BLENDCUA: SELF-EVOLVING COMPUTER-USE HARNESS FOR BLENDER USE

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper introduces BLENDCUA, a self-evolving method for harness generation for an LLM agent carrying out Blender tasks given in natural language with reference imagery, covering modelling, materials, physics simulation and animation. Such an agent depends on its harness, the tools it can use and the skills that show it procedures, and building one by hand for every new family of tasks costs as much as the expert demonstrations it was meant to replace. Self-improving agents grow a harness by compiling successful trajectories into skills, which captures what the agent did rather than what its result got wrong. We instead evolve the harness from the defects in the agent's own finished work, through outcome-driven credit assignment. Starting from a hand-built harness, BLENDCUA builds each task, has an independent checker name the defects in the finished model, traces each defect to the part of the harness responsible, and keeps a repair to that part, whether a plan correction, a skill or a tool, only if a rebuild shows the defect closed. Evaluated on held-out Blender tasks with judged and computed measures of quality, BLENDCUA outperforms a harness learned from transcripts on every measure under the same agent and curriculum, reducing major defects by 29%. These results establish that the harness an agent needs for professional creative software can be grown from its own finished work rather than authored by an expert.

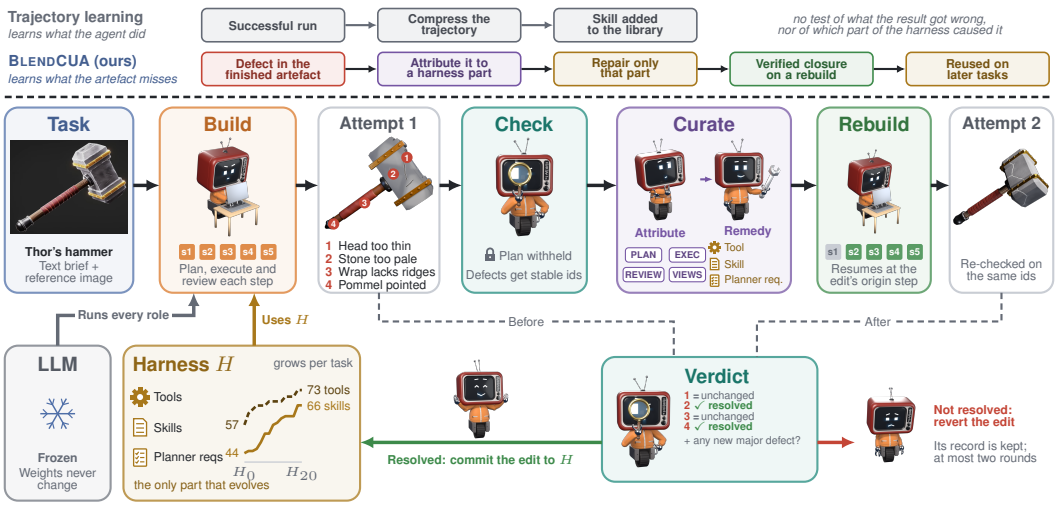


Figure 1: **Learning from what the artefact misses.** *Top:* trajectory learning compresses a successful run into a skill; BLENDCUA traces a defect in the finished artefact to one part of the harness, repairs only that part, verifies closure on a rebuild and reuses the repair. *Bottom:* the loop on one task. A checker that never sees the plan names the defects in the finished model; the curator attributes each and writes a tool, a skill or a planner requirement; the task is rebuilt from the step that remedy first touches, and the edit is committed to H only if its defect is resolved. The LLM stays frozen.

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

Building a 3D asset is iterative, skilled work. An artist given a brief and a few reference images blocks out the form, notices that a proportion is off or that a part does not meet its neighbour, and goes back. Large language model agents can now take part in this loop: they write Python against Blender's scripting interface, render the result and revise it (Hu et al., 2024; Huang et al., 2024; He et al., 2026).

Operating Blender is no longer the limiting factor. What limits these agents is the harness around the model: the procedural knowledge that says how a kind of object is built, the checks that say when a step is done, and the tools that make awkward operations and measurements tractable. In current systems this harness is written by hand, and He et al. (2026) attribute their gains over a monolithic agent to harness design rather than to the model. A harness written for furniture does not carry over to animated creatures or cloth, and extending it to each new family of tasks costs as much as the expert demonstrations it replaces (Gupta et al., 2026). The missing step is to encode, from the agent's own experience, the procedural knowledge, checks and tool support that a whole family of future tasks will need.

There are two ways to learn this from experience. Trajectory-based methods learn from *what the agent did*: a successful run is compiled into a reusable skill or function and retrieved on later tasks (Wang et al., 2023; Hu et al., 2024; Wang et al., 2024; Zheng et al., 2025). This works when success can be checked by a program, but in 3D work the record of what the agent did says little about what the result misses. An agent can leave a chair back floating clear of its seat in a few untroubled actions that leave no trace in the log, and compiling that run preserves the procedure that produced the flaw. We learn instead from *what the artefact misses*: a defect in the finished model is traced back to the part of the harness that allowed it, and only that part is repaired. We call this *outcome-driven credit assignment*. The question is not which behaviour to keep, but which component of the harness is responsible for a specific failure, and what change to it would have prevented it.

This paper introduces BLENDCUA, which grows a computer-use harness in this way (Figure 1). After the agent builds a task, an independent checker names the defects in the finished model; a curator attributes each to the plan, the step review, the execution or the generated reference views, and writes the repair that cause demands: a planner requirement when the property was never asked for, a skill when the agent lacked the knowledge, a tool when it lacked the capability. The task is rebuilt with the repair as the only change, and the repair is kept only if its defect closes. Kept repairs serve every later task, so a lesson learned on one object serves its family (Figure 2). The loop starts from H_0, a harness written by hand by an experienced Blender user, and grows by extending it: new tools follow the patterns of existing ones and new skills refine procedures the agent has already seen. The model is frozen throughout.

Over a curriculum of 20 Blender tasks, evaluated on 10 held-out ones, the sharpest comparison is with our earlier loop, which mines the harness from transcripts and shares the agent, model and curriculum, so it differs only in the learning signal. Against it, learning from defects lowers major defects per build from 2.8 to 2.0, raises computed mesh quality from 84.5 to 92.3 and improves mean rank from 3.9 to 2.5. The updates are selective and reusable: the gate rejects between a quarter and three fifths of the remedies it judges for every cause but one, and 14 of the 16 tools and 14 of the 22 skills the harness gains are used again on later tasks. Our contributions are:

- **Learning from defects.** A defect in the finished artefact, found by an independent checker, replaces the successful trajectory as the learning signal for a computer-use harness.
- **Credit assignment to harness components.** Each defect is attributed to the plan, review, execution or reference views, and only that component is repaired; a rebuild in which the repair is the only variable decides whether it is kept.
- **A hand-built starting harness.** H_0, written by an experienced Blender user, supplies the core tools and procedures that the agent's repairs extend rather than replace.

2 RELATED WORK

LLM agents for 3D content. SceneCraft (Hu et al., 2024) turns a description into a Blender script through a scene graph and refines it from renders; 3D-GPT (Sun et al., 2023) calls procedural

modelling functions; Holodeck (Yang et al., 2024) composes environments from retrieved assets; BlenderAlchemy (Huang et al., 2024) searches over program edits judged visually; Thinking in Blender (He et al., 2026) reconstructs a scene in stages, each closed by a generator–verifier loop; and expert tutorials have guided procedural material generation (Gupta et al., 2026). In these systems the tools, procedures and checks are written in advance. SceneCraft's outer loop does compile recurring code into a skill library, but it reads what the agent did often; ours reads what it got wrong.

Agents that grow their own tools and skills. Voyager (Wang et al., 2023) keeps executable skills that pass self-verification, SkillWeaver (Zheng et al., 2025) distils reusable APIs from web exploration, Agent Workflow Memory (Wang et al., 2024) induces workflows from past trajectories, DynaSaur (Nguyen et al., 2024) accumulates actions the agent writes itself, Toolformer (Schick et al., 2023) learns when to call tools, and ADAS (Hu et al., 2025) searches over whole agent designs. These agents also grow tools and skills around a frozen model; the difference is where an update comes from. Prior self-improving agents compile successful trajectories into reusable behaviour, and accept it, where there is a test at all, when the code runs. BLENDCUA starts from a failure in the outcome, attributes it to one harness component, and updates only that component, keeping the update only if a rebuild shows the failure gone. In 3D modelling this matters, because a script can run cleanly and still produce a visibly wrong model.

Critique and credit assignment. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) turn a model's critique of its output into revisions or remembered feedback, and trained critics sharpen such critique (McAleese et al., 2024; Bai et al., 2022). These improve the answer to the current task; we use critique for credit assignment across tasks. EvoQuality (Wen et al., 2026) builds its own supervision but updates the model's weights, whereas ours are frozen. Computer-use benchmarks such as OSWorld (Xie et al., 2024) score completion programmatically; creative 3D work has no such check, which our checker supplies.

3 METHOD

The agent that builds each task is fixed scaffolding (Section 3.1); the contribution is the loop that runs after each build and decides what the harness keeps. A checker names the defects in the finished model (Section 3.2), a curator attributes each to a cause and writes the remedy that cause demands (Sections 3.3 and 3.4), and the task is rebuilt from the step the remedy first touches, with the remedy committed only if its defect closes (Section 3.5). Algorithm 1 summarises one task.

Setting. A task T is a brief in natural language with reference images. The harness $H = (\mathcal{T}, \mathcal{S})$ holds *tools* $\mathcal{T}$, Python modules the agent can import, and *skills* $\mathcal{S}$, documents filed under the role that reads them: a planner skill constrains how a task is specified, an executor skill describes how something is built, and a reviewer skill adds a check. The agent produces a plan $P = \textsc{Plan}(T, H)$ and an artefact $A = \textsc{Build}(P, H)$, a saved Blender scene with its renders. Over a curriculum $T_1, \ldots, T_N$ the model is frozen and only the harness changes, from the hand-built H_0 to H_n, the harness after task T_n has been curated.

3.1 BUILDING WITH THE HARNESS

A planner turns the brief into ordered steps, each carrying the checklist it will be judged against. An executor carries out one step at a time, writing `bpy` code and calling the tools in H, and a reviewer checks each step against its checklist and can send it back a bounded number of times. The scene is saved after every step, so a rebuild can start from any step with everything before it unchanged. This structure is the same in every configuration we evaluate.

3.2 OUTCOME CRITIQUE

The step reviewer judges each step against the plan, so it cannot catch a property the plan never asked for. Once the build finishes, an independent checker critiques the artefact against the brief and the references alone, never the plan, so its critique reflects what is visible in the finished model rather than what the agent meant to build. It receives renders from eight azimuths, top, bottom and an

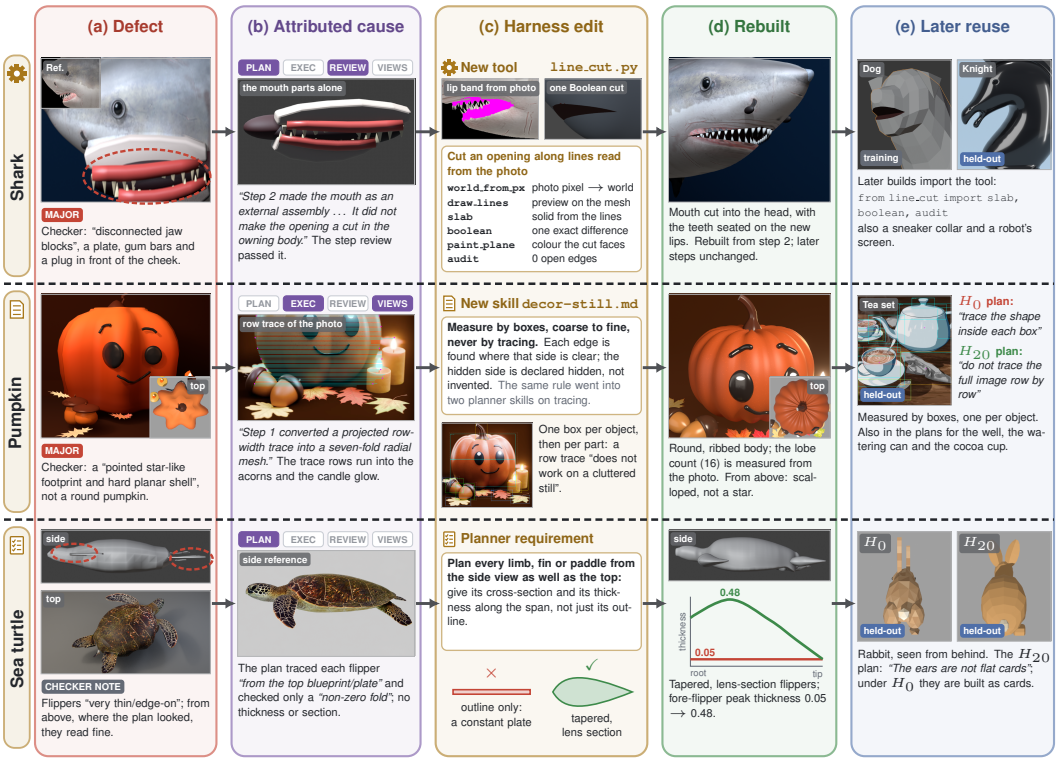


Figure 2: **From a defect in the finished model to a harness edit that later tasks use.** Each row follows one defect: (a) the defect in the finished model, (b) the part of the run it was traced to, (c) the resulting edit to the harness, (d) the rebuilt model and (e) a later task that uses the edit. The rows cover the three kinds of edit: a *tool* for an operation the harness could not perform, a *skill* for a procedure that fails on a cluttered photograph, and a *planner requirement* for a property the plan never asked for. Each edit is used beyond the task that produced it, including on held-out tasks.

untextured view, and can query the saved scene: from renders alone it missed a chair back set 0.73 units off its seat, which a scene query found at once, so every check includes a structural pass for detached, interpenetrating and loose parts. Against human defect lists on 18 earlier builds it caught 26 of 33 gradeable issues outright and 5 more partly. It reports at most two major and two minor defects per object, since unbounded it reports $2.7\times$ as many findings as a human; only majors are acted on. Each defect d carries an identifier $\mathrm{id}(d)$ so it can be recognised in a later attempt. We write $D = \textsc{Check}(A, T)$; in *verdict* mode, given D and a rebuilt artefact A', the checker marks each identifier *resolved*, *unchanged* or *worse* and scans for new major defects, $v = \textsc{Check}(A', T, D)$.

3.3 Attribution

One symptom can have several causes. When the members of a wrought-iron gate read as round hoses rather than flat straps, the plan may never have specified a cross-section, the agent may have had no way to measure member width, or it may have defaulted to a circular profile that nothing challenged. Each calls for a different repair, and telling them apart needs the build trajectory, which the checker never saw. The curator groups the major defects and assigns each group g a cause $c(g) \in \{\textsc{plan}, \textsc{review}, \textsc{execution}, \textsc{views}, \textsc{none}\}$, citing the file that shows it. It asks, in order, whether the property was asked for at all, at the right step and measurably; whether the reviewer passed it anyway; and whether the executor lacked the knowledge, lacked the capability, or missed a tool that already existed. VIEWS covers errors inherited from generated reference views, and NONE produces no edit.

Algorithm 1: Harness evolution for one task. Over a curriculum, each task starts from the harness the previous task returned.

Input: task T, harness H, round budget $R = 2$
Output: harness H
$P \leftarrow \textsc{Plan}(T, H)$; $A \leftarrow \textsc{Build}(P, H)$;
for $r \leftarrow 1$ **to** R **do**
  $D \leftarrow \textsc{Check}(A, T)$; `// plan withheld; ≤2 major, 2 minor per object`
  if *D has no major defect* **then**
    $H \leftarrow H \cup \textsc{Lessons}(A, T)$; **break**;
  end
  foreach *major defect group g in D with $c(g) \neq$ NONE* **do**
    $e \leftarrow \textsc{Remedy}(g, c(g))$; `// planner / executor / reviewer skill, or tool`
    $A' \leftarrow \textsc{Rebuild}(A, P, H \cup e, s(e))$; $v \leftarrow \textsc{Check}(A', T, D)$;
    if *e passes Eq. 1* **then**
      $H \leftarrow H \cup e$; $A \leftarrow A'$; `// commit`
    else
      $H \leftarrow H \cup \textsc{Salvage}(e, v)$; `// roll back; keep what v supports`
    end
  end
end
$\textsc{Snapshot}(H)$;

3.4 Remedy

For each attributed group the curator writes a remedy e: a local fix for this task and a durable edit to H, whose form follows from the cause. A *planning* defect produces a patched plan and a planner requirement that would have prevented it; planner skills are inlined into the planner's prompt, so they bind every later plan. A *knowledge* gap produces an executor skill. A *capability* gap produces a tool, grown additively by new functions or parameters and committed with a trigger saying when to use it. A *review* defect adds a reviewer check or a measurement the reviewer lacked. Checks are never weakened and the brief is never edited, since changing the specification to match the output would destroy the signal.

3.5 Rebuild and acceptance

A remedy is tested on the task that motivated it. The curator finds its *origin step* $s(e)$, the earliest step it would have changed, restores the scene saved before $s(e)$ and replays the build with the remedy applied, $A' = \textsc{Rebuild}(A, P, H \cup e, s(e))$. Because everything before $s(e)$ is unchanged, the remedy is the only difference between the attempts. The checker returns its verdict on the same identifiers, and

$$H \leftarrow \begin{cases} H \cup e & \text{if } v[\mathrm{id}(g)] = \textsc{resolved} \text{ and no new major defect is attributable to } e, \\ H & \text{otherwise.} \end{cases} \tag{1}$$

A new defect counts against e only if the remedy could have caused it, tested by fingerprinting each object's geometry. A rejected edit is rolled back, keeping only any part the verdict supports. Each task runs at most two rounds, and the harness is then saved as a snapshot, so consecutive snapshots differ by exactly what one task taught.

4 Experiments

We ask whether a harness grown from defects builds better than one learned from transcripts under the same agent and curriculum, whether its updates are selective and reused beyond the task that produced them, and which parts of the method and of the harness carry the gain, and whether the harness and the method depend on the model that grew them.

Table 1: Main result on the 10 held-out tasks, one build per cell. The upper block is a ladder: every row runs the same base model, and one thing changes at each step. M1 (0–5) and M2 (0–4) are judged blind; M3 (0–100) is computed from the scene; M4 counts major defects per build. CLIP and DINO are embedding similarities to the reference. Rank is the mean rank among all seven systems' builds for a task (1 = best). The last row is a production coding agent from a different model family, outside the ladder.

System	M1 ↑	M2 ↑	M3 ↑	M4 ↓	CLIP ↑	DINO ↑	Rank ↓
B1: scripting interface only	3.35	2.70	76.6	3.80	0.77	0.62	5.90
B2: + skill compaction	3.30	3.00	82.0	3.90	0.76	0.64	5.20
B3: transcript-mined curation	3.50	3.10	84.5	2.80	0.81	0.65	3.90
Ours, H_0	4.10	3.50	87.4	2.90	0.84	0.68	3.10
Ours, H_{10}	**4.35**	3.45	89.2	2.50	**0.86**	**0.72**	3.00
Ours, H_{20}	4.15	**3.60**	**92.3**	**2.00**	0.85	0.70	**2.50**
Production coding agent	3.20	2.50	79.7	3.20	0.74	0.60	4.40

4.1 SETUP

Tasks and implementation. The harness is grown on a curriculum of 20 tasks spanning modelling, materials, physics simulation and animation, among them a wrought-iron gate, a Newton's cradle with rigid-body constraints, an animated black hole, a draped tablecloth and a low-poly dog; a task takes two to three hours. We evaluate on 10 held-out tasks that curation never sees (an alien lizard bust, an office chair, a bowie knife, a chess knight, a demon cauldron, three low-poly characters, a stylised hammer and a toon tea set), with curation switched off. Every role uses GPT-5.6-luna at maximum reasoning effort, frozen throughout. We evaluate the hand-built H_0 and the snapshots after 10 and 20 curriculum tasks, H_{10} and H_{20}; over the curriculum the harness grows from 57 to 73 tool modules and from 44 to 66 skills. A build costs a median of $2.24 in API calls.

Baselines. Every ladder row uses the same model on the same test tasks. **B1** is the model with the brief, the references and a single tool that runs Python in Blender. **B2** adds skill compaction: after each of the same 20 curriculum tasks the trajectory is compressed into reusable skills, retrieved before later tasks (Wang et al., 2023; Zheng et al., 2025). **B3** is our earlier loop and the closest comparison. It runs the same planner, executor and per-step reviewer on the same curriculum and grows the same kinds of tools and skills, but its curator reads a digest of the run rather than the finished model: redo counts, outlier and rewritten actions, and failed review rounds, the record of how the run went that trajectory-learning methods learn from (Shinn et al., 2023; Wang et al., 2024). It has no checker on the artefact, no attribution and no rebuild that tests an edit, so the learning signal is the only difference. Outside the ladder we report a **production coding agent**, Claude Sonnet 5 in Claude Code with the same briefs and references, from a different model family.

Metrics. We never average measures. **M1** (brief compliance, 0–5) grades whether each required part is present and identifiable; **M2** (form fidelity, 0–4) grades proportion and silhouette against the reference; **M3** (mesh quality, 0–100) is computed from the saved scene with no model involved, penalising non-manifold and loose geometry, degenerate faces, n-gons and unapplied transforms; **M4** counts major defects found by a separate checking pass. CLIP (Radford et al., 2021) and DINO (Oquab et al., 2024) give the similarity between the reference and the best-matching turntable view. **Rank** is the mean position when the compared builds of a task are ranked side by side. The judged measures use GPT-6-luna, blind to the system; since it shares a model family with our agent, the measures with no language model (M3, CLIP, DINO) check it. We run one build per cell, and for paired comparisons count the tasks on which one system beats the other, with a two-sided sign test.

4.2 MAIN RESULTS

Defects against transcripts. The comparison that tests our claim is against B3, which holds agent, model and curriculum fixed and changes only what the harness learns from (Table 1). Learning from defects is better on every measure: H_{20} lowers major defects from 2.8 to 2.0 per build (29%), raises mesh quality from 84.5 to 92.3, compliance from 3.50 to 4.15 and form fidelity from 3.10 to 3.60,



Figure 3: Three of the held-out tasks, one build per system, in the row order of Table 1. Each build is re-rendered from its own camera with its studio backdrop removed; nothing else is changed.

and improves mean rank from 3.9 to 2.5. The measures with no language model agree (CLIP 0.81 to 0.85, DINO 0.65 to 0.70). B3 grows the same kinds of tools and skills from the same tasks; what it lacks is a signal that names what the finished model got wrong, so its edits address how the build went rather than what it produced.

Other baselines and the harness curve. In Table 1, skill compaction helps little: B2 improves on B1 in form and mesh quality but not in compliance or defects, as expected of a library that records procedures rather than faults. The production coding agent ranks between B2 and B3. Much of the gap to B1 and B2 is already present at H_0, so the comparison that isolates the loop within our agent is H_0 against H_{20}: major defects fall by 31%, mesh quality rises by 4.9 points, and H_{20} ranks above H_0 on 9 of the 10 tasks ($p = 0.02$). From H_0 to H_{10} the judged measures and embedding similarities rise (M1 4.10 to 4.35, DINO 0.68 to 0.72); from H_{10} to H_{20} compliance levels off while form fidelity improves on 7 tasks and worsens on none ($p = 0.02$; per-task counts in Appendix J), and H_{20} ranks above H_{10} on 7. Mesh quality (87.4, 89.2, 92.3) and major defects (2.9, 2.5, 2.0) improve throughout, whereas our transcript-mined loop added only three tools and two skills over its second ten tasks.

Qualitative comparison. In Figure 3, B1 and B2 give the chess knight a human profile and H_0 builds its head as two separate slabs, while H_{10} and H_{20} recover the muzzle, eye and layered mane. On the alien lizard, H_{20} reproduces the ridged neck and bust proportions the rounded baseline heads miss; on the bowie knife, H_0 and H_{20} build the curved clip-point blade where B1 and B2 build a straight wedge.

4.3 From a defect to a reusable edit

Figure 2 follows three defects through the loop. The shark's bolted-on mouth needed an operation the harness lacked, so the repair is a tool, `line_cut`, which later cuts the dog's mouth and the held-out knight's. The star-shaped pumpkin came from tracing a cluttered photograph, so the repair is a skill, and the held-out tea-set plan under H_{20} says "do not trace" where the H_0 plan says "trace". The turtle's paper-thin flippers were planned from the top view only, so the repair is a planner requirement, and the held-out rabbit's H_{20} plan asks for ears that "are not flat cards". Each edit is written for a family of objects, not the one that exposed it.

4.4 Selective and reusable updates

Because every defect is attributed before it is repaired, the curriculum yields a distribution over causes (Table 2). Execution is the largest, at 48% of defects, followed by review at 29%: the agent more often builds a named part badly, and the reviewer passes it, than it is asked for the wrong thing. The

Table 2: **Left:** cause of each of the 84 distinct defects found over the 20 curriculum tasks (a defect blamed on two parts is split between them), and the remedies for each cause accepted by the gate. **Right:** how selective and reusable the updates are over the curriculum.

Cause	Share	Accepted
Plan	16%	6/8 (75%)
Execution	48%	11/17 (65%)
Review	29%	6/15 (40%)
Generated views	7%	6/6 (100%)

Remedies proposed (28 rounds, 18 tasks)	54
Rebuilt and judged at the gate	29
Accepted by the gate, by cause	40–100%
New tools used on a later task	14/16
New skills used on a later task	14/22

Table 3: Ablations on the held-out tasks, each block compared only within itself: Rank is the mean rank among the block's builds for a task, and W–L counts the tasks on which the block's first row ranks above or below the row. The attribution block retrains the loop for 10 tasks; the others change what is kept or what is visible at test time.

Variant	M1 ↑	M2 ↑	M3 ↑	M4 ↓	Rank ↓	W–L
Acceptance gate						
Full method, H_{20}	**4.15**	**3.60**	92.3	**2.0**	**1.1**	—
+ every rejected remedy	2.90	2.10	**94.1**	3.3	1.9	9–1
Attribution (retrained to H_{10})						
Full method	**4.35**	**3.45**	**89.2**	**2.5**	**1.2**	—
Curator sees the defect, not its cause	3.50	2.65	85.3	3.7	1.8	7–3
Composition of H_{20} (masked at test time)						
Full H_{20}	**4.15**	**3.60**	92.3	**2.0**	**2.17**	—
Tools only	3.50	2.30	93.5	3.6	3.50	8–2
Skills only	3.17	2.00	**94.5**	3.6	4.50	9–1
Executor skills only	3.20	2.40	90.0	4.0	4.17	7–3
Planner skills only	3.10	2.30	93.8	3.7	4.17	8–2
Nothing ($= H_0$)	4.10	3.50	87.4	2.9	2.50	9–1

gate is most selective where attribution is hardest, accepting three quarters of planning remedies but only 40% of review remedies, which often add a check the rebuilt model still fails. Of the 54 remedies proposed, 29 were rebuilt and judged; the rest stopped when the remedy could not be applied or the rebuild failed, and none entered the harness. What is kept is reused: 14 of the 16 new tools and 14 of the 22 new skills were used again on a later task.

4.5 Ablations

The acceptance gate. Putting back every remedy the gate rejected is the costliest change we measured (Table 3): major defects rise from 2.0 to 3.3, compliance falls from 4.15 to 2.90 and form fidelity from 3.60 to 2.10. The full harness ranks higher on 9 of the 10 tasks ($p = 0.02$) and scores higher on both judged measures on 9 ($p = 0.004$). Mesh quality rises slightly without the gate (94.1 against 92.3, $p = 0.18$): many rejected remedies tidy the geometry while breaking the form, which a closure test on the targeted defect catches and a mesh audit does not.

Attribution. Retraining for 10 tasks with a curator given the defect but not its cause yields more major defects (3.7 against 2.5) and lower compliance, fidelity and mesh quality. The full method ranks higher on 7 of the 10 tasks; this is not significant ($p = 0.34$), but every measure moves the same way. Without a cause, a defect described in prose tends to be read as missing executor knowledge, even when the fix belongs in the plan or the review.

Composition. Masking parts of H_{20} shows that its tools and skills work as a unit: every partial harness is worse than the full one on the judged measures and in major defects, and worse even than H_0; tools alone lose form fidelity against H_0 on 7 tasks and win on none ($p = 0.02$). A new tool arrives with the skill that says when to use it, and a planner skill assumes the executor skill that carries out what it asks, so removing one half leaves the other pointing at nothing. The partial harnesses

Table 4: Transfer and retraining with a second model, Muse Spark 1.3. Rank is within each block. **Transfer:** a model builds with a harness grown by the other, unchanged. **Retrain:** each model runs the whole loop in every role for 10 curriculum tasks, and builds with its own H_{10}.

Builder, harness	M1 ↑	M2 ↑	M3 ↑	M4 ↓	Rank ↓
Transfer					
Muse Spark, luna's H_0	4.00	3.20	87.1	3.2	3.1
Muse Spark, luna's H_{10}	4.10	**3.65**	86.5	2.8	2.3
Muse Spark, luna's H_{20}	**4.30**	3.50	**88.4**	2.4	**2.1**
luna, Muse Spark's H_{10}	4.00	3.25	83.8	**2.0**	2.5
Retrain					
luna, own H_{10}	**4.35**	**3.45**	89.2	2.5	**1.4**
Muse Spark, own H_{10}	3.90	3.25	**91.8**	**2.2**	1.6

score higher on mesh quality only because they build simpler geometry. This is also why H_0 matters: the loop's edits extend the tools and procedures already present rather than standing alone.

4.6 Beyond one model

Transfer and retraining. A harness that only patched the quirks of the model that grew it would not help another. Handed unchanged to Muse Spark 1.3, from a different model family, luna's harness lowers Muse Spark's major defects from 3.2 at H_0 to 2.8 at H_{10} and 2.4 at H_{20} and raises compliance from 4.00 to 4.30 (Table 4); H_{10} ranks above H_0 on 7 of the 10 tasks, and luna building with Muse Spark's H_{10} has the fewest major defects in the block. Transfer shows that the harness carries, not that another model can run the method, so we also run the whole loop with Muse Spark in every role for ten tasks. Its own H_{10} is close to luna's: fewer major defects (2.2 against 2.5), higher mesh quality, slightly lower judged scores and a 4–6 split on rank. The method is not tied to the model it was developed with.

5 Discussion and Conclusion

Why defects rather than transcripts. B3 is the same system reading how the build went; BlendCUA reads what the finished model got wrong, and that change alone lowers major defects by 29%. The ablations locate the difference: without attribution the curator repairs the wrong component, and without the gate the harness keeps edits that do not close their defect. A transcript records the procedure, including the one that produced a flaw; a defect names the flaw, and credit assignment decides where the fix belongs.

Limitations. The loop needs an artefact a critic can judge from renders and the saved scene, a resumable build and defects recognisable across attempts, as Blender provides. Its main limitation is that it repairs what a defect points to but does not take conceptual leaps. A step change we saw, turning generated views into measured blueprints, was a human idea added by hand to H_0; the remedies the loop wrote for view errors, such as generating and verifying extra plates, worked within that pipeline (Appendix K). Such leaps might come from imitating human demonstrations. Ten held-out tasks with one build per cell cannot resolve small differences, and the judge shares our agent's model family; the computed measures agree with it but do not replace human judgement.

Conclusion. BlendCUA grows an agent's harness from the defects in its own finished work: a checker names each defect, a curator traces it to the responsible part of the harness and repairs it there, and the repair is kept only if a rebuild shows the defect closed. With the model unchanged, the grown harness beats both its hand-built start and a harness mined from transcripts on held-out Blender tasks, its kept tools and skills are reused, and it carries over to a second model (Section 4.6). Outcome-driven credit assignment can serve as the learning signal for an agent's harness where quality cannot be scored automatically.

REFERENCES

Yuntao Bai et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Kunal Gupta, Gaurav Joshi, Yen-Ru Chen, Seemandhar Jain, Ishit Mehta, and Manmohan Chandraker. Reflecting process expertise in procedural material generation. *arXiv preprint arXiv:2607.13318*, 2026.

Guangzhao He, Rundong Luo, Wei-Chiu Ma, and Hadar Averbuch-Elor. Thinking in Blender: Staged executable inverse graphics with vision-language models. *arXiv preprint arXiv:2606.02580*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations (ICLR)*, 2025.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. SceneCraft: An LLM agent for synthesizing 3D scenes as Blender code. In *International Conference on Machine Learning (ICML)*, 2024.

Ian Huang, Guandao Yang, and Leonidas Guibas. BlenderAlchemy: Editing 3D graphics with vision-language models. *arXiv preprint arXiv:2404.17672*, 2024.

Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Cerón Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. LLM critics help catch LLM bugs. *arXiv preprint arXiv:2407.00215*, 2024.

Dang Nguyen et al. DynaSaur: Large language agents beyond predefined actions. *arXiv preprint arXiv:2411.01747*, 2024.

Maxime Oquab et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3D-GPT: Procedural 3D modeling with large language models. *arXiv preprint arXiv:2310.12945*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Wen Wen, Tianwu Zhi, Kanglong Fan, Yang Li, Xinge Peng, Yabin Zhang, Yiting Liao, Junlin Li, and Li Zhang. Self-evolving vision-language models for image quality assessment via voting and ranking. In *International Conference on Learning Representations (ICLR)*, 2026.

Tianbao Xie et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3D embodied AI environments. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Boyuan Zheng et al. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

SUPPLEMENTARY MATERIAL

This supplementary material gives the detail the main paper leaves out, in three parts. *How a build works:* Appendix A describes how the agent plans, executes and reviews a task, and why; Appendix B follows one complete run from the brief to the delivered animation. *How the loop learns:* Appendix C traces one defect from the checker's report through attribution, remedy, rebuild and verdict to its reuse on a later task, Appendix D shows first and final attempts on seven curriculum tasks, and Appendix E gives examples of each cause and the counting behind Table 2. *Further results:* Appendix F shows the transfer builds behind Table 4, Appendix G shows builds whose deliverable is motion, Appendix H reports cost and time, and Appendices I and J give the rebuild timings and the per-task comparisons behind the reported means. *Limits and impact:* Appendix K gives the evidence for the main limitation stated in the paper, and Appendix L discusses potential negative effects.

A HOW THE AGENT BUILDS A TASK

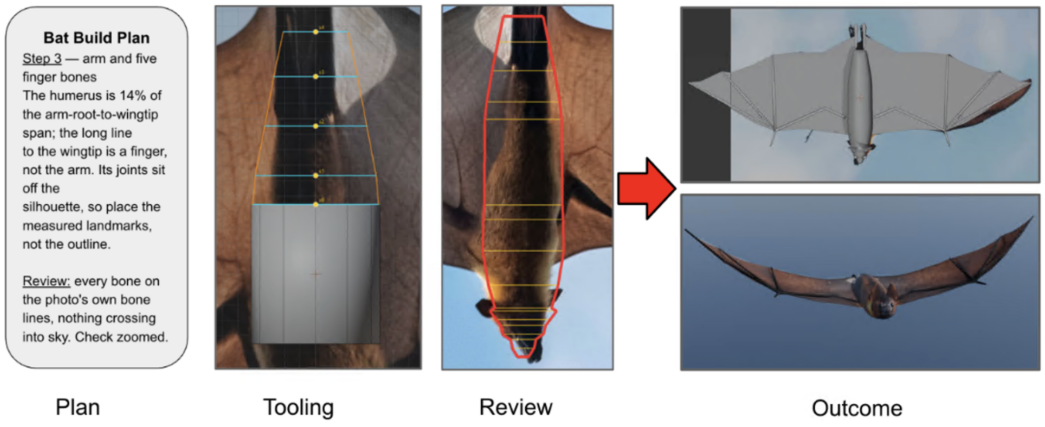


Figure 4: One step of a bat build. *Plan:* the planner states what the step builds and how it will be judged, as measured proportions rather than a description. *Tooling:* the executor places the arm's joints at landmarks measured on the reference, with the stations drawn over the photograph. *Review:* the reviewer overlays the built outline on the reference and checks the step against its checklist. *Outcome:* the finished model, in the viewport and rendered.

Figure 4 shows the three roles of Section 3.1 on one step. The planner writes each step with the checks it will be judged against, and states them as measurements the reviewer can repeat: the humerus is 14% of the arm-root-to-wingtip span, and every bone must lie on the photograph's own bone lines. The executor builds against measured landmarks drawn on the reference rather than against the outline, and the reviewer passes the step only when the built geometry, projected back onto the reference, meets those checks. A step that fails goes back to the executor with the reason. This is the structure every configuration in the paper shares; the curation loop acts on it by changing what the planner asks for, what the executor knows and has, and what the reviewer checks. Table 5 summarises the roles, and the paragraphs below give the details. Every role runs GPT-5.6-luna at maximum reasoning effort.

Table 5: The three roles that build a task. The checker and curator of the main paper run only after the build is finished.

Role	Sees	Produces	Limits
Planner	The brief and its reference image; any plates or blueprints named in the brief, read with file tools; every planner skill, inlined in full.	A plan: a toolkit section, then ordered steps, each with an objective, instructions and a review checklist.	No Blender access; 90 turns; one replan if a mechanical check of the plan fails.
Executor	One step of the plan with the plan's preamble, a handover from earlier steps, the index of tools and skills, and the result of each action.	Actions: short Python files run in the live Blender scene, each ending with a screenshot.	A fresh conversation per step; about 20 actions per step as guidance, and a hard limit on turns.
Reviewer	The reference, front and three-quarter renders of the scene, the step's checklist, the full plan and the executor's notes; it can run inspection scripts.	A paragraph per check, ending in `PASS` or `Try again` with one reason.	20 turns per review; two review rounds per step in the curriculum, one on held-out tasks.

The plan. The planner reads the brief and the reference, measures what it needs with scripts (landmark positions, part windows, contours; see Figure 5) and writes the plan as a Markdown file. The preamble names the tools the build should use; each step then has an objective, numbered instructions and a review checklist split into *important* and *other* checks. The planner is told that the reviewer is a separate agent that sees the checklist and nothing else of its intent, so the checks must be concrete: for the turtle's first step, "the supplied plates pass the identity and shape-authority gate; quote the measured bounding boxes" and "residuals … each must be under 12 px". Before the build starts, a mechanical check rejects a plan whose steps do not parse, lack checks or lack a toolkit, and warns about checks worded perceptually ("looks right"); a rejected plan is rewritten once.

Steps and actions. The executor carries out the steps in order. Each step is dispatched to a fresh conversation, so context stays small, and a short handover written after each step carries forward what later steps need. Within a step the unit of work is an *action*: one Python file, numbered and named, which is sent to the live Blender session, run, and ends by taking a screenshot. The executor gets back whether it ran, its note or traceback, and, when it asks, the new screenshots as images. Analysis that does not change the scene, such as measuring a plate or drawing an overlay, is run as a separate script. Tools are imported as Python modules and skills are fetched by name from an index, so the executor's prompt carries only what each entry is for, not its body. Rewriting an action that already ran rewinds the scene to before it and replays the step up to that point, so the saved record is always the sequence of actions that produced the scene.

Review. When the executor reports the step done, the reviewer judges it. It receives fresh renders of the scene, the step's checklist and the executor's notes, and it can write its own inspection scripts: measure a mesh, project it onto a plate, count open edges. It writes one paragraph per check and ends with `PASS` or `Try again` and a reason, which goes back to the executor as a fix round. The executor's own claim that it is done is not authoritative; only the reviewer's verdict ends a step. If the last round still fails, the executor gets one corrective pass, the step is marked exhausted and the build moves on, so a hard step cannot stall the task. In the turtle's first run, for example, the first review of step 1 returned "Try again" with a registration residual of 64.6 px against the 12 px limit, and the second passed.

Checkpoints. The scene is saved at the end of every step, whatever the verdict. The saved scenes are what make the rebuild of Section 3.5 cheap (Appendix I): a remedy is tested by loading the scene before its origin step and replaying from there. After a rebuild, a hash of each object's evaluated geometry is compared with the original, which is how a new defect is ruled in or out as the remedy's doing.

Why it is organised this way. Three choices follow from failures we saw. First, a model has no prior trajectory for a long Blender build, and an agent left to find one wanders; a plan written before execution supplies it, and splitting it into steps lets each step start with a small context and end in a saved scene. Second, reviews are unreliable when they are asked whether something looks right and repeatable when they are asked something concrete, so the checks are written at planning time, as measurements, by the role that knows the intent. Third, the step must be judged by someone other than its builder: when the executor's own pass ended a step, a failed review was followed by one

unverified edit and recorded as a pass, with a muzzle about three times too wide. More review rounds did not help either, as later rounds mostly repeated the objection of the one before, which is why there are at most two.

B ONE RUN FROM BRIEF TO DELIVERY: THE SEA TURTLE

Figures 5 and 6 follow curriculum task 15, a green sea turtle, through the run that ended the task. The brief supplies a photograph and three orthographic plates (top, side and bottom) and asks for the turtle modelled and textured from the plates, four flippers traced and built as separate pieces, and a rig with a swimming loop. The plan has seven steps, the last an underwater shot of the loop, and the figures show each one with the evidence the agent produced for it: the measurements and overlays it made to decide what to build, and the checks the reviewer ran before letting the step pass. Two panels are screenshots of the Blender session itself, with the step's own code in the text editor; the solid-shaded and swimming views are re-rendered from the scene the agent saved at the end of that step. Nothing in the scene was edited for the figure.

Three things the figures make concrete. First, most of the agent's effort goes into measuring rather than modelling: before the body is lofted, the planner has already read landmark positions, part windows and flipper contours off the plates, and every later step is checked by projecting the model back onto the plate it was built from. Second, the reviewer's checks are numeric where they can be, for example a registration residual below a 12-pixel gate or a body mesh with no open edges, so a step fails for a reason the executor can act on. Third, each step leaves a saved scene, which is what allows the curation loop to rebuild from the step a remedy first touches instead of from the start.

C ONE DEFECT FROM REPORT TO REUSE: THE MONSTERA LEAVES

Figure 7 follows one defect through the whole loop, on curriculum task 3, a potted monstera. It is chosen because every stage left a clear record and the remedy was later used on a different task.

The defect. The first build looks close to the reference when textured, but the checker, which inspects an untextured Solid view and the scene itself, found that each leaf was a flat rectangular card of 25 vertices with the leaf shape carried only by an alpha-masked image. It reported this as its worst major defect, `leaves.geometry.flat_card_planes`, together with a sibling defect for the lack of variation between leaves.

Attribution. Reading the build trajectory, the curator traced the cards to the plan: step 1 prescribed an alpha-textured card over a rough cage, and step 4 copied it, so the executor had done what it was asked. It named review as a second cause, since step 4's review passed the textured render without checking the mesh in Solid view. A first round had tried a wording-only change to the plan; the gate dropped it, because the rebuilt leaves were still cards.

Remedy. In the second round the curator judged that no existing tool could build a leaf with real cut-throughs, a capability that recurs in other flat objects, and so wrote a tool, `fenestrated_lamina`, which builds a thin, folded leaf blade from an outer contour and a set of opening contours. It paired the tool with a planner-skill section saying when to plan such a blade instead of an image card ("the image remains a colour/vein source only") and extended a reviewer skill. Because the fault began in the plan's first step, the origin step was step 1, and the task was rebuilt from there.

Verdict and reuse. On the same identifiers, the checker marked both leaf defects resolved: the blades became irregular lobed meshes of 162–200 vertices, still visibly faceted, and no new major defect appeared. The remedy was committed and appears in every later harness snapshot. Twelve tasks later, the sea turtle's planner named the tool for the flippers; its first runs built the flippers with it, the reviewer used the tool's audit function to check them, and the final run, which gives the flippers a thicker lens-shaped section, still takes their fold from the tool (Figure 6, step 4). The gate is not a guarantee of an overall improvement: in the same rebuild two other defects, the leaves' edge-on depth and the pot's proportions, were marked worse, and a separate remedy for the trunk's root attachments was rolled back.

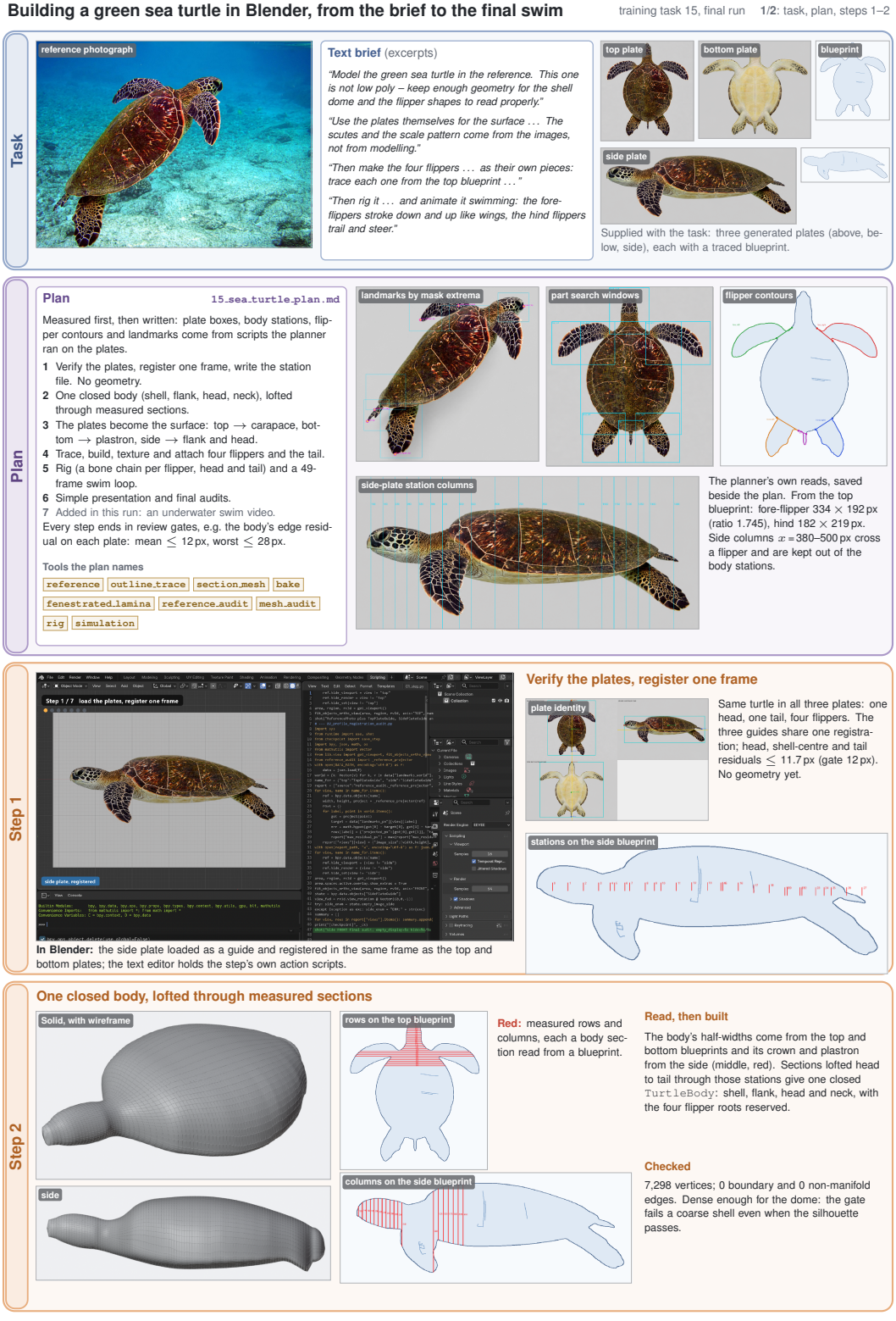
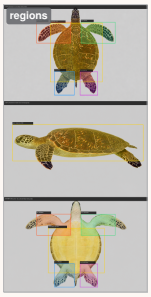


Figure 5: Building a green sea turtle (curriculum task 15), part 1: the task and its plates, the plan with the planner's own measurements, step 1 (register the plates, shown in the Blender session) and step 2 (the body, built from measured stations and checked against the plates).

Building a green sea turtle in Blender (continued) 2/2: steps 3–7 and the result

Step 3

The plates become the surface







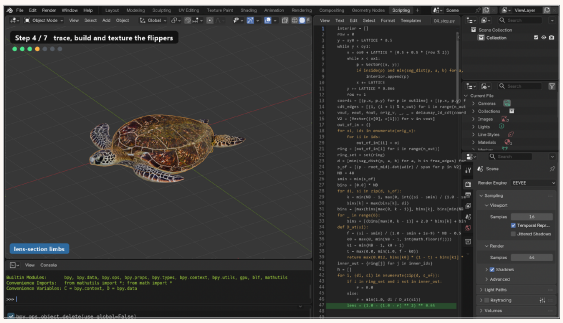


Scutes and scales are image pixels, not geometry. Top-plate pixels go on the carapace, bottom-plate pixels on the plastron, side-plate pixels on the flank, head and neck; each face takes its plate by region and normal.

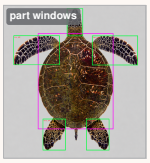
Left: every region cut from its plate and drawn back on it, the check that the texture lands on its owning surface.

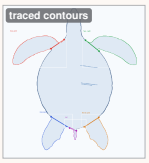
Step 4

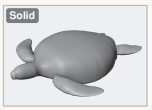
Four flippers and the tail: traced, built, textured, attached

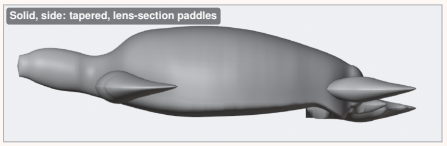


In Blender: the finished limbs; the editor shows the step's limb builder at the thickness field, $h = H(s) \cdot \mathrm{lens}(d/D(s))$ with $\mathrm{lens}(r) = (1-(1-r)^2)^{0.65}$. About 3,300–3,500 vertices per fore flipper.









Each flipper follows its own top-blueprint contour, is built as a tapered paddle with a lens-shaped section (thickest over the proximal third, 0.48 at its peak on a 2.0 span), has its root buried in the body, and is textured with its own crops of the top and bottom plates.

Step 5

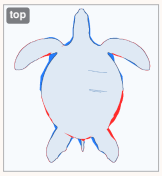
Rig and a 49-frame swim loop

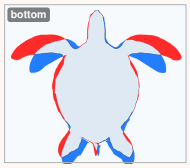


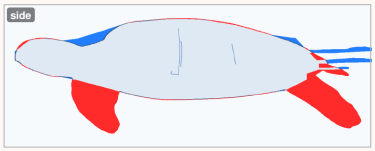
Seen head-on. 14 bones in six chains: one per flipper, head and tail. The fore pair strokes down (frame 13) and up (frame 37) like wings; the hind pair trails and steers. Frame 49 returns to frame 1.

Step 6

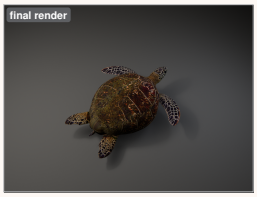
Presentation and final audits







The finished model, all six meshes, projected onto each blueprint: red blueprint only, blue model only; the rest is covered by both.



Step 7

Underwater swim video (added in this run)



A separate scene links the same rig and meshes. The seabed, rocks and marine snow drift past while the turtle swims, under a water volume with light shafts, and caustics play on the sand. 240 frames at 1280 × 720 and 24 fps: ten seconds, five full strokes.

Figure 6: Building a green sea turtle, part 2: step 3 (texture from the plates), step 4 (flippers traced and built, shown in the Blender session), step 5 (rig and swim loop, five head-on frames), step 6 (the finished model projected onto each plate) and step 7 (the underwater animation).

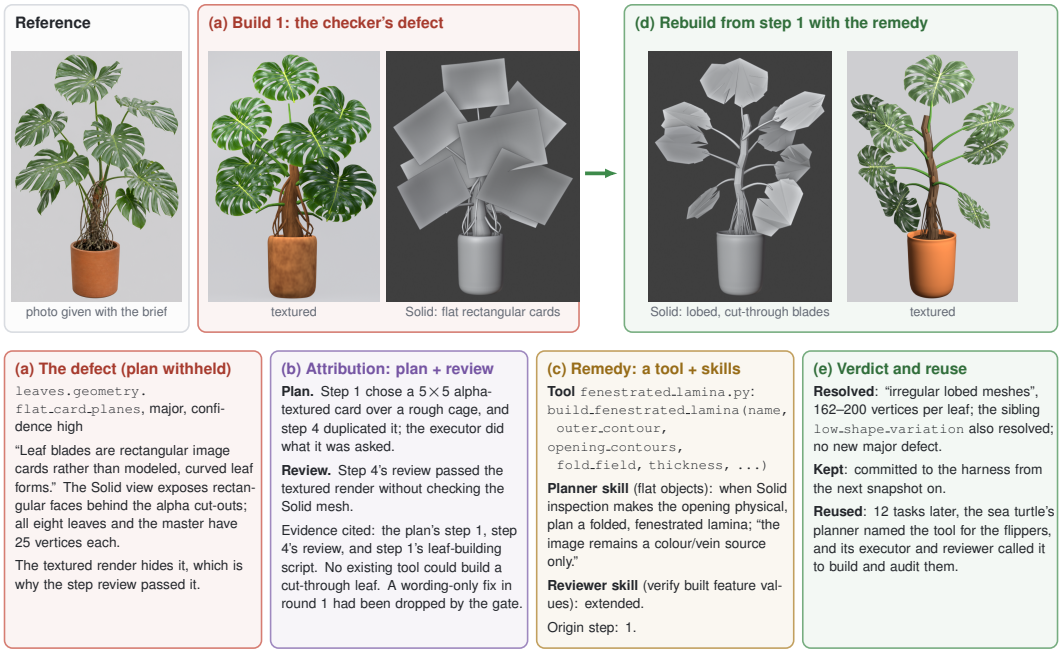


Figure 7: One curated defect traced end to end (curriculum task 3). *(a)* The first build and the checker's defect: the leaves are flat image cards, visible only in the Solid view. *(b)* The curator's attribution, from the build trajectory. *(c)* The remedy: a new tool and two skill edits, with step 1 as the origin step. *(d)* The rebuild from step 1, the same views. *(e)* The checker's verdict on the same identifiers, the commit, and the tool's reuse on a later task.

D First and final attempts on curriculum tasks

Figure 8 shows seven curriculum tasks, each with its first attempt, built under the harness of that point in the curriculum, and its final attempt, after the rebuilds that tested remedies. Each build is shown from its own camera with its studio backdrop removed. The changes between attempts follow the remedies of Appendices C and E: the monstera's leaves go from flat cards to cut-through blades, and the hammer's head moves toward the reference's pale stone with gold bands. Not every intermediate rebuild was better; those whose remedies the gate rejected were rolled back.

E Where the defects come from: examples and counting

The cause shares in Table 2 count 84 distinct defects (a task and a defect identifier) over the 20 curriculum tasks, each at its first attribution. The curator ran 28 rounds over 18 tasks; the other two, the low-poly dog and the Newton's cradle, produced no curated defect. Of 138 attributions over all rounds, 94 name more than one cause; the main paper splits such a defect evenly between its causes. Counted instead once under every cause it names, execution is still the largest cause (69% of defects, against 57% for review, 30% for plan and 11% for generated views), and no defect was attributed to NONE. Of the 54 remedies the curator chose, 29 reached a gate verdict; the other 25 stopped earlier, when the remedy could not be applied, the rebuild failed, or the round ended before the verdict, and none of them entered the harness.

Table 6 gives two defects for each cause, as the checker reported them, with the evidence the curator cited, the remedy it wrote and the gate's verdict. No defect was attributed to review alone: in every case the reviewer passed a step that also had a planning or execution fault, so the review examples carry a second cause.

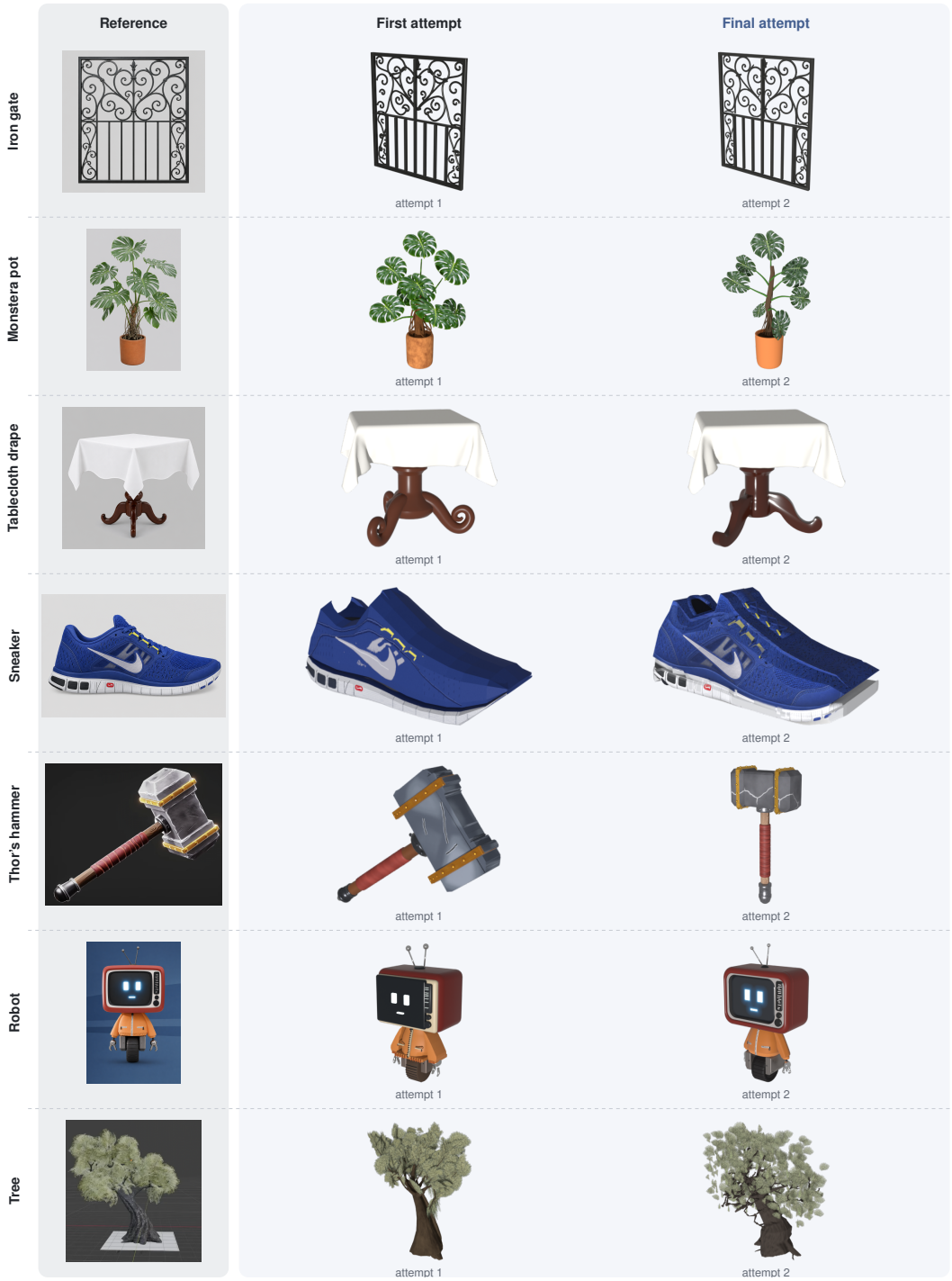


Figure 8: First and final attempts on seven curriculum tasks, beside each task's reference. Each build is rendered from its own camera with its backdrop removed; nothing else is changed.

Table 6: Two curriculum defects per cause. *Evidence* is the file the curator cited from the build trajectory; *verdict* is the acceptance gate's decision on the rebuild (kept: committed to the harness; dropped: rolled back).

Task	Defect (checker)	Attribution (evidence)	Remedy written	Verdict
Plan				
Sneaker	Upper is an angular low-poly wedge, not a rounded padded running-shoe upper.	Step 1 of the plan prescribes a 12–16-station faceted loft and never requires the padded envelope; the executor followed it.	Planner skill (trace the reference): save a boundary polygon, add rings where curvature changes, gate column by column against the mask.	dropped
Iron gate	Scrolls are rounded swept tubes, not flat strap iron.	The plan recommends a tube generator and calls an 8-sided tube a "starting approximation"; no strap profile or joint rule (plan + review).	Tool `flat_strap_network` with an executor skill: sweep one shallow strap profile over a network with shared nodes.	kept
Execution				
Desk chair	Chair sits right of and behind the desk instead of tucked at the right pedestal.	The final step's handover script hard-sets the chair root to a wrong pose although the plan gave the overlap anchor.	Executor skill: choose the pose under the final camera using both projected overlap and collision clearance.	kept
Datestone	Stone too pale and cream against the reference's muted grey-beige.	The texturing script writes the colour map from a fixed base with no rendered check; the plan's target was present.	Executor skill: sample face medians under the final camera and tune the material until they match.	dropped
Review				
Tree	Trunk and root flare too smooth and conical; no twisted ridges or buttresses.	The final review passed without repeating the plan's grazing-light and buttress check; the executor had built capped rib tubes (execution + review).	Reviewer skill adding a fused-trunk construction check, and an executor skill update.	kept
Robot	Jacket far too tall for the TV head.	Step 3's review passed on bounding boxes and never measured jacket height against head height; the plan omitted the ratio (plan + review).	Reviewer skill: measure the jacket-to-head height ratio before a pass.	dropped
Generated views				
Thor's hammer	Stone head too deep in the three-quarter view.	Only front and photo measurements existed; the depth scale was guessed from the projection, with no side or top plate.	Planner skill: generate and verify a side or top plate before choosing a depth.	kept
Monstera	Leaves face one direction and collapse edge-on from two compass views.	The plan left hidden depth to be chosen in 3D, with no verified front, side or top plates.	The same planner skill, plus a missing-axis canopy check for the reviewer.	kept

F TRANSFER BETWEEN MODELS: PER-TASK COMPARISONS AND BUILDS

The transfer block of Table 4 hands the harness grown by GPT-5.6-luna, unchanged, to Muse Spark 1.3 at each of its three snapshots, and in the other direction has GPT-5.6-luna build with the H_{10} that Muse Spark grew when it ran the whole loop itself. Nothing is retrained in either direction. Table 7 gives the per-task comparisons behind it.

Table 7: Per-task comparisons behind the transfer block of Table 4. Each entry counts the held-out tasks on which the first system is better, worse or tied (for rank, better and worse only); p is a two-sided sign test over the untied tasks, shown where it is below 0.4.

Comparison	Rank	M1	M2
Muse Spark: luna's H_{10} vs. luna's H_0	7–3 (0.34)	6–2–2 (0.29)	7–1–2 (0.07)
Muse Spark: luna's H_{20} vs. luna's H_0	6–4	5–2–3	6–3–1
luna on Muse's H_{10} vs. Muse Spark on luna's H_{10}	5–5	3–2–5	3–4–3

The harness carries across models. With luna's H_{10} or H_{20}, Muse Spark improves on the hand-built H_0 in brief compliance, form fidelity, major defects and rank: defects fall from 3.2 to 2.8 and 2.4 per build. Task by task, the gain is clearest at H_{10}, which ranks above H_0 on 7 of the 10 tasks and has better form fidelity on 7, worse on 1. At H_{20} the counts are smaller (6–4 on rank, 6–3–1 on form), although its means for compliance, mesh quality and major defects are better than H_{10}'s, so on this model the evidence for H_{20} over H_{10} is mixed. The two crossed H_{10} builds are level task by task (5–5 on rank, 3–2–5 on M1, 3–4–3 on M2); in the means, luna on Muse's harness has fewer major

defects (2.0 against 2.8) and Muse on luna's has higher form fidelity (3.65 against 3.25). Figure 9 shows four of the held-out tasks built by each row.

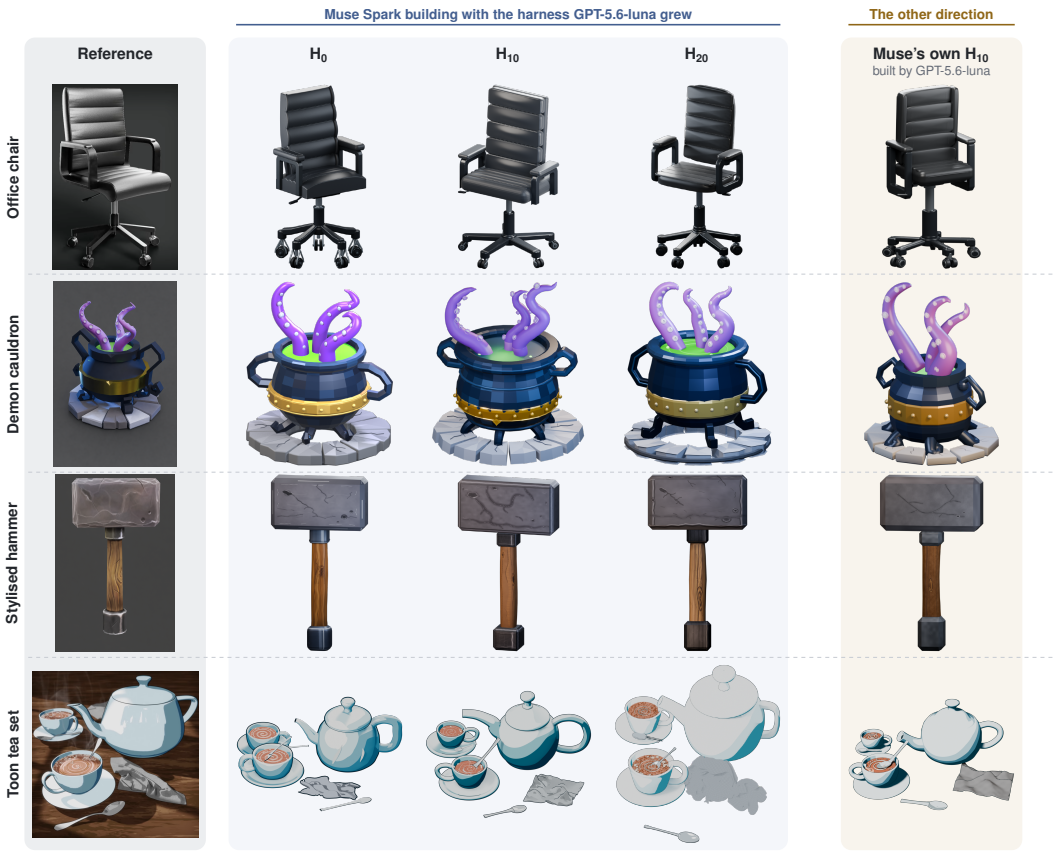


Figure 9: Four held-out tasks, one build per row of the transfer block of Table 4, in the same order. Each build is re-rendered from its own camera with its studio backdrop removed; nothing else is changed. References are shown as given to the agent.

G Animation and simulation

Several tasks deliver motion rather than a still model: a rigged character with a walk cycle or an attack loop, or a physics simulation. A single render cannot show whether such a deliverable works, so Figure 10 shows five frames of each, spaced to cover the action. The two characters are held-out tasks built with H_{20}; the Newton's cradle is a curriculum task whose balls are rigid bodies hung on constraints, so its motion is simulated rather than keyframed. In every case the agent built the rig or the simulation, animated it and chose the camera itself; the brief specifies only the motion (for example, "a wind-up that raises the axe back over the right shoulder, a hard diagonal swing down and across the body, a short follow-through, and a return to the guard pose").

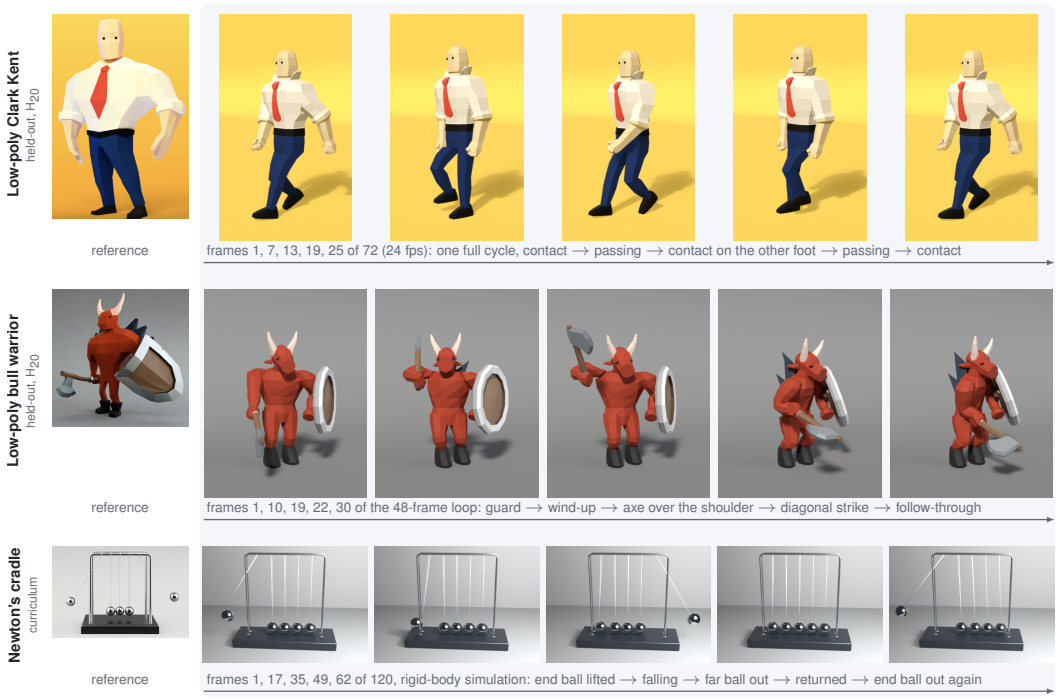


Figure 10: Frames from three animated deliverables, rendered from each delivered scene with its own camera, lights and materials. Frame numbers are chosen from the motion itself: one full walk cycle, one axe swing from guard to follow-through, and the cradle from the lifted end ball to its return.

H COST AND TIME

Table 8 gives the cost and wall-clock time of every build with a complete log: one build of each of the 10 held-out tasks and all curriculum builds, including the rebuilds the loop ran to test remedies. Cost is the API price of all model calls in the build (step subagents, reviewers and the supervising executor) at GPT-5.6-luna's current API prices; wall-clock time includes Blender's execution and rendering. Table 9 gives the same for the stages outside the build: planning, checking, curation and applying a remedy.

Table 8: Cost (US$) and wall-clock time (hours) per build. A curriculum task's total adds its original build and every rebuild. Held-out builds allow one review round per step, curriculum builds two.

		Cost ($)			Time (h)		
Builds	n	Median	Mean	Total	Median	Mean	Total
Held-out tasks	10	2.29	2.27	22.7	2.75	3.03	30.3
Curriculum, complete builds	30	2.21	2.24	67.3	2.98	3.10	92.9
Curriculum, all builds and rebuilds	42	2.02	2.05	86.1	2.52	2.85	119.7
Curriculum, per task (all its builds)	20	4.15	4.31	86.1	5.29	5.98	119.7

Table 9: Cost and time of the stages outside the build, per call, over all tasks.

Stage	Calls	Median $	Median min	Total $
Planner	42	0.20	13	8.48
Checker	34	0.13	14	4.78
Curator	29	0.14	10	4.96
Applying a remedy	25	0.12	8	2.93

A typical build has 5 steps and about 70 actions. A step costs a median of $0.42 and takes 34 minutes on the held-out tasks ($0.47 and 38 minutes in the curriculum), and an action about $0.03 and 2.6–2.8 minutes. A median build reads about 60M input tokens, 97% of them from the prompt cache, and writes about 0.6M output tokens including reasoning; each action resends a long and mostly unchanged context, so caching is what keeps the cost low. The build dominates the cost of learning: planning, checking and curation together cost $0.22–1.80 per curriculum task, $19.0 over the curriculum, against $86.1 for its builds and rebuilds. Growing H_{20} from H_0 therefore cost about $105 and 144 hours of wall-clock time in total, and evaluating a harness on the 10 held-out tasks costs about $23.

I REBUILD SCOPE

A remedy is tested by resuming the build at its origin step, the earliest step it would have changed, with every earlier step restored from its saved scene (Section 3.5). This keeps the remedy the only difference between the two attempts; it is also cheaper than rebuilding from step one, which we measure from the logs of the main training run (Table 10). Of its 18 rebuilds, 15 resumed past step one. Those took 6.8 hours, against an estimated 44.6 had they restarted at step one, a saving of 85%. Over all 18 rebuilds the loop spent 15.7 hours instead of 53.5, $3.4\times$ less.

Table 10: Wall-clock of the 18 rebuilds in the main training run, as run and as estimated had every rebuild started at step one.

Rebuilds	Count	Actual (h)	From step one (h)	Saved
Started past step one	15	6.8	44.6	85%
Started at step one	3	8.9	8.9	0%
All	18	15.7	53.5	71% ($3.4\times$)

J PER-TASK COMPARISONS

The main paper reports means over the 10 held-out tasks. Table 11 gives the per-task comparisons behind them (the transfer comparisons are in Table 7): for each pair, the number of tasks on which the first system is better, worse or tied, with a two-sided sign test over the untied tasks where it is below 0.4. Rank has no ties.

Table 11: Per-task comparisons on the held-out tasks, first system against second, as better–worse–tied (rank: better–worse). p is a two-sided sign test over the untied tasks, given where it is below 0.4; bold marks $p < 0.05$.

Comparison	Rank	M1	M2	M3	M4
Harness curve					
H_{10} vs H_0	6–4	5–3–2	2–3–5	4–5–1	
H_{20} vs H_{10}	7–3 (0.34)	5–2–3	**7–0–3** (0.02)	6–4	
H_{20} vs H_0	**9–1** (0.02)	7–2–1 (0.18)	6–1–3 (0.13)	7–3 (0.34)	
Acceptance gate					
+ rejected remedies vs H_{20}	**1–9** (0.02)	**0–9–1** (0.004)	**0–9–1** (0.004)	7–2–1 (0.18)	0–5–5 (0.06)
Attribution (both at H_{10})					
no attribution vs full	3–7 (0.34)	2–7–1 (0.18)	1–3–6	4–6	1–5–4 (0.22)
Composition of H_{20}					
full vs tools only	8–2 (0.11)	7–1–2 (0.07)	**8–0–2** (0.008)		
full vs skills only	**9–1** (0.02)	4–0–2 (0.13)	**8–0–2** (0.008)		
full vs executor skills only	7–3 (0.34)	**8–1–1** (0.04)	**7–0–3** (0.02)		
full vs planner skills only	8–2 (0.11)	**8–0–2** (0.008)	**7–0–3** (0.02)		
full vs none (H_0)	**9–1** (0.02)	7–2–1 (0.18)	6–1–3 (0.13)		
executor only vs planner only	3–3	2–1–7	2–1–7		
tools only vs none (H_0)	2–4	2–7–1 (0.18)	**0–7–3** (0.02)		
Retraining with Muse Spark					
Muse, own H_{10} vs luna, own H_{10}	4–6	4–1–5 (0.38)	3–2–5	7–3 (0.34)	5–3–2

K LIMITATION: THE LOOP DOES NOT MAKE CONCEPTUAL LEAPS

The loop repairs what a defect points to. Each remedy it writes answers one attributed fault: a planner requirement the plan lacked, a procedure the executor did not know, a tool for a capability it did not have, or a check the reviewer did not run. What it has not done is change how the agent approaches a whole class of tasks.

The clearest example is the one change we made by hand before the curriculum began (Figure 11). An earlier version of the harness could already generate extra views of an object with an image model, but the agent used them only as pictures and kept building from the reference photograph by eye; on Thor's hammer it produced a dark head that departs from the reference's pale stone. We added to H_0 a tool that turns each generated plate into a measured blueprint (a contour with a known scale, from which stations and widths can be read) and rewrote the planner skill for generating plates. The idea came from watching artists model animals from orthographic blueprints in video tutorials. With it, the plan measures before it builds, and the first attempt on the same task already has the reference's pale stone head, gold bands and wrapped handle. Blueprints are part of every harness in this paper from H_0 on; the turtle of Appendix B is built from them.

The remedies the loop wrote for view errors that we inspected stayed inside that pipeline rather than changing it: Table 6 shows two, both of which ask the planner to generate and verify an extra side or top plate before choosing a depth. Only one of the six view remedies that reached the gate was kept. A change of the blueprint kind requires seeing that the representation, not the procedure, is wrong, and importing a technique from outside the task; neither the checker's defect nor the build trajectory contains it. We expect such leaps to need a different source of signal, for example imitation of human demonstrations or retrieval from tutorials, alongside outcome critique.



Figure 11: A step change that came from a person, not from the loop. With an earlier harness, the agent built Thor's hammer from the reference by eye. Before the curriculum we added, by hand, a tool that turns a generated plate into a measured blueprint; with it, the first attempt already matches the reference far more closely. The loop's own remedies for view errors worked within this pipeline.

L Broader impact and potential negative effects

Work and authorship. An agent that builds, textures, rigs and animates models from a brief and a picture automates part of the work of 3D artists, and the harness it grows encodes technique learned from their practice (the blueprint workflow of Appendix K came from artists' tutorials) without crediting it. We see the likelier near-term use as drafting and blocking-out inside an artist's workflow, but the same system could reduce paid work for junior modelling roles.

Reproducing protected designs. The agent reproduces whatever the reference shows, and several of our tasks are recognisable characters and props. Nothing in the loop checks whether a reference may be copied, so the method makes it cheaper to produce derivative assets of protected designs; deployments should add that check at the brief, not at the model.

Self-modifying code. The curator writes Python tools that later run with the agent's permissions in Blender and a shell. The acceptance gate tests whether a tool closes a defect, not whether it is safe, so a harness grown this way accumulates code no person has reviewed. We ran every build and curation on dedicated machines, kept each harness as a snapshot that can be rolled back, and forbade the curator from editing briefs or weakening reviewer checks, since otherwise the cheapest way to close a defect is to change what counts as one. Applying the method in less contained settings, with network access or real user data, would need code review or sandboxing of every committed tool.

Evaluation by models. Both the checker that drives learning and the judge that scores the results are language models. A loop optimised against a model critic can learn to satisfy the critic rather than the brief; our computed measures (mesh quality, CLIP, DINO) agree with the judge here, but they cover only part of what a person would check.

Cost. Growing H_{20} took about 144 hours of wall-clock time and $105 of API calls (Appendix H), and a single build reads about 60M input tokens. Most of that is served from the prompt cache, but the energy cost of long agent runs is not negligible, and methods that learn by rebuilding multiply it.

M Use of large language models

We used large language models as assistants in preparing this work: for coding assistance (writing and debugging experiment scripts, analysis and figure code), for brainstorming ideas, and for refining the writing of the paper, including editing for clarity, spelling and grammar. All ideas, experiments, results and text were reviewed and verified by the authors, who take full responsibility for the content. Separately, language models are the object of study: the agent, the checker, the curator and the judge described in the paper are language models, as reported in Section 4.1.