AdaCraft: Evolving Reliable Skills for Articulated 3D Generation

Anonymous authors
Paper under double-blind review

Abstract

Generating a 3D articulated asset requires geometry that remains coherent as its parts move. Recent systems use language-model agents to write construction programs, inspect the resulting assets, and repair errors using execution feedback. Yet, errors in joint frames, clearance and attachment recur across objects. Retaining successful corrections as reusable construction skills offers a way to improve future generations. However, a correction that benefits one mechanism can harm another, and subsequent skill revisions can undo earlier improvements. We introduce AdaCraft, a framework that automatically evolves construction skills from execution feedback while keeping the generation model frozen. Its Motion Memory stores tested repairs together with the mechanical conditions under which they apply. Within this memory, a Homological Motion Signature summarizes connectivity between required poses across clearance levels on a sampled motion graph and identifies the configurations that limit clearance along selected routes. An Adaptive Edit Budget controls how many rules may change using variation in improvements across objects, execution uncertainty, geometric dependencies, and motion clearance, effectively restricting revision scope when improvements transfer inconsistently or required motions have little spare clearance. Across 240 test tasks spanning new object instances, categories, and mechanism compositions, AdaCraft achieves 75.1% functional yield compared with 58.1% for the strongest baseline, while requiring 41.9% fewer deployment model calls per functional asset.

1 Introduction

An articulated 3D asset must remain coherent throughout its intended motion. A cabinet can appear plausible when closed while its door rotates around an incorrect pivot, intersects the frame during opening, or separates from its hinge. Such failures make an otherwise convincing asset unsuitable for interaction. Previous work has investigated part structure, geometric consistency, and motion for 3D generation and interaction (Liu et al., 2023a; Yu et al., 2026a;b; Xiang et al., 2020; Cao et al., 2026a). Recent generation agents further combine construction programs, geometric tools, and execution feedback to revise their outputs (Zhou et al., 2026; Le et al., 2025). An asset can pass checks at a few poses yet collide between them, and changing one part can disrupt the motion of another. Producing a functional articulated object therefore calls for persistent optimization over geometry, structure, and motion, where each repair builds on previous corrections while preserving earlier capabilities.

Reusable skills offer a way to retain this optimization across objects. A procedure for checking a hinge frame and its opening clearance could help construct cabinets, microwave doors and hinged containers. Existing 3D systems demonstrate the value of skill libraries and reusable construction motifs (Xu et al., 2026a; Kang et al., 2026; Wang et al., 2026). General-purpose agents also retain reflections and evolving context to guide later attempts (Shinn et al., 2023; Zhang et al., 2026). However, a reusable skill must not only capture what worked, but also when it worked and which constraints must be preserved. As the skill is applied to new objects, execution feedback provides opportunities to refine this knowledge. The challenge is to incorporate such feedback without overwriting previously useful guidance or making changes that generalize poorly across objects.

Skill evolution therefore requires both ***stability*** and ***update adaptivity***. Stability means that later revisions preserve useful construction knowledge rather than overwrite successful corrections

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

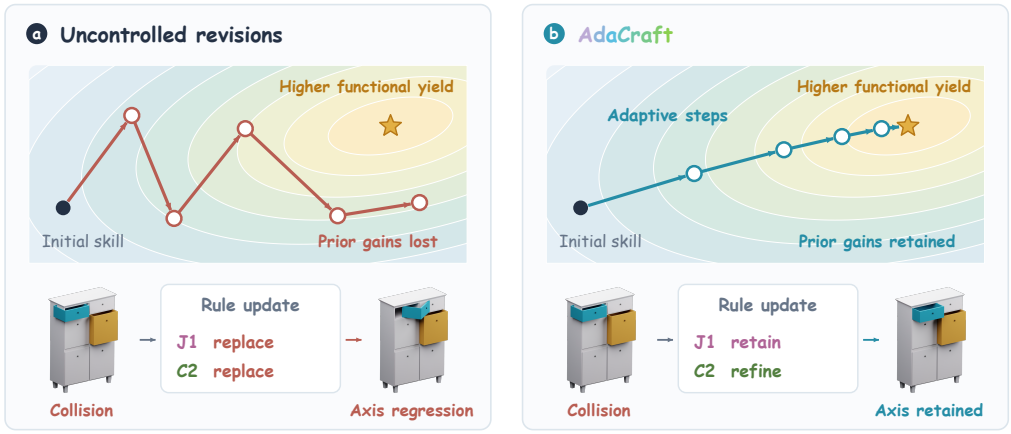

Figure 1: **Preserving successful rules during skill revision.** Nodes represent skill libraries and arrows represent revisions on a schematic functional yield landscape. (a) Replacing both joint rule J1 and clearance rule C2 can undo a correct joint axis. (b) AdaCraft retains J1 and refines C2 using stored repair evidence and an adaptive edit limit. The paths illustrate the analogy to Adam, where accumulated feedback guides update direction and scales update size.

from earlier iterations. Update adaptivity means that the extent of revision should depend on how consistently a proposed change improves different objects and configurations. Existing skill optimizers already use reflection, memory, and controlled edits (Li et al., 2026a; Yang et al., 2026; Chen et al., 2024a; Ni et al., 2026), but articulated generation demands additional reasoning about how changes to part geometry and coordinate frames affect motion across connected parts.

Therefore, we propose AdaCraft for stable and adaptive skill evolution in articulated asset generation. *Motion Memory* stores tested repairs with their part relations, joint frames, diagnostic records and the conditions under which they apply. Within this memory, a *Homological Motion Signature* summarizes the clearance available along sampled routes between required poses and identifies the configurations that limit it. An *Adaptive Edit Budget* adjusts the number of rules that may change according to variation in functional improvements across compatible objects.

Our contributions are threefold.

- We introduce **AdaCraft**, a framework that evolves reusable construction skills for articulated 3D generation from execution feedback. Its **Motion Memory** retains tested repairs together with the mechanical conditions under which they apply. Within this memory, a **Homological Motion Signature** summarizes connectivity between required poses across clearance levels on a sampled motion graph and identifies the configurations that limit clearance.
- We develop an **Adaptive Edit Budget** that controls how many rules may change in each skill revision. It combines variation in functional gains across compatible objects, execution uncertainty, geometric dependencies, and a penalty for low motion clearance to restrict revision scope when improvements transfer inconsistently or required motions have little spare clearance.
- Across 240 test tasks covering unseen object instances, categories, and mechanism compositions, AdaCraft achieves 75.1% functional yield, outperforming the strongest baseline by 17.0 percentage points while using 41.9% fewer deployment calls per functional asset.

2 Related Work

Skill evolution and memory in agents. Agents combine reasoning, tool use and iterative feedback (Yao et al., 2022; Schick et al., 2023; Madaan et al., 2023; Shinn et al., 2023), and consolidate experience into reusable guidance (Wang et al., 2023; Chen et al., 2024a; Wang et al., 2024b; Ni et al., 2026; Zhao et al., 2024; Alzubi et al., 2026). Prompt and agent optimization revise instructions, pipelines and workflows (Yang et al., 2024a; Pryzant et al., 2023; Yuksekgonul et al., 2025; Agrawal et al., 2026; Khattab et al., 2023; Hu et al., 2025; Zhang et al., 2025), while skill optimizers regulate revision scope and applicability (Li et al., 2026a; Yang et al., 2026; Gautam et al., 2026). Persistent memory supports retrieval, reflection and structured knowledge updates (Park et al., 2023; Zhong

et al., 2024; Packer et al., 2023; Kang et al., 2025; Xu et al., 2026b; Zhang et al., 2026), and continual learning addresses capability retention (Wang et al., 2024a). We couple skill revision with mechanical applicability and motion evidence while keeping model weights frozen.

Articulated generation and motion connectivity. Recent advances in 3D asset generation have substantially improved geometric quality and controllability (Poole et al., 2022; Lin et al., 2023; Liu et al., 2023b), while articulated generation extends this setting by explicitly modeling part structure, joints and motion (Lei et al., 2023; Liu et al., 2024; Su et al., 2025; Chen et al., 2024b; Liu et al., 2023a; Li et al., 2026b; Yu et al., 2026a; Li et al., 2026c). Programmatic and physics-based pipelines further incorporate execution feedback, validation and scene constraints to improve functional correctness (Zhou et al., 2026; Le et al., 2025; Shui et al., 2026; Cao et al., 2026a;b; Yang et al., 2024b; Xia et al., 2026; Pfaff et al., 2026; Ranawaka et al., 2026), while others reuse construction motifs or skills across subsequent generations (Xu et al., 2026a; Kang et al., 2026; Wang et al., 2026). However, preserving articulation requires more than validating individual poses because an asset must maintain feasible connectivity throughout its intended motion. Drawing on configuration space connectivity (LaValle, 2006; Kavraki et al., 1996) and persistent homology (Edelsbrunner et al., 2002; Zomorodian & Carlsson, 2004; Cohen-Steiner et al., 2005; Pokorny et al., 2014), our Homological Motion Signature summarizes required pose connectivity across clearance thresholds and identifies bottleneck configurations to guide skill revision across articulated objects.

3 AdaCraft Skill Evolution

AdaCraft couples Motion Memory, which retains the conditions and outcomes of earlier repairs, with an Adaptive Edit Budget, which controls revision scope from functional feedback. Figure 2 summarizes adaptation at two timescales, with asset repair occurring within an episode and skill evolution occurring across episodes.

Problem Setup. A task $x = (u_x, \mathcal{R}_x)$ specifies an object description u_x and fixed structural and motion requirements $\mathcal{R}_x$. An asset a contains a construction program, part geometry, a joint graph and physical parameters. Let $\mathcal{A}_\theta(x; \mathcal{S}, \xi)$ denote the generation agent's output given a deployed skill library $\mathcal{S}$ and execution randomness ξ. Model parameters θ, tools and the generation and repair budgets remain fixed. Library entries contain procedures, reference examples and applicability conditions. Repairs change the current asset within an episode, while skill revision changes guidance for future generations. Binary indicators $V_{\mathrm{str}}(a, x)$ and $V_{\mathrm{mot}}(a, x)$ record satisfaction of structural requirements and prescribed motion checks, respectively. We maximize functional yield over a task distribution $\mathcal{D}$ by requiring both indicators to hold

$$\begin{aligned} \mathcal{S}^\star \in \arg\max_{\mathcal{S}} \; & \mathbb{E}_{x\sim\mathcal{D},\xi}\left[V_{\mathrm{str}}(a, x)\, V_{\mathrm{mot}}(a, x)\right], \\ & a = \mathcal{A}_\theta(x; \mathcal{S}, \xi). \end{aligned} \tag{1}$$

Skill optimization preserves the task requirements and evaluation protocol. Motion validity is defined on the prescribed probes. Visual quality and resource cost are reported separately.

Motion Memory. The skill library contains modules for geometry, joint structure and frames, motion clearance, and physical attributes and export. Stable rule identifiers make revisions auditable. Our implementation edits instructions and referenced examples while retaining the fixed SDK. For rule j, we maintain a persistent record

$$s_j = (\chi_j, \pi_j, \mathcal{D}_j, \mathcal{W}_j, \mathcal{T}_j, \mathcal{P}_j, \mathcal{N}_j, \mathcal{H}_j). \tag{2}$$

Here χ_j specifies applicability and π_j defines the procedure. The remaining fields record affected geometric relations and checks ($\mathcal{D}_j$), motion check records ($\mathcal{W}_j$), Homological Motion Signatures ($\mathcal{T}_j$), supporting cases and counterexamples ($\mathcal{P}_j, \mathcal{N}_j$), and revision attempts and outcomes ($\mathcal{H}_j$). Evidence supports a rule only under its recorded conditions.

The collection of these rule records forms Motion Memory $\mathcal{M}_t$, which persists across optimization iterations. Retrieve relevant records by joint type, graph motif and compatible frame relations before proposing an edit. Earlier successful cases identify constraints to retain, while failed attempts and counterexamples discourage repeating ineffective corrections. Update the evidence and attempt history after both accepted and rejected proposals. The active procedure and applicability condition change only after acceptance.

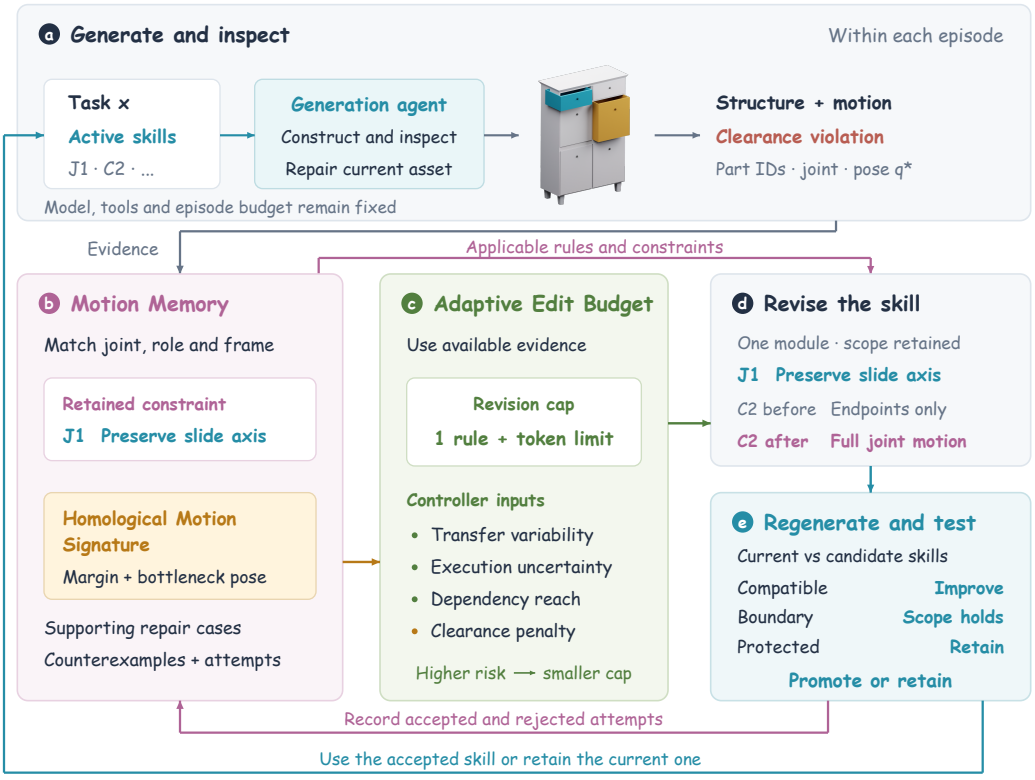

Figure 2: **Overview of AdaCraft.** Within an episode, a fixed agent constructs, inspects, and repairs an asset. Across episodes, Motion Memory retrieves applicable constraints and retains motion evidence. Its records inform the Adaptive Edit Budget, which limits changed rules and tokens. The example preserves joint rule J1 while extending clearance rule C2 to the full joint motion.

For each generated asset, the official checks provide evaluation signals and diagnostic records used to update Motion Memory. They return a metric vector $E(a, x)$ and diagnostic records identifying the tested configuration and relevant parts

$$w = (\text{check}, \text{part and joint IDs}, q, \text{residual}, \text{frame}, \text{program version}). \tag{3}$$

Checks cover required parts, graph structure, valid frames and inertias, motion targets, forbidden penetration and declared simulation tests. Motion probes include endpoints, interior configurations and selected trajectories involving several joints. Visual assessment separately detects malformed geometry that local checks may miss. Generation agents can add diagnostic tests but cannot change official requirements or contact exemptions.

Homological Motion Signature. Two assets can have the same number of collision free samples but differ in whether their start and target poses are connected. We therefore organize motion evidence by connectivity across clearance levels, using established configuration space and persistence constructions (LaValle, 2006; Edelsbrunner et al., 2002). For each task, fix a graph $\Gamma_x = (V_x, E_x)$ of configurations and permitted local motions independently of the candidate. We use reversible motions of one joint or selected pairs of coupled joints. A prescribed trajectory becomes a chain, so an alternative route cannot replace required motion. Figure 3 illustrates the construction on a schematic mechanism.

Let $d_a(q)$ denote the minimum signed separation above the official clearance threshold over forbidden part pairs at configuration q. We normalize this value by a reference length fixed by the task. Positive values indicate spare clearance. An edge receives the minimum measured margin over its endpoints and interior probes, $c_a(e) = \min_{q \in S_x(e)} d_a(q)$. Retaining vertices and edges whose margins are at least a level η produces nested graphs as η decreases. Tracking component mergers yields zeroth persistent homology, H_0. For required poses s and g, define the connection margin

$$\mu_a(s, g) = \max_{p \in \mathrm{Paths}_{\Gamma_x}(s,g)} \min_{e \in p} c_a(e). \tag{4}$$

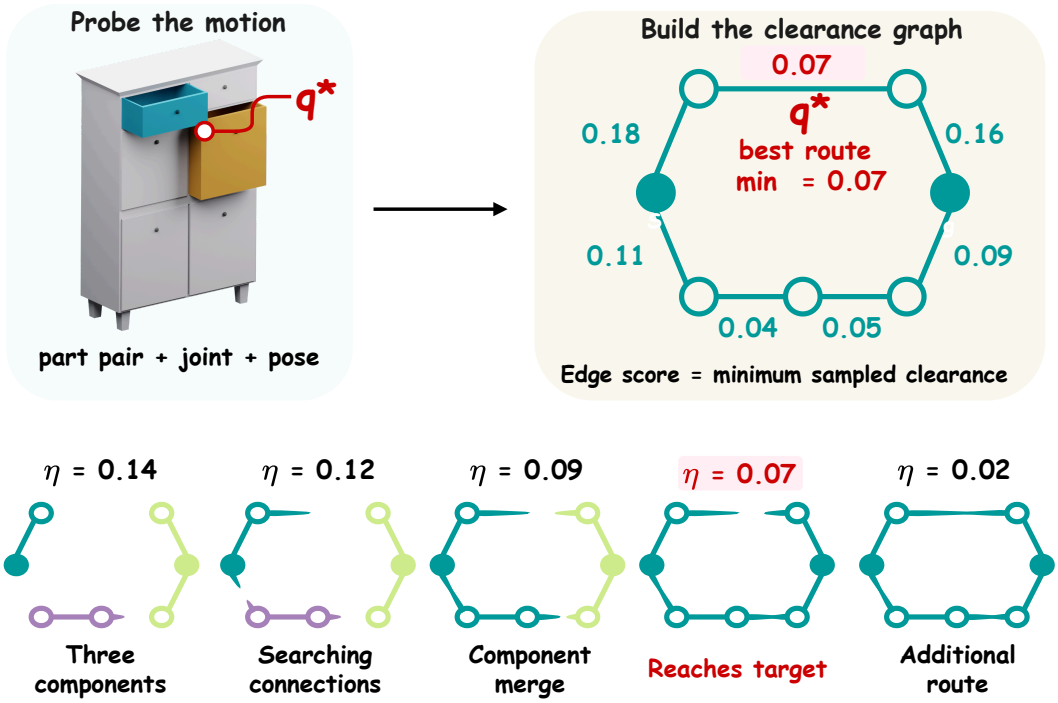

Figure 3: **Homological Motion Signature.** Five clearance levels show tapered connections growing together. Colors distinguish components and unify after mergers. Hollow circles denote intermediate poses; filled dots mark Start and Target. The required poses first connect at 0.07, and the additional route at 0.02 forms a loop. Motion Memory stores this clearance margin, the limiting configuration and the conditions under which the result applies.

This is the largest clearance level at which the required poses remain connected on the measured graph. For the required pose pairs $\mathcal{U}_x$, the *Homological Motion Signature* is

$$\Sigma_a = \left\{(s, g, \mu_a(s, g), \mathcal{B}_a(s, g))\right\}_{(s,g)\in\mathcal{U}_x}, \tag{5}$$

where $\mathcal{B}_a(s, g)$ records the configurations with the smallest clearance along a maximizing path, together with the relevant part pairs and joint roles. Motion Memory stores these signatures with the applicable geometric conditions and repair outcomes. Such evidence helps distinguish an isolated bad pose from a narrow passage that limits an entire motion. On a shared finite graph, two edge score fields satisfying $\|c - c'\|_\infty \le \varepsilon$ obey

$$|\mu_c(s, g) - \mu_{c'}(s, g)| \le \varepsilon. \tag{6}$$

Consequently, a measured connection with margin greater than ε remains positive under such a perturbation. The bound does not apply to arbitrary language edits or certify motion between probes. Missing parts and unmapped joints remain failures under the official evaluator.

Skill applicability and validation. The optimizer proposes an instruction and its applicability condition together. Conditions use a restricted vocabulary of joint type, part role, graph motif, frame relation and required motion. Each predicate must map to task metadata or an available inspection tool. Unknown conditions trigger inspection or abstention. A broad category label alone is insufficient evidence that a mechanical correction applies.

We evaluate each candidate on three groups of tasks. Compatible tasks $\mathcal{C}_j$ test whether the rule transfers to new objects that satisfy its conditions. Boundary tasks $\mathcal{B}_j$ change one relevant condition or revisit a known counterexample to check where the rule should not apply. Protected tasks $\mathcal{R}_t$ test whether the revision preserves previously acquired capabilities. These groups draw from fixed task pools and variation templates within the optimization data. Separate development tasks select checkpoints, and test tasks never inform skill revision. We track which rules the agent retrieves and applies. On a subset of tasks, we compare runs with and without the candidate rule to assess its effect. A rule should remain inactive outside its stated conditions and should not reduce functional

performance on those tasks. When a revision narrows those conditions, we also test the cases it excludes to check whether useful coverage has been lost.

Adaptive Edit Budget. The Adaptive Edit Budget adjusts revision size according to how consistently an edit improves objects that use the same construction rules. The outcome vector includes geometric and motion checks, so a clearance gain cannot hide a loss of required parts before variability is measured. Start with one active skill module per candidate. Match the current and candidate skills on task, generation schedule, repair limit and probe policy. For normalized metric m, task i and repeat r, define

$$\delta_{i,r,m} = E_m(a_{i,r}^{\mathrm{cand}}, x_i) - E_m(a_{i,r}^{\mathrm{cur}}, x_i). \tag{7}$$

All metrics are oriented so that larger values are better. Let $\widehat{v}_{t,k}$ summarize variation of task mean improvements within the active module k, $\widehat{u}_{t,k}$ summarize uncertainty from repeated executions, and $d_{t,k} \in [0,1]$ be the fraction of rules and checks reached by the declared dependency closure of the edit. The Homological Motion Signature also provides a clearance penalty $f_{t,k} \in [0,1]$, which increases as the clearance along required motion routes falls below a fixed threshold. This limits edits when small geometric changes could block motion, even if earlier improvements were consistent across tasks. The controller sets the next allowed rule count as

$$b_{t+1,k} = \mathrm{clip}_{[b_{\min}, b_{\max}]} \left\lfloor \frac{b_{\max}}{1 + \lambda_v \widehat{v}_{t,k} + \lambda_u \widehat{u}_{t,k} + \lambda_d d_{t,k} + \lambda_p f_{t,k}} \right\rfloor. \tag{8}$$

At step t, we set the edit budget using only evidence available before proposing the candidate revision. During initialization and for modules with limited evaluation history, we use a conservative fixed rule cap. We compute the clearance penalty from assets generated with the current skill, evaluating all required motions under the fixed coverage policy rather than selecting only those with high clearance. A separate cap on changed tokens prevents the optimizer from bypassing the rule cap by rewriting an entire module under a single rule identifier.

Skill acceptance and retention. A candidate can enter the persistent library only after compatible, boundary and protected tasks have been executed with the updated skill. Rechecking an unchanged archived mesh can reproduce a diagnostic record but cannot demonstrate retention by the generator. The proposed gate requires a supported improvement on the designated compatible objective, preservation of required motion and structure, no unacceptable regression in protected mechanism groups, and acceptable rule use on the boundary cohort. Thresholds, cohort sizes and uncertainty rules are specified in the evaluation protocol and shared by the corresponding controls.

We compute uncertainty intervals for paired comparisons of the current and candidate skills, treating each source task—not each pose—as the sampling unit. If the comparison is inconclusive, we retain the current skill or run additional repeats within the resource budget. Because testing is repeated and adaptive, these intervals guide acceptance decisions but do not guarantee that every accepted revision improves performance. We limit the number of protected tasks to control evaluation cost, select them to cover different mechanism groups, and record which capabilities they test. Candidates cannot gain acceptance by reducing motion below task requirements, deleting required parts, disabling collisions, or relaxing official evaluation thresholds.

4 Experiments

Experimental setup. We evaluate transfer to new object instances, categories, and mechanism compositions on 240 test tasks, using three independent runs with two generation attempts per task in each run. All methods use the same frozen generation backbone (Zhou et al., 2026), GPT-5.5 with medium reasoning effort, official motion probes, and resource limits. Skill optimization stops after 24 revision rounds or 300M model tokens, whichever comes first. Baselines include fixed skills, case retrieval, skill accumulation, and iterative skill optimization (Chen et al., 2024a; Ni et al., 2026; Yang et al., 2026; Li et al., 2026a). We select checkpoints based on development performance and never use test feedback to update the skill library. Our primary metric is functional yield: the fraction of all generation attempts that satisfy both structural requirements and prescribed motion checks. We also report visual quality assessed by reviewers blinded to the method, deployment model calls per functional asset, and total optimization cost. Appendix A provides additional implementation and evaluation details.

Table 1: **Functional generalization and deployment cost.** Functional yield (%) is reported on new object instances, categories, and mechanism compositions. Overall is the equally weighted mean across these splits. Yield values are means ± sample standard deviations across three runs. Visual reports blinded quality ratings on a 1–5 scale. Calls reports deployment model calls per functional asset, including calls spent on failed attempts. SkillAdam + shared gate uses the same skill acceptance criteria and evaluation cohorts as AdaCraft. **Best** and second best values are highlighted.

	Functional yield ↑				Visual	Calls
Method	Instances	Categories	Compositions	Overall	↑	↓
Fixed agent	47.3 ± 1.0	36.0 ± 1.0	25.2 ± 1.0	36.2 ± 0.9	4.05	48.1
Fixed skills	51.2 ± 1.2	40.6 ± 1.2	30.2 ± 1.0	40.7 ± 1.1	4.13	42.5
Case retrieval	55.4 ± 1.0	46.2 ± 1.2	34.6 ± 1.0	45.4 ± 1.0	4.17	39.2
Naive accumulation	49.6 ± 1.3	40.0 ± 1.2	28.8 ± 1.2	39.4 ± 1.3	4.08	48.7
SkillOpt	64.6 ± 1.3	52.7 ± 1.3	42.3 ± 1.0	53.2 ± 1.2	4.20	32.7
SkillAdam	67.1 ± 1.3	55.6 ± 1.2	44.8 ± 1.0	55.8 ± 1.2	4.22	30.3
SkillAdam + shared gate	69.2 ± 1.0	57.5 ± 1.2	47.5 ± 1.2	58.1 ± 1.1	4.21	28.9
AdaCraft	$\mathbf{82.3 \pm 1.0}$	$\mathbf{75.0 \pm 1.2}$	$\mathbf{67.9 \pm 1.0}$	$\mathbf{75.1 \pm 1.0}$	**4.29**	**16.8**

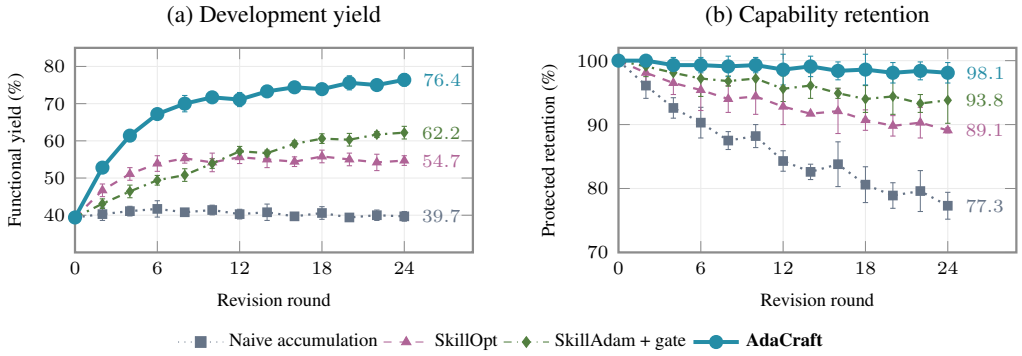

Figure 4: **Progress and retention through skill revision.** (a) Functional yield on development tasks, starting from a shared initial skill library. (b) Capability retention, measured as the functional yield of fresh generations on a fixed set of audit tasks for which both initial generation attempts were successful. Curves show means across three runs, error bars indicate sample standard deviations, and endpoint labels report final means. AdaCraft achieves the highest final development yield and capability retention among the compared methods.

Generalization across objects and mechanisms. Table 1 reports functional yield for each generalization split and the equally weighted average across splits. AdaCraft outperforms SkillAdam + shared gate, the strongest baseline with the same feedback access and skill acceptance criteria, on all three splits. Overall yield increases from 58.1% to 75.1%, with the largest gain on new mechanism compositions, where yield improves from 47.5% to 67.9% (+20.4 percentage points). These gains are accompanied by a higher visual quality score (4.29 vs. 4.21) and fewer deployment model calls per functional asset (16.8 vs. 28.9).

Optimization stability and resource use. Figure 4 shows that AdaCraft increases development yield from 39.4% to 76.4%, exceeding SkillAdam + shared gate by 14.2 percentage points at the final round. By round six, it already surpasses this baseline's final yield of 62.2%. This improvement accompanies 98.1% capability retention, compared with 93.8% for the strongest control. In contrast, naive accumulation ends near its initial yield at 39.7%, while its retention falls to 77.3%. Figure 5(a) shows that these gains also require fewer optimization tokens. AdaCraft uses 85.4M tokens versus 214.7M for the strongest control, a 60.2% reduction, and achieves the highest final yield at the lowest cost among the compared optimizers. Panel (b) illustrates how a larger clearance penalty reduces the maximum number of edited rules when required motions have little spare clearance. Full trajectories and cost accounting appear in Appendices B.5 and A.

Which components support reliable revision? Table 2 isolates each component under identical resource budgets. Removing Motion Memory retains current probes and controller statistics but reduces functional yield from 75.1% to 63.4% and increases failure recurrence from 7.9% to 21.3%.

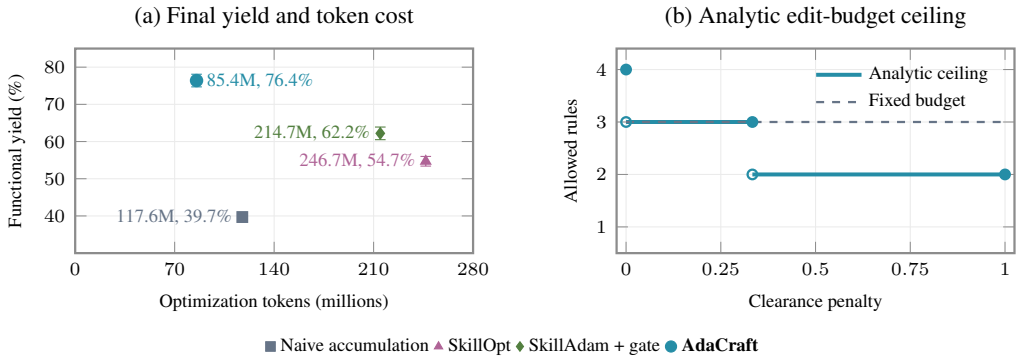

Figure 5: **Optimization efficiency and adaptive edit scope.** (a) Final development yield versus optimization tokens, averaged across three runs; vertical bars show sample standard deviations. Cached input is included; monitoring costs are reported separately. (b) After initialization, the analytic edit ceiling is $\lfloor 4/(1+f) \rfloor$ for clearance penalty f, with other penalties set to zero. The dashed line shows the fixed budget of three rules. Filled and open endpoints denote included and excluded values, respectively.

Table 2: **Component ablations.** All variants share the initial library and generation budget. Function is overall functional yield, while Structure and Motion are separate pass rates. Retention measures functional yield on initially successful audit tasks. False triggers and Recur. measure out-of-scope rule activation and recurrence of resolved failures. Values are percentages averaged across three runs. **Best** and second best highlighted.

Variant	Function ↑	Structure ↑	Motion ↑	Retention ↑	False triggers ↓	Recur. ↓
Neither memory nor adaptation	54.8 ± 2.1	81.6	66.9	80.6	15.7	27.1
Without Motion Memory	63.4 ± 1.9	85.2	73.1	86.8	12.4	21.3
Fixed edit budget	65.8 ± 1.8	86.4	76.1	90.3	9.3	18.2
Without motion signature	68.2 ± 1.7	87.3	77.5	93.5	7.8	14.4
Without clearance penalty	71.0 ± 1.5	88.1	80.0	94.7	6.4	12.8
Without scope challenges	69.1 ± 1.9	86.9	79.2	93.1	13.6	16.1
Without protected replay	71.8 ± 1.6	88.4	80.5	88.7	6.9	17.2
AdaCraft	$\mathbf{75.1 \pm 1.0}$	**88.8**	**83.2**	**98.1**	**4.7**	**7.9**

Table 3: **Motion summaries under identical probes.** Chain and Routes report functional yield (%) for prescribed motion chains and tasks permitting alternative routes. Dense failures is the percentage of initially passing assets that fail denser motion checks. Time reports summarization cost (ms), excluding shared distance queries. **Best** and second best values are highlighted.

Motion summary	Chain ↑	Routes ↑	Dense failures ↓	Time ↓
Raw check records	68.4 ± 1.9	52.9 ± 2.3	10.9	**0.6**
Violation count	69.6 ± 1.7	54.1 ± 2.0	10.2	0.7
Minimum clearance	$\mathbf{76.9 \pm 1.4}$	58.6 ± 2.1	8.8	0.7
Clearance quantiles	75.8 ± 1.6	61.4 ± 1.8	8.4	1.2
Homological Motion Signature	$\mathbf{76.9 \pm 1.4}$	$\mathbf{72.7 \pm 1.7}$	**5.7**	4.8

Fixing the edit budget at three rules lowers yield to 65.8% and retention to 90.3%. Removing the Homological Motion Signature retains minimum clearance and its diagnostic record, yet reduces yield to 68.2%; removing only its clearance penalty yields 71.0%. Without scope challenges, false triggers rise from 4.7% to 13.6%, while removing protected replay lowers retention from 98.1% to 88.7%. These results support complementary roles for repair memory, motion connectivity, adaptive editing and targeted validation.

Does motion connectivity add information? Table 3 shows that the homological motion signature improves functional yield when motion permits alternative routes. Compared with minimum clearance, it raises yield from 58.6% to 72.7% and reduces failures under denser motion checks from 8.8% to 5.7%. On prescribed chains, both summaries achieve 76.9% because the connection margin reduces to minimum path clearance. These results indicate that identifying which configurations



Figure 6: **Qualitative comparison of articulated objects.** Each plain/material pair shares four camera views and one pose.

remain connected provides useful repair guidance beyond individual clearance measurements. Computing the signature takes 4.8 milliseconds, excluding the distance queries shared by all summaries.

Qualitative observations. Figure 6 illustrates reported observations through examples of a drawer cabinet, refrigerator, microwave and folding arm. GT Rendering shows released assets for reference. Each row also describes the object and its intended motion, and each comparison panel contains four camera views: plain and material grids share geometry, pose and cameras. The reconstructions depict cleaner geometry with AdaCraft and articulation defects in comparison conditions. The cabinet's blue drawer slides correctly throughout, while its yellow drawer and the refrigerator's pink drawer rotate instead of translating in the faulty condition. Other defects include misplaced pivots and overlapping links.

5 Conclusion

We present AdaCraft for stable and adaptive skill evolution in articulated 3D asset generation. Motion Memory retains reusable repair evidence and motion signatures, while an Adaptive Edit Budget regulates revision scope. Our method improves functional generalization and preserves established capabilities during skill optimization. These gains are accompanied by fewer deployment calls per functional asset, suggesting that reliable skill reuse can also reduce generation cost.

AI USE STATEMENT

We used LLMs to improve writing clarity and grammar. LLMs and vision language models also support articulated asset generation, failure diagnosis and skill revision in AdaCraft, as described in the paper. The authors reviewed text edited with AI assistance and take full responsibility for the paper's content and conclusions.

REPRODUCIBILITY STATEMENT

Section 4 describes the simulation setup, model configurations, and evaluation protocols. Appendix A specifies implementation details, resource budgets and evaluation settings. We will publicly release our implementation, including prompt templates, skill optimization and evaluation scripts, and the learned skill libraries upon publication.

ETHICS STATEMENT

AdaCraft is intended to support research on articulated asset generation for simulation and design. Generated geometry and joint specifications require independent validation before physical deployment, since passing simulated motion checks does not establish real world safety. Reference assets and generated models should be used in accordance with their licenses and without misleading claims about the functionality or safety of real products.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Ziang Cao, Fangzhou Hong, Zhaoxi Chen, Liang Pan, and Ziwei Liu. Physx-anything: Simulation-ready physical 3d assets from single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5839–5848, 2026a.

Ziang Cao, Yinghao Liu, Haitian Li, Runmao Yao, Fangzhou Hong, Zhaoxi Chen, Liang Pan, and Ziwei Liu. Physx-omni: Unified simulation-ready physical 3d generation for rigid, deformable, and articulated objects. *arXiv preprint arXiv:2605.21572*, 2026b.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631, 2024a.

Zoey Chen, Aaron Walsman, Marius Memmel, Kaichun Mo, Alex Fang, Karthikeya Vemuri, Alan Wu, Dieter Fox, and Abhishek Gupta. Urdformer: A pipeline for constructing articulated simulation environments from real-world images. *arXiv preprint arXiv:2405.11656*, 2024b.

David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pp. 263–271, 2005.

Edelsbrunner, Letscher, and Zomorodian. Topological persistence and simplification. *Discrete & computational geometry*, 28(4):511–533, 2002.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. Skillaxe: Sharpening llm-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Haoqiang Kang, Xiaokang Ye, Yuhan Liu, Siddhant Hitesh Mantri, Lingjun Mao, James Fleming, Drishti Regmi, and Lianhui Qin. Simworld studio: Automatic environment generation with evolving coding agent for embodied agent learning. *arXiv preprint arXiv:2605.09423*, 2026.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Steven Michael LaValle. *Planning algorithms*, volume 1. Cambridge university press Cambridge, UK, 2006.

Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model. In *International Conference on Learning Representations*, volume 2025, pp. 17578–17602, 2025.

Jiahui Lei, Congyue Deng, William B Shen, Leonidas Guibas, and Kostas Daniilidis. Nap: Neural 3d articulated object prior. *Advances in Neural Information Processing Systems*, 36:31878–31894, 2023.

Gaoyuan Li, Meihao Fan, Yizhe Liu, Shaolei Zhang, Ju Fan, Siyi Wang, Jiaheng Hou, Xudong Weng, Honghan Tian, and Zang Li. Skilladam: Stable and efficient skill evolution for agents. *arXiv preprint arXiv:2609.08944*, 2026a.

Ruining Li, Yuxin Yao, Chuanxia Zheng, Christian Rupprecht, Joan Lasenby, Shangzhe Wu, and Andrea Vedaldi. Particulate: Feed-forward 3d object articulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 27708–27718, 2026b.

Ruining Li, Yuxin Yao, Matt Zhou, Chuanxia Zheng, Christian Rupprecht, Joan Lasenby, Shangzhe Wu, and Andrea Vedaldi. Instruct-particulate: Scaling feed-forward 3d object articulation with kinematic control. *arXiv preprint arXiv:2606.14699*, 2026c.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 300–309. IEEE, 2023.

Jiayi Liu, Ali Mahdavi-Amiri, and Manolis Savva. Paris: Part-level reconstruction and motion analysis for articulated objects. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 352–363. IEEE, 2023a.

Jiayi Liu, Hou In Ivan Tam, Ali Mahdavi-Amiri, and Manolis Savva. Cage: Controllable articulation generation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17880–17889. IEEE, 2024.

Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9264–9275. IEEE, 2023b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes. In *Forty-third International Conference on Machine Learning*, 2026.

Florian T Pokorny, Majd Hawasly, and Subramanian Ramamoorthy. Multiscale topological trajectory classification with persistent homology. In *The 2014 Robotics: Science and Systems Conference, July 12-16, 2014 Berkeley, USA*. MIT Press Journals, 2014.

Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7957–7968, 2023.

Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, et al. Simfoundry: Modular and automated scene generation for policy learning and evaluation. *arXiv preprint arXiv:2606.28276*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. Articad: Articulated cad assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026.

Jiayi Su, Youhe Feng, Zheng Li, Jinhua Song, Yangfan He, Botao Ren, and Botian Xu. Artformer: Controllable generation of diverse 3d articulated objects. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1894–1904. IEEE, 2025.

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.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024a.

Xinjie Wang, Liu Liu, Taojun Ding, Andrew Choi, Chaodong Huang, Mengao Zhao, Ziang Li, Jackson Jiang, Chunlei Yu, Shengxiang Liu, et al. Embodiedgen v2: An agentic, simulation-ready 3d world engine for embodied ai. *arXiv preprint arXiv:2607.07459*, 2026.

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

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. *arXiv preprint arXiv:2602.10116*, 2026.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11094–11104. IEEE, 2020.

Ruijie Xu, Xinnan Zhu, Jiayu Ying, Daoguo Dong, Yuzhou Ji, and Xin Tan. Muse: Agentic 3d scene authoring via memory-grounded incremental requirement satisfaction. *arXiv preprint arXiv:2606.14168*, 2026a.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026b.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024a.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16277–16287. IEEE, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Tianjiao Yu, Xinzhuo Li, Muntasir Wahed, Jerry Xiong, Yifan Shen, Ying Shen, and Ismini Lourentzou. Dreampartgen: Semantically grounded part-level 3d generation via collaborative latent denoising. *arXiv preprint arXiv:2603.19216*, 2026a.

Tianjiao Yu, Vedant Shah, Muntasir Wahed, Ying Shen, Kiet A Nguyen, and Ismini Lourentzou. Part^{2}GS: Part-aware modeling of articulated objects using 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18913–18923, 2026b.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, number 17, pp. 19632–19642, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, number 17, pp. 19724–19731, 2024.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation. *arXiv preprint arXiv:2605.15187*, 2026.

Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 347–356, 2004.

A Implementation and Evaluation Details

This appendix specifies the task construction, model configuration, resource accounting and evaluation protocol used throughout the experiments.

A.1 Task construction and evaluation schedule

Table 4: **Task allocation and feedback access.** Mechanism columns count tasks with primarily revolute, primarily prismatic, or multi-joint motion. Each task belongs to one row and one mechanism group.

Pool	Total	Rev.	Pris.	Mixed	Permitted use
Diagnosis	120	40	40	40	Proposals, retrieval and protected replay
Promotion	36	12	12	12	Compatible and boundary checks
Retention audit	24	8	8	8	Monitoring only; traces hidden
Development	60	20	20	20	Checkpoint selection; traces hidden
New instances	80	30	30	20	Final test within familiar categories
New categories	80	30	30	20	Final test of unseen categories
New compositions	80	30	30	20	Final test of unseen joint arrangements
Total	480	170	170	140	240 test tasks, 1,440 test attempts per method

Table 4 summarizes task allocation and permitted feedback access for diagnosis, skill acceptance, and monitoring. Tasks derived from the same source asset or construction template, including all geometric variants, remain in the same partition. For the new-category and new-composition splits, we exclude the held-out categories and joint arrangements, respectively, from both optimization and development. Each task specifies its required parts, joint roles, motion targets, contact exemptions, and reference length L before generation. We define L as the diagonal of the task's fixed reference extent, not that of the generated asset. Tasks with closed kinematic loops are excluded. Before any optimization run, we freeze the task manifest and record its hash.

We perform three complete runs with task and probe schedules seeded by 42, 43, and 44. These seeds control local sampling but do not make API generations deterministic. Each run includes 160 generation attempts per test split and 120 development attempts per checkpoint. Before the first revision, we evaluate the shared initial library once on the development and retention-audit tasks. We reuse these initial generations across methods, giving all methods the same round-zero development scores and eligible audit tasks within each run. The shared initial checkpoint contains no method-specific edits or accumulated evidence. At subsequent checkpoints, each method is evaluated independently. For each method and run, we select the checkpoint with the highest development functional yield, breaking ties first by fewer generation calls and then by the earlier revision round. Checkpoint selection uses only aggregate development results; the optimizer receives neither development traces nor test feedback.

At each checkpoint, we generate two fresh assets for each of the 24 retention-audit tasks. We define established capabilities once per run using the subset of tasks for which both initial generations are functional. We report this subset's size and mechanism coverage and keep its membership fixed, without replacing difficult tasks to enlarge it. At each later checkpoint, retention is the fraction of functional generations on this fixed subset; it is undefined if no task qualifies. Audit scores and execution traces are used only for monitoring and never inform candidate proposals, skill acceptance, or checkpoint selection.

A.2 Model, environment, and resource accounting

Both generation and skill revision use the official GPT-5.5 model snapshot listed in Table 5, with medium reasoning effort.[1] We disable model-based context compaction. Input token counts include

[1] https://developers.openai.com/api/docs/models/gpt-5.5, accessed 21 September 2026.

Table 5: **Model and evaluation settings.** Resource limits apply to every method. Controller parameters are fixed before evaluation. Tokens include repeated and cached input.

Setting	Value
Generation and revision model	`gpt-5.5-2026-04-23`
API and reasoning effort	Responses API; `medium`
Sampling overrides	Temperature and `top_p` omitted
Input and output per request	131,072 input tokens; 16,384 output tokens
Generation episode	At most 32 model calls and 1M total tokens
Revision run	At most 24 revision rounds and 300M total tokens
Optimizer calls per round	At most 8, excluding asset generation
Independent runs	3; schedule seeds 42, 43 and 44
Development and test repeats	2 fresh generations per task per run
Monitoring schedule	Rounds 0, 2, . . ., 24 and the final completed round
Skill library	4 modules; 2,048 tokens per module
Candidate edit	1 module; at most 1,024 changed tokens
Retrieved evidence	At most 8 records and 4,096 tokens
Rule budget	$b_{\min} = 1$, $b_{\max} = 4$; initial cap 2
Fixed budget ablation	3 rules, with the same token cap
Controller coefficients	$(\lambda_v, \lambda_u, \lambda_d, \lambda_p) = (1, 1, 0.5, 1)$
Variance smoothing	$\beta = 0.9$; correction for initial bias
Clearance threshold	$\tau = 0.005$ in task normalized clearance
Candidate cohorts	8 compatible, 4 boundary, 8 protected tasks
Paired repeats	2 per skill and task; 4 if inconclusive
Length tolerance	$\epsilon = \max(10^{-4}\,\mathrm{m}, 10^{-4}L)$
Single motion graph	17 vertices; 3 interior probes per edge
Permitted pair graph	9×9 vertices; axis adjacent edges
Dense motion audit	12 interior probes per edge
Uncertainty	2,000 clustered bootstrap replicates; 95% intervals

instructions, skills, retrieved evidence, and tool history; output limits include reasoning tokens when reported by the provider. We do not issue requests that would exceed the context or resource limits.

Each model request counts as one call, regardless of the number of associated tool invocations. Before issuing a request, we reserve its output-token allowance, reducing it as needed to stay within the remaining run budget. An episode ends when any of its limits is reached. We then evaluate the last complete exported asset, or count the attempt as a failure if none exists. Optimization likewise stops when a run limit is reached; candidates with incomplete evaluations cannot be accepted. Optimization costs include rejected revisions, unsuccessful programs, evaluations of the current skill, replay, scope audits, and memory maintenance. The optimization token axis excludes development evaluations and the fixed retention audit. We report their costs separately, along with initialization, final testing, and geometric execution. Token accounting uses API usage records and counts cached input on every request.

All runs start with the same frozen SDK guidance. Methods that use skills also share the same four initial task-skill modules. No initial library is selected using final evaluation results. Episode caps and library limits are imposed by our evaluation adapter and should not be interpreted as defaults of the upstream implementation. We record request identifiers, API usage, asset versions, the package lockfile, and the actual CPU configuration. Geometry timings use identical CPU workers, each allocated four cores and 16 GiB of memory.

A.3 Revision schedule, memory, and skill acceptance

Each revision round diagnoses 20 tasks from the 120-task diagnosis pool, with two generation attempts per task. All adaptive methods receive the same prescheduled batches, formed by four shuffled passes through this pool. Reflection prioritizes failed cases in groups of at most eight. We revise the module with the most failed checks, breaking ties using a fixed module order. Before observing candidate outcomes, we use task metadata to select eight compatible and four boundary tasks from the acceptance pool. For protected replay, we select eight established tasks from a reservoir of at most 48 diagnosis tasks, balanced across the available mechanism groups. If fewer than

eight established tasks are available, we replay all of them and fill the remaining slots using a fixed, mechanism-balanced diagnosis cohort. If too few compatible or boundary tasks are available for the proposed scope, we retain the current library and record insufficient coverage.

We evaluate the current and candidate libraries on the same tasks and probe schedules, with two independent generations per task for each library. A standard acceptance evaluation therefore requires 80 generation attempts, in addition to the 40 diagnosis attempts. A comparison is inconclusive when its point estimates satisfy the acceptance criteria but an uncertainty interval crosses an acceptance threshold. We repeat an inconclusive evaluation once and combine the results, giving four generations per task for each library. If the combined comparison remains inconclusive or fails the criteria, we retain the current library. Candidates whose initial point estimates fail the criteria are rejected without additional repeats. All calls count toward the shared token limit, and cached assets never replace required fresh generations.

We organize the checks into five groups: parts; joint structure and frames; required motion targets; forbidden contacts; and physical attributes and export. Each group is scored as the fraction of its required checks passed. The equally weighted mean of these five scores defines the compatible-task objective Q; missing programs score zero. Acceptance requires an increase of at least 0.02 in mean Q on compatible tasks, with a lower bound greater than zero for the paired 95% bootstrap interval of this increase. Compatible functional yield must not decrease. For structure and motion separately, the candidate must not fail a task that passed the corresponding checks in every repeat under the current library. Within each represented mechanism group of the protected cohort, the lower interval bound for the change in functional yield must exceed -0.05. On boundary tasks, the candidate must neither activate rules outside their declared scope nor cause a previously functional task to fail. We compute acceptance intervals using 2,000 paired bootstrap replicates clustered by source task within the current run. Because testing is repeated and adaptive, these intervals guide acceptance decisions but do not guarantee against erroneous acceptance.

We retrieve memory records by compatible joint type, graph motif, part roles, and frame conditions, preferring more recent records when matches are tied. Active memory stores at most eight supporting cases, eight counterexamples, and eight recent revision attempts per rule; complete traces are archived separately. Each retrieval returns at most eight records within a 4,096-token budget. When an applicability condition cannot be determined, the agent must inspect the asset or abstain from applying the rule. To audit whether rule use matches its declared scope, we perform two additional generations with the candidate rule disabled on one prescheduled boundary task per round. These generations count toward optimization cost.

A.4 Controller normalization and initialization

We use the five check-group pass fractions and a binary indicator of functional success as the metric vector in Eq. 7. For a completed comparison involving n compatible tasks and R repeats per task, let $\bar{\delta}_{i,m}$ denote the mean paired difference over repeats for task i and metric m, and let $\bar{\delta}_m$ denote its mean across tasks. For the active module k, we compute

$$v_k^{\mathrm{raw}} = \max_m \frac{1}{n} \sum_{i=1}^{n} (\bar{\delta}_{i,m} - \bar{\delta}_m)^2, \tag{9}$$

$$u_k^{\mathrm{raw}} = \mathrm{clip}_{[0,1]} \left(\max_m \frac{1}{nR} \sum_{i=1}^{n} \frac{\sum_{r=1}^{R} (\delta_{i,r,m} - \bar{\delta}_{i,m})^2}{R-1} \right). \tag{10}$$

The first statistic measures variation in mean improvements across tasks and already lies in $[0, 1]$. The second uses variation across repeated generations to estimate execution uncertainty. We initialize both module-specific exponential moving averages to zero and update them after every completed comparison, whether the candidate is accepted or rejected, using decay $\beta = 0.9$. After n_k updates, we divide each average by $1 - \beta^{n_k}$ to obtain the bias-corrected statistics $\widehat{v}$ and $\widehat{u}$. We retain the initial two-rule cap until the module has completed at least three comparisons spanning at least eight distinct tasks.

Before generating a patch, we select its target rule identifiers. We compute d as the fraction of active rule and official check identifiers reached by the targets' combined dependency closure. The patch cannot edit rules outside this declared target set. If dependency information is missing, we

set $d = 1$. We compute clearance penalties from assets generated by the current skill on diagnosis tasks, including every required pose pair. For asset a and task x, the penalty is

$$f(a, x) = \frac{1}{|\mathcal{U}_x|} \sum_{(s,g)\in\mathcal{U}_x} \mathrm{clip}_{[0,1]} \left(1 - \frac{\mu_a(s, g)}{0.005}\right). \tag{11}$$

Missing geometry, unmapped joints, or missing required evidence contribute the maximum penalty of one.

We set the next rule cap in Eq. 8 using only evidence available before the proposal. Every added, deleted, or modified rule counts toward this cap, including changes to applicability conditions or examples. We enforce the separate 1,024-token limit by counting inserted plus deleted tokens in a token-sequence diff, preventing a long rule replacement from bypassing the limit. Patches that exceed either limit are rejected before asset evaluation.

A.5 Motion probes and official validity

For each prescribed motion, we construct a chain of 17 uniformly spaced configurations, including both endpoints, and sample three interior configurations per edge, giving 65 distinct configurations. For a pair of joints permitted to move independently, we use a 9×9 grid with 144 edges connecting adjacent vertices along one coordinate at a time. With three interior probes per edge, this gives 513 configurations. Prescribed coupled trajectories remain parameterized chains; an alternative grid route cannot substitute for the required motion. All other joints remain at the states specified in the task manifest. The dense audit uses 12 interior probes per edge, giving 209 configurations per chain and 1,809 per joint-pair grid. We evaluate every required motion without reducing probe density for more complex objects.

We require a connected joint tree, all required part and joint roles, finite transforms, valid joint limits, positive masses, and valid inertia tensors. We normalize joint axes and require rotation matrices to be orthogonal within a tolerance of 10^{-6}. Forbidden part pairs are evaluated using the fixed mesh-distance and collision backend, with the clearance threshold specified in Table 5. For intersecting pairs, we set geometric separation to zero and subtract the positive clearance threshold, producing a negative spare-clearance margin. This margin does not measure penetration depth. Contact exemptions are restricted to those specified in the task manifest. Motion validity requires every prescribed pose and trajectory to pass the official checks and every required pose pair to have a permitted route with nonnegative margin. An unsafe optional edge does not, by itself, invalidate a permitted route graph. Any additional simulation checks must be specified in the task manifest before generation. Sampled connectivity does not establish collision freedom between probes.

A.6 Controls, reporting, and uncertainty

Table 6: **Baseline configurations for articulated 3D generation.** All baselines share the model, evaluator, and resource limits. Skill-based methods also share the initial library.

Control	Study configuration
Fixed agent	SDK instructions and episode repair; no task skill library.
Fixed skills	Shared initial library, frozen throughout the run.
Case retrieval	Frozen library plus up to 8 compatible diagnosis records within 4,096 tokens; no rule revision.
Naive accumulation	Append up to 4 rules per round from diagnosis feedback; no promotion gate or revision of earlier rules. Stop appending at the shared library limit.
SkillOpt	4 passes of 6 batches of 20 diagnosis tasks; reflection groups of at most 8; cosine rule budget from 4 to 2; slow update on 20 tasks after passes 2 to 4; native selection.
SkillAdam	24 rounds with 20 diagnosis tasks per round; adaptive budget from 4 to 1, $\beta = 0.9$, variance scale 0.25; native selection.
SkillAdam + gate	Same proposal and budget configuration, with the common compatible, boundary and protected gate.

We implement the two iterative baselines, SkillOpt and SkillAdam, using the SkillAdam reference implementation at commit `72ef9ba48` (Yang et al., 2026; Li et al., 2026a). Table 6 specifies our

Table 7: **Sensitivity to motion probe density on frozen assets.** Interior probes are counted per edge; Chain and Pair report total configuration counts. Dense rejection is the percentage of passing assets that fail the 12-probe audit; dense recovery is the percentage of failing assets that pass it. At 12 probes, both rates are zero by definition whenever defined. **Best** and second best values are highlighted among the 1-, 3-, and 6-probe settings only.

Interior probes	Chain poses	Pair poses	Dense rejection ↓	Dense recovery ↓
1	33	225	9.1	0.8
3 (default)	65	513	5.7	0.6
6	113	945	**3.2**	**0.4**
12	209	1,809	0.0	0.0

task-specific adaptations. Native selection evaluates the current and candidate libraries on eight compatible tasks from the acceptance pool, with two generations per task for each library. These comparisons use the shared normalized checks, with Q as the scalar selection objective. SkillAdam + shared gate uses the full evaluation cohorts and acceptance criteria in Appendix A.3. All proposal, reflection, and slow-update calls count toward the eight-call optimizer limit per round and the common token budget. A slow update occurs within its scheduled round, and its edits count toward that round's cumulative change limit. Generic baselines receive the same inspection tools and source-task feedback. Only AdaCraft adds the proposed structured Motion Memory and geometric edit controller.

We compute functional yield and the separate structural and motion pass rates over all generation attempts. We define deployment calls per functional asset as the total number of deployment model calls, including those spent on failed attempts, divided by the number of functional outputs. For visual evaluation, we use a fixed subset of 60 test tasks, balanced across generalization splits and mechanism groups. Three independent reviewers, blinded to method, rate geometry, appearance, and agreement with the prompt on a 1–5 scale using matched camera views. Missing renders receive a score of 1. We average reviewer scores before aggregating across tasks and report inter-reviewer agreement.

We report means and sample standard deviations across the three runs. Cost curves use mean cumulative optimization tokens per run. For final method comparisons, we generate 2,000 paired bootstrap samples by resampling runs and source tasks within each test split, keeping each task's repeated generations together. We use identical resampled indices across methods to preserve pairing. We report 95% intervals for between-method differences, with functional yield as the primary comparison.

B Additional Analyses and Examples

The following analyses examine the effects of motion sampling, controller settings, and memory retrieval beyond the main comparisons. Task allocations and configuration counts follow the fixed protocol. Each intervention changes only the specified factor, retains the shared resource limits, and uses the uncertainty procedure in Appendix A.6.

B.1 Sensitivity to motion probe density

We reevaluate the frozen test assets from the motion-signature study using $p \in \{1, 3, 6, 12\}$ interior probes per edge. Graph vertices, permitted edges, motion targets, and clearance thresholds remain fixed, isolating evaluation resolution from generation quality. A prescribed chain contains $17 + 16p$ distinct configurations, whereas a permitted joint-pair grid contains $81 + 144p$. Table 7 reports these exact configuration counts and the corresponding audit outcomes.

Let M_p indicate whether an asset passes the motion checks at probe density p. Relative to the $p = 12$ audit, dense rejection is $\Pr(M_{12} = 0 \mid M_p = 1)$, whereas dense recovery is $\Pr(M_{12} = 1 \mid M_p = 0)$. Each rate is undefined when its conditioning set is empty. At the default density $p = 3$, dense rejection matches the dense-failure diagnostic in Table 3. We report both directions because uniformly spaced probe sets at different densities need not be nested: a configuration checked at a lower density may be absent from a higher-density set. Disagreement therefore reflects sensitivity

Table 8: **Sensitivity to controller settings.** Each row varies one parameter from the default. Development functional yield and independent audit retention are reported as percentages at the selected checkpoint. **Best** and second best values are highlighted.

Configuration	Development yield ↑	Retention ↑
Default	76.4	98.1
$\tau = 0.0025$	75.8	98.4
$\tau = 0.0100$	75.8	97.5
$\lambda_p = 0$	72.5	94.7
$\lambda_p = 0.5$	**76.7**	96.8
$\lambda_p = 2$	74.7	**98.6**
$\beta = 0.8$	76.1	97.5
$\beta = 0.95$	75.6	98.4

Table 9: **Sensitivity to memory retrieval capacity.** K is the maximum number of retrieved records, with a fixed 4,096-token evidence budget and unchanged memory storage limits. All metrics are percentages. **Best** and second best values are highlighted.

K	Development yield ↑	False triggers ↓	Recurrence ↓
2	73.1	8.8	11.9
4	75.3	6.0	9.2
8	**76.4**	**4.7**	**7.9**
16	76.1	5.1	8.3

to sampling locations. At $p = 12$, both rates are zero by definition whenever their conditioning sets are nonempty.

B.2 Sensitivity to controller coefficients

We vary the controller's clearance threshold τ, clearance penalty coefficient λ_p, and smoothing coefficient β one at a time around the defaults in Table 5. The patch token cap, rule limits, evaluation cohorts, and skill acceptance thresholds remain fixed. Each configuration restarts from the same initial library and uses the same three task schedules. Table 8 reports development yield at the selected checkpoint and retention on the independent audit cohort at that checkpoint. A larger τ penalizes a wider range of positive spare-clearance margins, whereas a larger λ_p strengthens the penalty's effect on the edit budget. A larger β makes the influence of earlier comparison statistics decay more slowly. These changes can affect both revision speed and capability retention. The default configuration was fixed before these analyses; the tested settings do not establish a global optimum. Setting $\lambda_p = 0$ removes only the signature-derived clearance term from the edit budget, while retaining the signatures for diagnosis.

B.3 Memory retrieval capacity and recurrence

We vary the maximum number of retrieved records over $K \in \{2, 4, 8, 16\}$ while keeping the 4,096-token evidence budget, ranking procedure, and record format fixed. Table 9 shows that increasing the retrieval limit from $K = 2$ to $K = 8$ raises development functional yield from 73.1% to 76.4%, while reducing false triggers from 8.8% to 4.7% and failure recurrence from 11.9% to 7.9%. The yield improvement therefore accompanies fewer out-of-scope rule activations and fewer recurrences of previously resolved failures. Increasing the limit further to $K = 16$ provides no additional gain: yield is 76.1%, with false-trigger and recurrence rates of 5.1% and 8.3%, respectively. These results suggest that retrieval benefits level off around eight records under the fixed 4,096-token evidence budget, supporting the default setting of $K = 8$.

B.4 Transfer across mechanism groups

After freezing the skill library selected on development data, we group the final test tasks by the mechanism labels in Table 4. The 240 tasks comprise 90 primarily revolute, 90 primarily prismatic,

Table 10: **Functional transfer across mechanism groups.** Attempts reports the number of generations per method across three runs. Functional yield (%) is averaged across runs. Gain is the improvement of AdaCraft over SkillAdam + gate in percentage points. **Best** and second best highlighted within each row.

Mechanism	Tasks	Attempts	SkillAdam + gate	AdaCraft	Gain
Revolute	90	540	64.1	**80.2**	16.1
Prismatic	90	540	62.2	**79.1**	16.9
Mixed	60	360	42.8	**61.4**	18.6
Overall	240	1,440	58.1	**75.1**	17.0

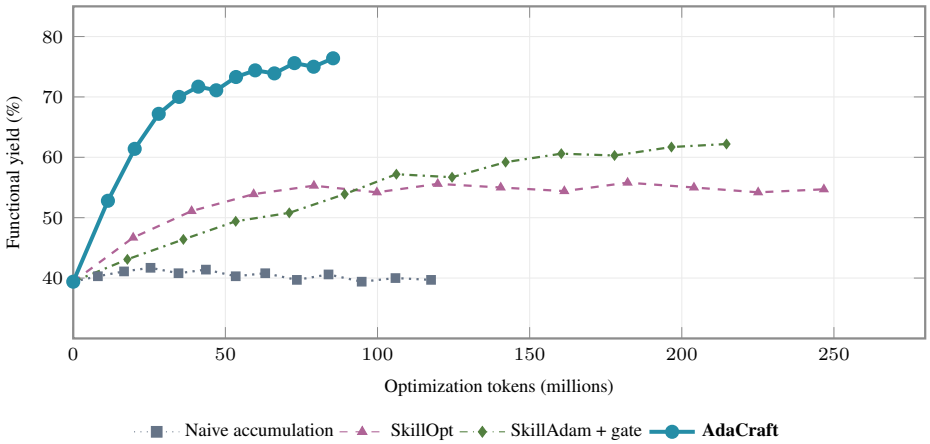

Figure 7: **Development yield versus cumulative optimization tokens.** Curves show mean functional yield and token expenditure across three runs, ending at each method's last recorded checkpoint. Token counts include generation, diagnosis, proposals, candidate evaluation, and replay, including rejected revisions and cached input. Initialization, development evaluation, and the fixed retention audit are accounted for separately. Endpoints correspond to Figure 5(a).

and 60 mixed mechanisms, yielding 540, 540, and 360 generation attempts, respectively, across three runs. We compare AdaCraft with SkillAdam + shared gate, which uses the same skill acceptance criteria. Table 10 reports the fixed task and attempt counts and the functional-yield gains for each mechanism group. The revolute, prismatic, and mixed groups contribute weights of $3/8$, $3/8$, and $1/4$, respectively, to the overall task average. Mechanism groups and generalization splits are two partitions of the same test attempts. Group-level differences alone cannot distinguish mechanism difficulty from category shift.

B.5 Development yield under cumulative token expenditure

Figure 7 shows the intermediate checkpoints underlying the final comparison in Figure 5(a). At each recorded round, the curves plot the current library's development yield against mean cumulative optimization tokens per run. This complements the revision-round axis in Figure 4 by comparing methods at similar token expenditure rather than only at the same revision round. Token totals reflect both the number of completed rounds and the cost incurred within each round.

C Additional Qualitative Results

Figures 8 to 11 present 32 articulated assets generated by AdaCraft, with eight distinct objects per figure. The examples span different object categories and mechanism structures. The first two figures show appearance, part decomposition, and joint placement; the remaining two show articulated motion across eight configurations.

Figures 8 and 9 show three views of each asset: a material rendering, a part-colored rendering, and a part-colored rendering with joint-axis overlays. Material renderings show overall shape and surface detail, while part colors distinguish individual moving links and the fixed base. Joint overlays show the position and direction of each joint relative to the geometry: red axes indicate rotation, blue axes indicate translation, and dots mark joint origins. All three views use identical geometry, pose, and camera framing. Part colors are specific to each object, and axis lengths are adjusted for visibility. Figures 10 and 11 show each asset at eight successive joint configurations. The examples include rotational and translational mechanisms, as well as assemblies with several moving links. Fixed cameras and consistent part colors make the motion of each link visible relative to neighboring parts. Percentage labels indicate progress between the selected start and end joint configurations. Reading each row from left to right shows how joint placement determines the paths of the moving parts.

D LIMITATIONS

The Homological Motion Signature summarizes connectivity on a finite graph of sampled configurations and does not establish collision freedom between probes. Skill acceptance and protected replay likewise rely on finite task sets. Under repeated adaptive testing, our uncertainty intervals guide acceptance decisions but do not provide a statistical guarantee against harmful revisions. The bounded protected reservoir also cannot represent every previously acquired capability, leaving potential regressions on untested tasks undetected. Our experiments use a fixed generation agent and software environment. Evaluating transfer to other agents and environments, and extending the framework to a broader range of tasks and mechanisms, remain directions for future work.



Figure 8: **Appearance and articulation of assets generated by AdaCraft.** Each column shows one of eight objects, with material renderings (top), part colors (middle), and joint-axis overlays (bottom). Red axes indicate rotation, blue axes indicate translation, and dots mark joint origins. Geometry, pose, and camera framing are fixed within each column.



Figure 9: **Additional geometry and joint structures generated by AdaCraft.** Each column shows an additional object with materials (top), part colors (middle), and joint-axis overlays (bottom). Red and blue axes denote rotation and translation, respectively; dots mark joint origins. The three views share geometry, pose, and camera framing.

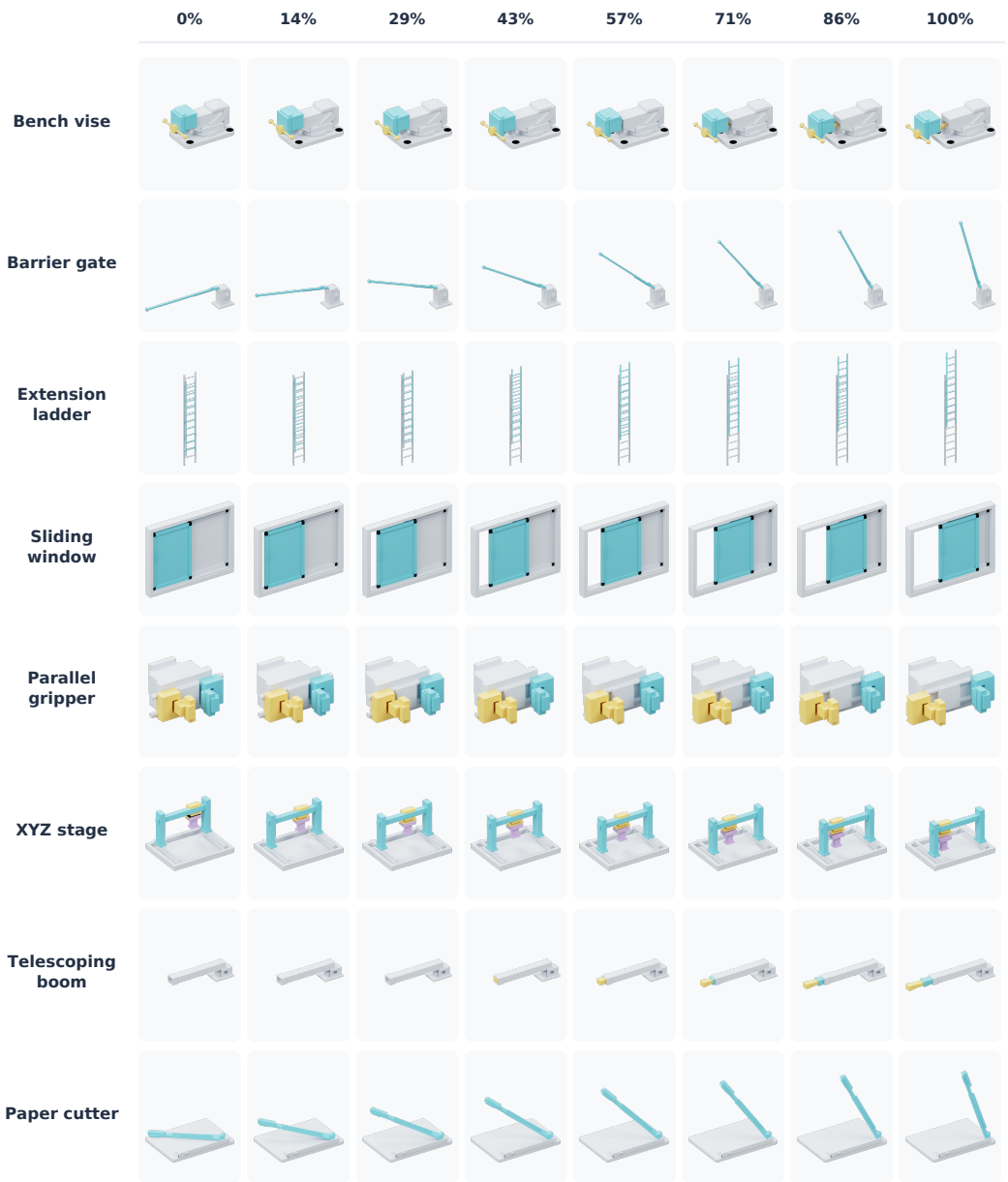

Figure 10: **Motion configurations of assets generated by AdaCraft.** Each row shows one of eight objects at eight successive joint configurations. Percentages indicate progress between the selected start and end configurations. Part colors and camera framing remain fixed within each row.

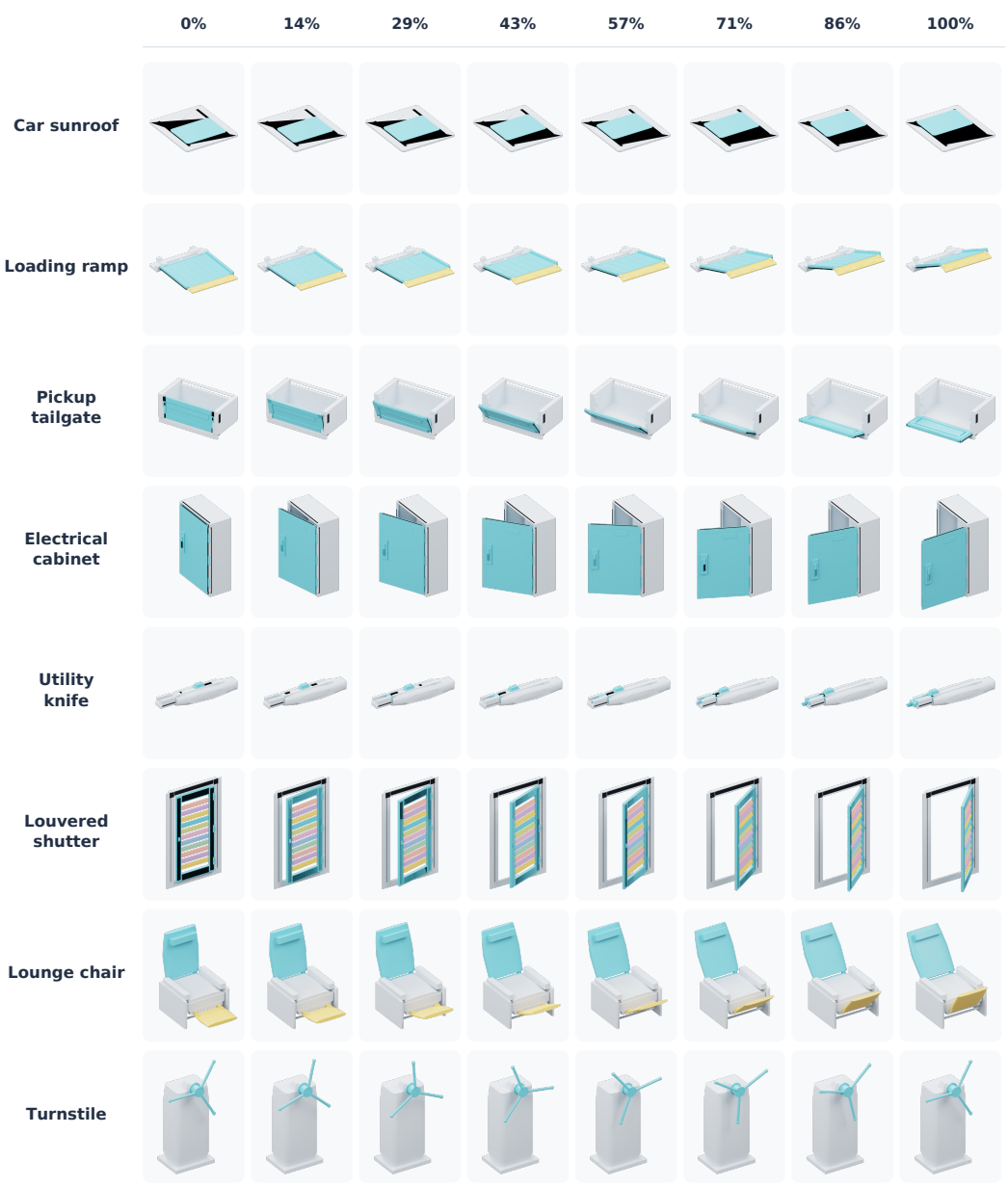

Figure 11: **Additional motion configurations of assets generated by AdaCraft.** Eight additional objects are shown, one per row, at eight successive joint configurations. Percentages indicate progress between the selected start and end configurations. Consistent part colors and fixed cameras show relative link motion.