside.

CONTROL. Press to start the loop, drag it round, release to finish. The loop closes itself with a straight line from your last point back to your first.

ENCODING. WHAT IS READ IS THE REGION YOU ENCLOSED, not where you pressed or released. No part of the drag decides the answer on its own — a loop one crossing different encloses a different set. Containment is even-odd, so a loop that crosses itself EXCLUDES the part it covers twice.

SCORING. A circle round everything green will often also catch something red. When it does, the loop has to be concave: go round one group, along a narrow corridor that passes clear of the red dots, round the next group, and back beside the way you came.

THIS EPISODE. Draw ONE closed loop with the cursor. Every green dot (5 of them) must end up inside it and every red dot outside. Press to start the loop, drag it round, and release near where you began — the loop closes itself between your last point and your first.

**canvas_trace** GOAL. Retrace the grey path drawn on the canvas, starting from the green dot.

CONTROL. One single continuous drag: press at the start, follow the path without releasing, release at the end.

ENCODING. Your cursor paints as it moves. The shape of the whole route is the answer — the endpoints alone are worth nothing.

SCORING. At least 75% of the grey path must be covered by your stroke, AND at most 30% of your stroke may fall outside it. Scribbling over everything fails the second condition.

**ping_pong** GOAL. Return the ball. The table is drawn in perspective: the far end is the opponent's, the near end is yours.

CONTROL. Press ON YOUR BAT and hold. While held, moving the cursor LEFT and RIGHT slides the bat across the table, and moving it UP and DOWN raises and lowers the bat. Releasing drops the bat. Pressing anywhere else does nothing. The bat has a speed limit and will not teleport.

ENCODING. Both cursor axes matter. Because of the perspective, a pixel near the far end of the table is worth much more table distance than a pixel near you. Where the ball strikes the bat decides the return: contact off to one side sends it that way, contact high or low changes the lift.

SCORING. The ball arcs and bounces, so meet it where it will be, at the height it will be.

THIS EPISODE. Balls are served at you one after another. Drag your red bat to return each one into the highlighted zone on the far side. Return 2 of 4.

**volley** GOAL. Win points by making the ball land on the opponent's sand.

CONTROL. Press on your player and hold to move them around your half of the court. The ball comes off their raised forearms.

ENCODING. THE BALL FLIES THE WAY YOUR HANDS WERE MOVING. Outgoing velocity is proportional to the speed of your hands at the moment of contact, plus a small fixed lift. Arriving at the ball and stopping there returns almost nothing — you must sweep THROUGH it. You get at most three touches on your side, and the ball must clear the net.

SCORING. A point is won when the ball lands in the opponent's half.

THIS EPISODE. Drag your player to bump the ball back over the net. The ball flies the way you SWING — sweep fast through the ball, don't just park under it. First to 1.

**fruit_slice** GOAL. Slice at least the required number of fruit before they fall off screen, and touch no bomb. One bomb ends the episode immediately as a failure.

CONTROL. While the button is held, the cursor is a blade. Anything its swept path crosses is cut.

ENCODING. The blade only cuts while MOVING FAST — resting the cursor on a fruit does nothing. Targets fly on ballistic arcs, so aim where a fruit WILL be, not where it is. One fast swipe can take several fruit at once.

SCORING. Fruit that lands untouched is lost; there is no second chance at it.

THIS EPISODE. Press and drag to swing the blade through the fruit. Slice at least 3 of 6. Do NOT touch the black bombs.

**hoop_flick** GOAL. Make the required number of baskets out of the shots you are given.

CONTROL. Press ON THE BALL, drag, release. Pressing away from the ball does nothing and wastes nothing; failing to release in time forfeits the shot.

ENCODING. THE DRAG IS NOT A DESTINATION. It is a throw. The ball is launched with a velocity proportional to the vector FROM your press point TO your release point: releasing up and to the right throws it up and to the right, and a longer drag throws it harder. The ball then flies as a projectile under gravity, so a drag aimed straight at the hoop falls short — you must aim above it. You cannot influence the ball once released.

SCORING. The ball must fall through the hoop from above.

THIS EPISODE. Press on the ball, drag, and release to shoot. The drag vector sets the throw (longer drag = harder). Make 1 of 3 shots into the hoop. Drag from the ball to the green RELEASE marker. An approximate release inside the guide tolerance is accepted.

**free_kick** GOAL. Score the required number of goals from the free kicks you are given.

CONTROL. Press near the ball, swipe, release.

ENCODING. Three things are read from your swipe. Its overall direction aims the shot. Its length sets the pace. Its CURVATURE sets spin, and spin bends the ball sideways in flight — a swipe that curves left makes the ball bend left. A perfectly straight swipe has zero spin and flies straight.

THE WALL BLOCKS EVERY STRAIGHT SHOT. The defenders are positioned so that no straight line from the ball to any part of the goal is open. Only a curved swipe, bending the ball around the wall and back in, can score. The keeper also reads the bend the ball is already showing and moves to meet it.

SCORING. The ball must cross the goal line between the posts.

THIS EPISODE. Press near the ball and swipe to shoot. A CURVED swipe bends the ball in flight — the wall blocks every straight shot. Score 1 of 3. Follow the cyan curve and release inside the marked ring.

### A.13 COMPLETE COMMON-SETTING RESULTS

Table 14: Family-balanced task quality on each of the three evaluated seeds and its sample standard deviation. The SD describes variability across these three seed-indexed suites; it is not a confidence interval.

| Policy | Seed 3000 | Seed 3001 | Seed 3002 | Seed SD |
|---|---|---|---|---|
| Claude Opus 5.5 | 69.6 | 69.3 | 71.5 | 1.2 |
| Kimi K3 | 59.2 | 65.1 | 58.0 | 3.8 |
| Kimi K2.6 | 31.0 | 33.6 | 33.3 | 1.4 |
| MiniMax M3 | 26.2 | 32.5 | 29.5 | 3.2 |
| Qwen3.8-27B | 5.4 | 6.5 | 8.5 | 1.6 |

Table 15: Recorded strict episode completion and terminal-status counts in the common-setting evaluation. Success is a subset of ordinary endings, which include successful and unsuccessful task execution. Terminal output failures count as unsuccessful even if the retained state satisfies the task predicate. The first four count columns partition each policy's 180 episodes, as the Total column shows; Success counts strictly completed episodes and is a subset of Ordinary end. Counts weight episodes equally, unlike family-balanced task quality.

| | Terminal status (partition) | | | | | |
|---|---|---|---|---|---|---|
| Policy | Ordinary end | Invalid | Truncated | Refusal | Total | Success |
| Claude Opus 5.5 | 158 | 22 | 0 | 0 | 180 | 124/180 |
| Kimi K3 | 178 | 2 | 0 | 0 | 180 | 93/180 |
| Kimi K2.6 | 134 | 34 | 12 | 0 | 180 | 24/180 |
| MiniMax M3 | 100 | 7 | 73 | 0 | 180 | 25/180 |
| Qwen3.8-27B | 179 | 1 | 0 | 0 | 180 | 5/180 |