SIMFORGE: AGENTIC GENERATION OF SIMULATION-READY WORLDS AND DEMONSTRATIONS FOR ROBOT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating a simulation-ready robot-learning world from language requires two distinct execution decisions: which construction stage should handle a failure, and how semantic intent should be expanded into many geometric instances. We present SIMFORGE, an agentic text-to-scene system built on a Graph-Native Harness. Within scene generation, a registered execution graph separates specialist execution from scene-local control: deterministic issue ownership and hard acceptance gates route failures, while bounded language model recommendations can continue or roll back only along registered transitions and retained checkpoints. For dense placement, the model specifies object groups, target quantities or density, support regions, and arrangement intent; geometry programs expand and validate instance poses using measured assets and supports. This arrangement-level interface avoids explicit per-instance pose enumeration in model outputs, addressing the observed context/OOM bottleneck while leaving geometric and simulation costs instance-dependent. Processed assets are separated from scene instances for targeted insertion, material or appearance changes, rigid or articulated substitution, and scene/task variants. An independent POLICYFORGE pipeline binds accepted worlds to tasks and robots and validates teacher demonstrations. On the common SceneEval scene-quality protocol, with the specified physics settings for COL and STB, the full system attains 89.4% and 89.7% count agreement at room and house level, respectively, with 0% detected collision and 100% stability. Internal controls separate the runtime effects of routing and batch placement. Under matched policy-training and evaluation conditions on the same LIBERO-Plus evaluation set with a common 13K checkpoint, augmented SimForge data reach 87.84% overall and 86.11% on Goal, improving over the Native reference by 11.87 and 31.28 percentage points, respectively. On LIBERO-Pro, the recorded augmented SimForge cohort reaches 48.7% overall at 13K, with effects that vary across generalization dimensions.

1 INTRODUCTION

Generating a robot-learning environment from language is not complete when a render looks plausible. The world must preserve requested objects and relations, remain physically executable, and expose reset and interaction conditions for a teacher. Existing text-to-scene and procedural systems broaden the space of worlds (Deitke et al., 2022; Yang et al., 2024; Raistrick et al., 2024), but execution remains difficult when heterogeneous construction stages fail or when a request contains many repeated objects.

These are two distinct execution bottlenecks. If a requested cup has no compatible asset, repeating placement cannot fix the missing prerequisite; the system must re-enter asset acquisition and binding. If the cup exists but intersects its support, regenerating the asset is wasteful; a scoped placement repair is appropriate. A second, complementary bottleneck arises within placement: even when the appropriate construction stage has been selected, asking a language model to enumerate every instance pose spends model capacity on repetitive geometric computation and can produce the observed context/OOM bottleneck. These observations expose two execution decisions: which construction stage should act next, and how semantic intent should be expanded into concrete instance poses.

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

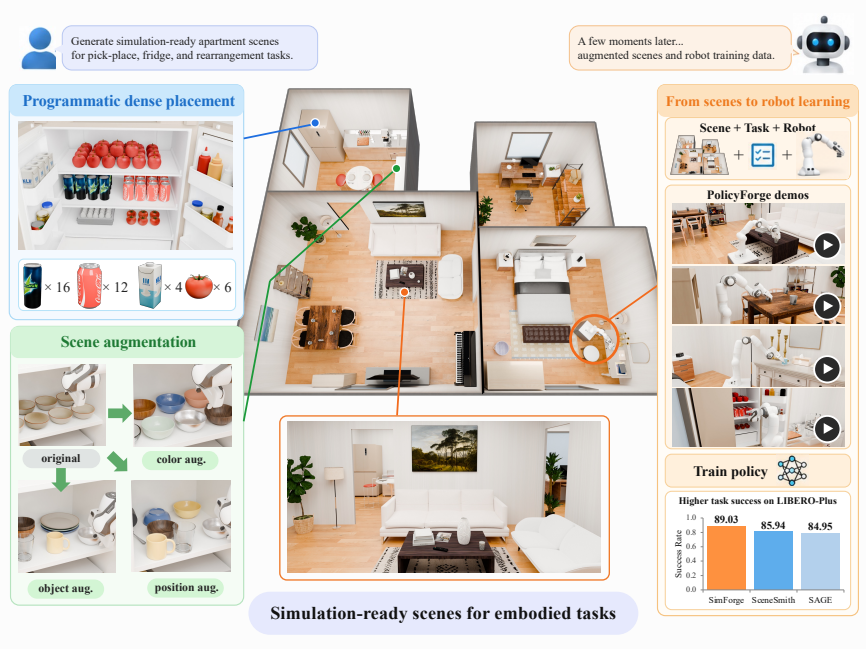

Figure 1: Overview of SimForge. A Graph-Native Harness organizes scene construction and recovery; model-guided procedural placement and targeted augmentation produce reusable worlds that PolicyForge converts into robot-learning data.

SimForge makes this execution structure explicit through a Graph-Native Harness. A registered execution graph connects layout, the scene-side asset-adapter node, placement, physics, critique, and export through legal transitions. Deterministic issue ownership and hard gates identify the responsible stage and block invalid advancement; at supported post-placement and nonterminal-critique decision points, a scene-local controller receives a compact diagnostic summary and may recommend continuation or an allowed rollback. The controller enforces registered transitions, budgets, and acceptance conditions before a specialist node receives its scoped state and performs the geometry or physics operation. The same mechanism supports ordinary construction and bounded cross-stage recovery while preserving unaffected scene state. Figure 1 summarizes this boundary: the Harness is inside scene generation, while the external asset service and downstream PolicyForge remain separate interfaces.

Within this architecture, SimForge separates semantic arrangement from repeated geometry. The model specifies object groups, target quantity or density, support region, and a supported pattern; deterministic routines use measured asset dimensions and supports to expand poses and check containment, spacing, collisions, and settling. The system returns unmet quantities and validation evidence rather than silently deleting required content. Processed assets are separated from scene instances, supporting targeted insertion, appearance or geometry substitution, and scene/task variants with mode-specific rebinding and revalidation. PolicyForge then binds an accepted world to a task and robot configuration, executes supplied teachers, and packages validated demonstrations.

The evaluation follows this causal chain. Room-level quality uses SceneEval, while house-level quality uses SceneSmith's multi-room wrapper and aggregation; COL and STB are reported under the specified physics configuration (Tam et al., 2026; Pfaff et al., 2026). The main ablation isolates Harness routing from batch placement and reports their quality and runtime effects separately; paired LIBERO cohorts test whether augmentation improves downstream data utility. Thus the results distinguish scene validity and execution cost from the usefulness of the resulting demonstrations.

Our contributions are:

(1) A Graph-Native Harness architecture for agentic text-to-scene generation. An explicit registered graph separates scene-level routing from specialist execution; deterministic issue ownership, bounded model recommendations, hard gates, and retained state support legal cross-stage recovery.

(2) Model-guided procedural placement with controllable density. A group-level intent interface lets the model specify quantities or density and arrangement patterns, while geometry-aware programs expand, validate, and report the effective repeated instances.

(3) Reusable assets and targeted scene augmentation. Asset–instance separation supports local insertion, material or appearance changes, rigid and articulated substitution, and scene/task variants with mode-specific rebinding and revalidation.

(4) Validated demonstration synthesis and downstream utility. PolicyForge binds accepted worlds to tasks and embodiments and validates teacher rollouts; controlled downstream experiments assess the learning utility of the resulting data.

2 Related Work

Graph-Native Harnesses and Agentic Scene Generation. Procedural scene generators such as ATISS, ProcTHOR, and Infinigen Indoors provide constraints for constructing diverse worlds (Paschalidou et al., 2021; Deitke et al., 2022; Raistrick et al., 2024). Holodeck combines language-guided retrieval with spatial constraint solving, while hierarchical models such as HSM organize scene structure at multiple scales (Yang et al., 2024; Pun et al., 2026). Language-model-driven builders, including LL3M and SceneWeaver, add tool use, critique, or iterative refinement (Lu et al., 2025; Yang et al., 2025); SceneSmith and SAGE connect scene construction with critics, saved artifacts, and embodied evaluation (Pfaff et al., 2026; Xia et al., 2026). SimForge focuses on the execution interface between these capabilities: layout, asset acquisition and binding, placement, checking, and export are registered as nodes in an explicit graph with legal transitions and shared state. A scene-local orchestrator selects the next registered node from intermediate feedback, while specialist nodes retain responsibility for geometry and physics. The graph therefore describes an executable construction process rather than only a scene relation or reasoning graph, and the same state representation can support normal generation and bounded recovery.

Procedural and Geometry-Aware Object Placement. Procedural layout systems expand repeated structures using support, collision, clearance, and packing constraints (Deitke et al., 2022; Raistrick et al., 2024; Pun et al., 2026). Language-guided builders provide semantic information such as object categories, room relations, and requested counts (Yang et al., 2024; Lu et al., 2025). SimForge connects these levels through a placement interface in which a model specifies categories, quantity or density, a target region, and an arrangement pattern. A geometric routine then uses measured asset dimensions, support regions, orientation constraints, and available space to expand the request into candidate poses. Support, spacing, collision, and settling checks determine which candidates become valid instances and return a diagnostic that no valid placement was found under the current finite search budget. This separation assigns semantic decisions to the model and repetitive pose computation to geometry-aware code, while leaving irregular arrangements available to model-guided handling.

Asset Reuse and Environment Augmentation. Asset repositories and 3D generation systems broaden the supply of candidate content: Objaverse provides a large annotated collection, while TRELLIS, Hunyuan3D, and SAM 3D represent complementary generation or reconstruction routes (Deitke et al., 2023; Xiang et al., 2024; Sun et al., 2025; Carion et al., 2025). Simulation use additionally requires normalization, collision and physical metadata, and compatibility with the intended interaction. MimicGen and LIBERO-Plus study systematic variation of scenes, instructions, viewpoints, or embodiments while preserving a task interface (Mandlekar et al., 2023; Fei et al., 2026). SimForge separates processed assets from scene instances: an asset stores reusable geometry, appearance, and physical metadata, whereas each instance stores its identity, pose, support, joint state, and task role. Local insertion, material/appearance substitution, and rigid or articulated substitution can therefore reuse unaffected content while rebinding changed dependencies and revalidating affected supports, neighbors, joints, and task conditions. The representation supports targeted scene variants without assuming that every field remains unchanged after an edit.

Demonstration Synthesis and Robot Learning. A scene becomes training data only after it is paired with a reset procedure, a task success predicate, a controller, and an embodiment interface. MuJoCo and Drake provide execution and verification substrates (Todorov et al., 2012; Tedrake, 2019); RoboCasa and BEHAVIOR-1K scale task and scene coverage, while LIBERO and LIBERO-Plus define task families and generalization axes (Nasiriany et al., 2024; Li et al., 2024; Liu et al., 2023; Fei et al.,

2026). Data-generation systems such as MimicGen distinguish adapting a demonstration to a changed configuration from executing a teacher in a modified world (Mandlekar et al., 2023). PolicyForge occupies this downstream interface in SimForge: it consumes an accepted world, a task specification, and a robot/runtime configuration, then uses available scripted, inverse-kinematics, or motion-planned teachers to execute, validate, and package trajectories. Policy learning remains a separate downstream stage, with policy backbones such as Diffusion Policy evaluated independently (Chi et al., 2023).

3 Problem Formulation and System Overview

3.1 Inputs, Outputs, and Module Boundaries

The central problem is not merely to render a plausible image from a text request, but to turn a request into a world that can be checked, edited, exported, and executed. The system therefore uses explicit handoffs: a request proposes semantic requirements, a scene controller selects construction stages, specialist stages change a shared scene state, and acceptance gates decide whether that state can become a world.

SimForge separates scene construction from downstream policy use through interfaces. A text request T specifies a room or house context and optional object, support, or task requirements. The scene controller invokes one scene-side asset-adapter node; retrieval, generation, canonicalization, collision decomposition, and physical enrichment remain inside that node's service scope. The controller receives candidate records and evidence, producing scene S and, after geometric, physical, critique, and package gates, world W. Thus the asset adapter supplies candidates, whereas the scene graph decides when a candidate is needed and whether its bound instance is acceptable. Targeted augmentation and benchmark substitution are separate operation contracts. PolicyForge consumes accepted worlds through a downstream interface.

PolicyForge receives W, task specification Q, and embodiment/runtime configuration Γ. Here Q names goals, target roles, interaction requirements, and success predicates; Γ names the robot adapter, simulator, observation/action schema, reset settings, teacher, and rollout budget. Their resolution against object and joint identities defines

$$B = \mathrm{Bind}(W, Q, \Gamma), \qquad \mathcal{D} = \mathcal{P}(B). \tag{1}$$

The binding operation resolves object identities and runtime parameters. World acceptance checks geometric and physical requirements, teacher execution checks task completion in the accepted world, and the resulting trajectories are packaged for learning.

The export boundary is explicit. Blender supports scene construction, observation, and canonical visual export; the implementation emits GLB/GLTF-backed assets and packages including USD/USDA, MJCF XML, and Isaac scene-builder artifacts. These products are consumed by the supported simulator adapters, including the Isaac Sim–to–MuJoCo transfer path.

3.2 Scene Representation and Operating Modes

A scene is represented as

$$S = (R, O, A, X, H, J, P, M), \tag{2}$$

where R is spatial topology, O stable object identities, A visual and collision assets, X poses, H support, containment, and adjacency relations, J articulation metadata, P physical properties, and M materials. Execution evidence, routing history, budgets, and snapshots surround S rather than adding geometry. A critique can change the execution path without changing an object identity. Density is defined over the usable floor or support area; candidate count and final valid count remain distinct. The model selects high-level placement intent and procedural mode, while geometry computes each pose and validation determines what is admitted to S.

The scene execution graph $\mathcal{G} = (V_{\mathrm{exec}}, E_{\mathrm{exec}})$ is distinct from H. The graph factory constructs the registered nodes and permitted edges once; runtime traversal only visits those edges. Registered nodes are layout, the scene-side asset-adapter, placement, physics, critique, export, and orchestration. Deterministic gates and bounded model advice choose runtime paths and revisits from this fixed graph. The resulting causal loop is explicit: feedback identifies a failed condition, the controller chooses a legal successor, the specialist changes only its owned state, and the validator decides

whether execution may continue. Configuration changes modes, thresholds, model settings, and budgets, while changing topology requires modifying graph construction. Asset acquisition and binding are re-enterable operations of the asset-adapter node, not extra graph vertices.

The scene entry generates a world from text in room or house modes. A saved-stage replay utility supports asset, placement, physics, critique, export, and rendering stages. These operating modes distinguish a new request from a controlled edit of an existing state: the former constructs a scene, whereas the latter retains unaffected state and revalidates the affected closure. Targeted augmentation and benchmark substitution use separate contracts (Section 4.3); PolicyForge consumes the resulting world downstream.

4 SIMFORGE

SimForge organizes text-to-scene execution around a Graph-Native Harness. Section 4.1 defines the execution architecture and its recovery boundary. Section 4.2 separates semantic arrangement from repeated geometric computation. The final sections describe reusable scene variants and validated demonstrations.

4.1 GRAPH-NATIVE HARNESS ARCHITECTURE FOR AGENTIC TEXT-TO-SCENE GENERATION

4.1.1 EXPLICIT EXECUTION GRAPH AND SPECIALIST INTERFACES

The first design decision is to make execution structure explicit rather than hide it in a fixed call sequence. The graph factory constructs the registered node and edge set before execution; runtime traversal cannot add edges. The graph contains layout, one scene-side asset-adapter node for acquisition and binding, placement, physics, critique, and export. It is distinct from the object-support graph: the execution graph describes permitted calls, whereas the support graph describes physical dependencies. The runtime state is summarized by $z_t = (S_t, n_t, e_t, h_t, b_t, \kappa_t)$, where n_t is the current node, e_t is stage evidence, h_t is routing history, b_t is the remaining budget, and κ_t contains scoped snapshots. The current graph factory fixes adjacency; runtime configuration changes stage settings and budgets, while the controller selects only a registered successor during traversal.

Different failure owners can therefore select different legal successors even at the same point in the nominal workflow.

4.1.2 SCENE-LOCAL ORCHESTRATION AND ADAPTIVE ROUTING

After placement or a nonterminal critique, the scene-local orchestrator reads the stage, critique scores, physics stability and removal counts, recent errors, and rollback history. Routing is mixed: deterministic issue ownership and hard gates define the admissible successors, while a bounded model call may suggest or rank a registered successor within the remaining budget. Missing assets route to the asset-adapter node, missing instances and support or physical faults route to placement, severe layout faults route to layout, and unresolved terminal checks block export. If model advice is unavailable, a deterministic stage default is used; geometry, physics, required-content, and export gates still decide acceptance. The handoff is explicit: *observe evidence* $\rightarrow$ *choose a legal node* $\rightarrow$ *execute the node on its scope* $\rightarrow$ *revalidate*. A policy score is not used to override a hard scene gate.

4.1.3 STATE PRESERVATION AND BOUNDED RECOVERY

The same handoff supports distinct causes. A child without a usable support first returns to its support owner; a missing compatible asset re-enters the asset-adapter node for acquisition and binding; an existing asset that collides returns directly to placement. The selected node receives the prior state and an issue scope, modifies only its owned records, and returns evidence for continuation, rollback, or termination. Recovery is conservative: the scope widens only when evidence marks a dependent closure. Checkpoints preserve unaffected content, while neighbors are rechecked when contacts or occlusions may have changed. Repeated actions on the same scene version are no-ops and exhausted required repairs do not become successes. The implementation uses scoped snapshots and stage-level invalidation, preserving unaffected content while rechecking the affected dependency closure.

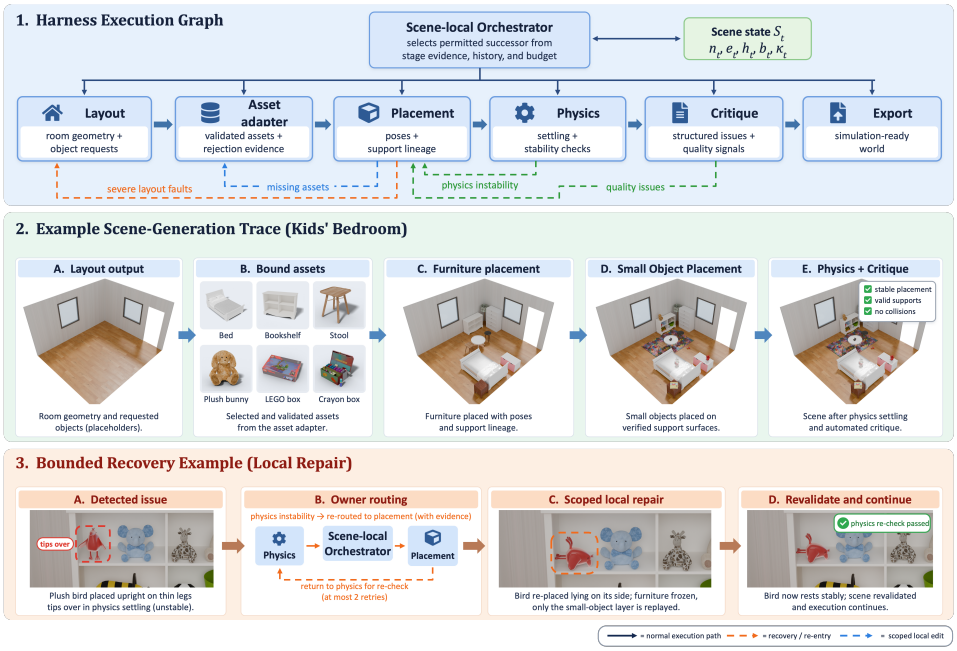

Figure 2: Graph-Native Harness execution and bounded recovery. The scene-local orchestrator selects registered successors from stage evidence, history, and budget. The trace separates normal execution from scoped re-entry: a physics issue is routed to placement, the affected layer is edited, and physics and critique are rerun before export continues.

Figure 2 summarizes the resulting execution trace: normal calls follow the registered graph, while a diagnosed issue sends the controller back to the responsible node, applies a scoped edit, and revalidates before construction continues.

4.2 Model-Guided Procedural Placement and Density Control

Figure **??** shows that The need for this split is operational. A naive placement interface asks the model to return one pose record per requested instance. As the target count grows, the prompt and response grow with it and may encounter deployment- specific context or memory limits; serial pose generation also makes wall time grow roughly with the number of instances. SimForge keeps the model decision at the arrangement level and lets a geometry routine expand that decision in batch, so density can be targeted without making every coordinate a language-model output.

4.2.1 Arrangement Intent and Parameters

Once the orchestrator selects placement, the model expresses a dense arrangement without becoming the source of every coordinate. The model outputs object or asset types, a target quantity or density, a support or region, an arrangement pattern, and supported spacing or orientation constraints. It can request one of the arrangement patterns supported by the implementation, with a target quantity or density and the associated spacing constraints. Irregular cases remain model- or tool-dependent; the interface does not assume one pattern or one call serves an entire scene. Figure 4 illustrates this decision boundary in a refrigerator: repeated cans and produce share a support and use dense placement, whereas door condiments and an egg tray retain independent placement because their semantics and supports differ.

4.2.2 Geometry-Grounded Batch Expansion

The placement routine turns this intent into candidate poses using measured asset dimensions, support surfaces, free space, and existing instances. For a support region Ω_j, footprint F_i, yaw θ, and clearance disk B_δ, a sufficient feasible translation set is

$$\mathcal{F}_{ij}(\theta) = \{t : t + (R_\theta F_i \oplus B_\delta) \subseteq \Omega_j\} = \Omega_j \ominus (R_\theta F_i \oplus B_\delta). \tag{3}$$

The implementation approximates this set with support hulls, footprint proxies, allowed rotations, and deterministic candidates. It rejects support, spacing, collision, and settling violations and records a capacity shortfall when no feasible slot remains in that support region. Thus "place 12 mugs on the measured shelf" can be reported at group level as requested $= 12$, candidates $= 16$, and accepted $= 10$ after validation; candidate slots are never counted as accepted objects. The shortfall is local to the measured region and does not establish that the whole scene is infeasible.

4.2.3 Feasibility, Density Control, and Irregular Cases

The placement routine emits batched per-instance poses from the same intent, assets, regions, and validators. Density is a target over usable support area, subject to clearance, collision, and settling checks; the accepted count can therefore be below the requested count. The routine reports target quantity or density error, requested/candidate/accepted counts, support and collision outcomes, and the resulting geometry.

4.3 Reusable Assets and Targeted Environment Augmentation

Figure 3 shows that once placement has produced valid instances, processed assets and scene instances can be shared across new scenes under explicit compatibility checks. The supported augmentation modes are deliberately distinct. (i) *Support-aware local insertion* adds small objects to measured surfaces such as tables, counters, shelves, workbenches, cabinet tops, or bedside surfaces while preserving a declared scene basis and its protected objects. (ii) *Material/appearance substitution* changes the visual or surface specification of a compatible asset while retaining its geometry, pose, support, and task identity. The new material record is checked for simulator compatibility; if it changes physical parameters such as friction, the corresponding physics gates are reopened. (iii) *Rigid substitution* replaces the geometry of a named target while rebinding its frame, scale, collision and inertial properties, reset pose, and task identity. (iv) *Articulated substitution* replaces links and joints for furniture or appliances, including limits, axes, initial state, collision geometry, and interaction regions; it preserves only the declared high-level goal, such as open or close. (v) *Scene and task cousins* vary declared environment or task fields while keeping their own fixed fields explicit.

A processed asset stores reusable geometry, appearance/material, scale, collision, physical, and articulation metadata. Each scene instance keeps an independent identity, pose, support, joint state, and task role. Candidate eligibility is a service-level outcome; scene acceptance still requires instance-level checks. Every edit rebinds changed dependencies and revalidates the affected closure, including neighbors when contacts or occlusions can change. Thus reuse avoids unnecessary reacquisition or reconstruction without sharing mutable placement or task state.

4.4 PolicyForge: Demonstration Synthesis for Robot Learning

PolicyForge is a downstream interface, not a node in the scene graph. It receives an accepted world W, task specification Q, and robot/runtime configuration Γ as parallel inputs and forms

$$B = \mathrm{Bind}(W, Q, \Gamma), \tag{4}$$

which resolves target identities, reset conditions, success predicates, observation/action conventions, and embodiment adapters. Supplied scripted, inverse-kinematics, or motion-planned teachers then execute replayable resets and produce candidate trajectories. New task families need compatible teachers and predicates; new robots need adapters. A valid episode must pass world acceptance, reset, teacher execution, and packaging checks. Its end-to-end probability is written as

$$p_D = \Pr(E_0) \prod_{k=1}^{3} \Pr\left(E_k \mid \bigcap_{\ell=0}^{k-1} E_\ell \right), \tag{5}$$

without assuming independence. PolicyForge produces validated trajectories; it does not introduce a new policy loss or an automatic PolicyForge-to-scene repair loop.

5 Experimental Evaluation

We evaluate scene quality, downstream data utility, and the two execution choices that define SimForge. Tables 1–3 report the corresponding comparisons.

Table 1: Scene quality at two evaluation scales under source-defined protocols. Room-level rows use SceneEval on 179 prompts; house-level rows use SceneSmith's multi-room protocol on 31 prompts (Tam et al., 2026; Pfaff et al., 2026). The displayed columns follow the automated evaluation schema reported by SceneSmith, but the two strata are kept separate and are not pooled or re-normalized. Metrics: #Obj is instantiated-object count; CNT is count agreement; ATR is attribute correctness; OOR is object–object relation correctness; OAR is object–architecture relation correctness; ACC is accessibility; NAV is navigability; COL is collision rate; STB is stability; and OOB is out-of-bounds rate. Values are means ± 95% confidence intervals when available; ↑ / ↓ indicate that higher/lower is better. The SAGE† row is a 30-scene source-native cohort and is not mapped to the SceneEval columns; its reported collision and stability entries use SAGE definitions (Xia et al., 2026).

Method	#Obj	CNT count↑	ATR attributes↑	OOR object–object↑	OAR object–architecture↑	ACC accessibility↑	NAV navigability↑	COL collision↓	STB stability↑	OOB out-of-bounds↓
Room-level generation (179 scenes)										
HSM	22.7±2.6	60.6±4.2	61.5±7.3	30.6±5.7	66.2±6.6	88.9±1.7	99.4±0.4	20.6±3.0	45.2±3.9	5.5±1.1
Holodeck	23.0±1.4	44.2±4.3	44.4±7.2	16.8±4.6	38.0±7.1	84.9±1.4	99.6±0.2	12.3±2.3	31.9±2.7	0.8±0.4
SceneWeaver	13.5±1.0	41.8±3.9	29.1±6.7	15.3±4.2	31.6±6.5	77.5±2.7	98.1±1.2	12.5±2.5	37.3±4.4	0.0±0.0
I-Design	13.0±1.0	69.3±4.0	50.3±7.2	28.6±5.5	66.6±6.7	70.1±2.8	95.9±1.4	3.0±1.5	60.8±4.6	4.3±2.0
LayoutVLM (Cur.)	11.2±1.6	41.0±4.0	25.6±6.6	14.1±4.0	22.4±6.1	93.8±1.9	99.7±0.2	25.9±4.4	19.4±3.6	6.2±2.1
LayoutVLM (Obj.)	14.2±1.7	55.5±4.7	34.3±7.0	20.7±5.0	17.4±5.6	91.5±1.6	98.7±0.8	28.9±3.5	8.1±1.7	5.1±1.4
SceneSmith	71.1±13.0	82.9±3.4	74.4±6.2	67.6±5.8	80.6±5.5	83.4±1.5	97.6±1.1	1.2±0.6	95.6±1.7	0.2±0.4
SimForge	67.6±5.1	89.4±4.2	76.8±7.1	73.1±5.4	79.3±5.1	98.1±1.4	98.4±1.2	0.0±0.0	100.0±0.0	0.0±0.0
SAGE†	48.2	—	—	—	—	—	—	1.9†	99.9†	—
House-level generation (31 scenes)										
Holodeck	81.3±18.3	64.6±6.8	43.2±27.6	30.1±10.1	58.4±6.3	79.4±3.0	99.6±0.2	3.8±3.1	17.9±5.6	0.3±0.2
SceneSmith	214.1±60.9	83.3±6.8	66.8±24.6	76.6±9.2	82.6±5.8	79.9±2.0	96.5±2.1	0.9±0.5	79.8±9.1	0.6±0.3
SimForge	242.3±32.9	89.7±6.4	70.3±26.7	78.9±10.3	81.9±6.1	93.2±2.7	98.1±1.3	0.0±0.0	100.0±0.0	0.0±0.0

5.1 Setup and Matched Baseline Comparisons

Room-level quality uses the SceneEval semantic evaluator on 179 prompts. House-level quality uses the same semantic metrics on 31 multi-room prompts, with SceneSmith's multi-room wrapper and aggregation retained (Tam et al., 2026; Pfaff et al., 2026). We report the two strata separately and keep SAGE in its source-native schema. COL and STB are reported with the specified physics checks.

The internal controls pair requests and seeds where possible and hold assets, simulators, validators, hardware, tolerances, caches, graph permissions, and budgets fixed. Workflow uses a fixed stage order, Harness routes among the same registered successors, and placement changes only the arrangement interface. Scene construction and PolicyForge use separate ledgers.

For downstream learning, every condition trains the same dual-stream World Action Model (WAM): a video diffusion transformer and an action diffusion transformer initialized from Wan2.2-TI2V-5B. LIBERO-Plus uses a fixed evaluation subset. The subset and checkpoint protocol are held constant across data conditions. The detailed input/output interface, optimizer, precision, data ratio, hardware, and checkpoint schedule are given in Appendix F;

5.2 Scene Quality and Simulation Readiness

Table 1 uses the source metric definitions: CNT, ATR, OOR, OAR, accessibility, NAV, COL, STB, and OOB (Tam et al., 2026; Pfaff et al., 2026). The #Obj column reports content scale rather than a quality objective. At room level, SimForge improves over SceneSmith by 6.5 points in CNT, 5.5 in OOR, 14.7 in accessibility, 0.8 in NAV, and 4.4 in STB, while COL and OOB are zero; ATR and OAR remain close. At house level, the gains are 6.4, 2.3, 13.3, 1.6, and 20.2 points for the same metrics, again with zero COL and OOB, while OAR is comparable.

5.3 Downstream Robot Benchmarks and Augmentation

Table 2 compares paired data conditions under a common policy recipe. For LIBERO-Plus, all rows use the fixed 13K checkpoint: the augmented SimForge condition reaches 87.84 overall, 80.31 Object, 90.48 Spatial, 86.11 Goal, and 95.00 LIBERO-10, compared with 75.97 overall for Native. The corresponding overall gain is 11.87 percentage points; the Goal gain is 31.28 points. The fixed-subset definition and complete learning curves are given in Appendix F. LIBERO-Pro rows use the declared checkpoint records in the appendix, including the augmented SimForge condition at

Table 2: Paired training-data cohorts under fixed policy and training settings. LIBERO-Plus uses the same evaluation subset and a common 13K checkpoint. LIBERO-Pro retains the recorded checkpoint for each directly identified cohort; the augmented SimForge row is reported at 13K. Bold marks the augmented SimForge condition.

	LIBERO-Plus					LIBERO-Pro					
Training-data condition	Overall	Object	Spatial	Goal	LIBERO-10	Overall	Object	Swap	Language	Task	Environment
Baseline (native data)	75.97	84.13	81.25	54.83	84.52	46.7	79.5	3.0	93.5	18.0	39.5
SAGE-10k without augmentation	78.76	86.51	85.83	58.30	85.32	47.9	77.0	4.0	93.5	21.0	44.0
SAGE-10k + SimForge augmentation	85.14	73.75	90.87	84.52	92.08	48.1	81.5	2.5	97.5	17.5	41.5
SceneSmith without augmentation	78.96	65.25	87.70	82.54	80.83	45.3	80.5	0.0	92.5	20.0	33.5
SceneSmith + matched augmentation	86.24	73.36	90.08	90.08	92.08	48.0	80.5	1.0	97.0	16.5	45.0
SimForge without augmentation	81.06	64.09	87.70	82.14	91.25	47.1	80.5	4.0	96.0	18.5	36.5
SimForge with augmentation	**87.84**	**80.31**	**90.48**	**86.11**	**95.00**	**48.7**	**81.0**	**1.5**	**98.0**	**18.5**	**44.5**

Table 3: Matched internal routing/placement controls. CNT, OOR, ACC (accessibility), COL, and STB use the main scene definitions; ↑ is better except for COL and time. The four rows hold specialist nodes, validators, checkpoints, and budgets fixed while toggling Harness routing and batch placement.

Configuration	Harness	Batch placement	CNT ↑	OOR ↑	ACC ↑	COL ↓	STB ↑	Wall time (h:m:s) ↓
Workflow only	No	No	73.4±3.6	48.7±5.9	81.7±2.7	0.0±0.0	97.4±3.5	2:40:53
Workflow + procedural placement	No	Yes	76.8±4.1	51.2±5.3	81.9±1.6	0.0±0.0	97.8±2.7	1:36:42
Graph-Native Harness + per-instance placement	Yes	No	85.9±4.1	72.4±4.3	98.0±1.4	0.0±0.0	100.0±0.0	2:29:21
Full SimForge	Yes	Yes	89.4±4.2	73.1±5.4	98.1±1.4	0.0±0.0	100.0±0.0	**1:20:44**

13K. The Isaac Sim–to–MuJoCo path tests cross-simulator transfer for the matched cases; it is not a simulator-independent guarantee.

5.4 Architectural, Placement, and Runtime Ablations

Table 3 compares four internal configurations under the same request cohort and validation protocol. . The reported times are 2:40:53, 1:36:42, 2:29:21, and 1:20:44, respectively. At fixed per-instance placement, Harness routing raises OOR from 48.7 to 72.4 and accessibility from 81.7 to 98.0, while the reported wall time changes by about 7.2to 89.4 and reduces the reported wall time from 2:29:21 to 1:20:44 (45.9without Harness it raises CNT from 73.4 to 76.8 and reduces time from 2:40:53 to 1:36:42. All four internal SimForge configurations have COL=0.0; workflow STB is 97.4/97.8 and Harness STB is 100.0. These controls support a division of labor: routing mainly changes which construction stage acts and improves semantic/accessibility outcomes, whereas batch placement removes repeated pose-generation work.

6 Limitations and Broader Impact

The Graph-Native Harness selects only registered nodes from visible evidence, checkpoints, and budget; scoped invalidation gives sufficient reuse conditions without guaranteeing an optimal path. Procedural placement is bounded by implemented modes, support geometry, and free space, while reuse requires semantic, geometric, physical, and interaction compatibility. Material changes that affect friction or restitution reopen the corresponding physics checks, and geometric edits may enlarge revalidation through neighboring contacts.

PolicyForge remains bounded by supplied teachers, reset contracts, embodiment adapters, and success predicates. Its results do not guarantee sim-to-real performance or automatic feedback.

7 Conclusion

SimForge is an agentic text-to-scene system organized by a Graph-Native Harness: the graph routes failures and preserves valid state, procedural placement turns arrangement intent into checked geometry, and asset–instance separation enables controlled scene variants. An independent PolicyForge pipeline converts accepted worlds into validated robot demonstrations. Source-protocol scene scores and paired LIBERO results show that these design choices produce usable worlds and improve downstream data utility, while the conclusions remain bounded by registered nodes, supported placement modes, supplied teachers, and simulator-specific validation contracts.

REFERENCES

Nicolas Carion et al. Sam 3d: Segment anything in 3d. *arXiv preprint*, 2025.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Matt Deitke et al. Procthor: Large-scale embodied ai using procedural generation. In *Advances in Neural Information Processing Systems*, 2022.

Matt Deitke et al. Objaverse: A universe of annotated 3d objects. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: A progressive robustness benchmark for visual-language-action models. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Chengshu Li et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024.

Bo Liu et al. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, 2023.

Yujie Lu et al. Ll3m: Large language 3d modelers. *arXiv preprint*, 2025.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning*, 2023.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems*, 2024.

Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. In *Advances in Neural Information Processing Systems*, 2021.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes. *arXiv preprint arXiv:2602.09153*, 2026.

Hou In Derek Pun, Hou In Ivan Tam, Austin T. Wang, Xiaoliang Huo, Angel X. Chang, and Manolis Savva. Hierarchical scene motifs for multi-scale indoor scene generation. *arXiv preprint arXiv:2503.16848*, 2026.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21783–21794, 2024.

Jing Sun et al. Hunyuan3d 2.0: Scaling diffusion models for high resolution 3d assets. *arXiv preprint*, 2025.

Hou In Ivan Tam, Hou In Derek Pun, Austin T. Wang, Angel X. Chang, and Manolis Savva. Sceneeval: Evaluating semantic coherence in text-conditioned 3d indoor scene synthesis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 7355–7365, 2026. doi: 10.1109/WACV61042.2026.00710.

Russ Tedrake. Drake: Model-based design and verification for robotics. *Software documentation*, 2019.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

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

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. *arXiv preprint arXiv:2412.01506*, 2024.

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *Advances in Neural Information Processing Systems*, 2025.

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. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

AI Assistance Statement

Language-model tools were used to assist with prose editing, LaTeX organization, and presentation checks. The authors reviewed the manuscript, implementation descriptions, figures, and reported results and remain responsible for all technical claims and data.

A Evaluation Protocol and Controlled Configurations

A.1 Scene-control configurations

Fixed-order control keeps the specialists' local repair loops and processes the common stages in a fixed order. Rule-routed control selects among the same legal successors using fixed rules. The full controller combines model recommendations with deterministic guards. Specialist nodes, validators, cached assets, checkpoints, observations, recovery interfaces, and budgets are shared across configurations; only the routing policy changes. Local repair remains enabled in the fixed-order control.

For recovery comparisons, each condition receives the same checkpoint, ordinary diagnostics, and history. Fault labels, injected causes, and reference repairs remain evaluator-only. A route is valid when it satisfies the original requirements within the declared budget; a designated reference path is not required.

A.2 Accounting and uncertainty

A run manifest identifies the request, seed, graph and checkpoint hashes, asset/cache snapshot, visible observations, route decisions, node calls, repair scopes, validation outcomes, termination status, and artifact hashes. Cost events include failures and retries; scene construction and PolicyForge use separate ledgers. Cache setup records cold and warm conditions separately.

Request-level aggregates are formed separately for room and house strata. Robot episodes sharing worlds or task templates remain grouped by their source world or template.

A.3 Scene-quality strata

The room-level request set follows the 179-prompt SceneEval evaluation used by SceneSmith: 94 SceneEval-100 prompts, 50 type-diversity prompts, 25 object-density prompts, and 10 themed prompts. The house-level set contains six SceneEval-100 house prompts and 25 additional house-level prompts. We keep these strata separate because the latter uses SceneSmith's multi-room aggregation and source-specific reporting. Room-level entries therefore follow SceneEval, whereas house-level entries follow the SceneSmith protocol (Tam et al., 2026; Pfaff et al., 2026).

A.4 Evidence-scoped capability comparison

The following comparison is evidence-scoped rather than a pooled benchmark. A checkmark denotes a capability directly documented by the cited method or implementation, a triangle denotes a narrower or adjacent mechanism, and a dash means that the available source does not establish the criterion. The bold SimForge rows identify the two method contributions evaluated in the main paper.

Capability comparison. Evidence-scoped comparison with representative scene-generation systems. A ✓ marks a capability directly documented as part of the cited method, △ marks an adjacent or narrower mechanism, and – means that the cited source does not establish the strict criterion in this row. A dash is not a claim that the system cannot provide the capability. Bold rows mark SimForge's two primary method contributions; the table is not a pooled benchmark.

Capability	ProcTHOR	Holodeck	SceneWeaver	SceneSmith	SAGE	**SimForge**
Text-driven scene request	–	✓	✓	✓	✓	✓
Graph-Native Harness architecture	–	△	–	△	△	✓
Model-guided procedural placement & density	–	△	△	△	△	✓
Scene-local orchestration	–	–	✓	✓	✓	✓
Feedback-conditioned recovery	–	△	✓	△	✓	✓
Geometry-grounded placement	✓	✓	△	✓	✓	✓
Asset–instance state separation	△	△	△	△	△	✓
Targeted local augmentation	–	–	△	△	✓	✓
Demonstration / policy evaluation	–	–	–	✓	✓	✓

B Scene Orchestrator Implementation Details

B.1 Execution state and registered topology

The scene pipeline stores serialized scene/layout data alongside stage metrics, failures, asset requests/results, placement targets, repair issues/history, scene version, physics results, critique history, checkpoint snapshots, export status, and an orchestrator decision log. These fields implement the execution-state summary z_t in Section 3.2. Object-support lineage belongs to S, whereas permitted calls belong to $\mathcal{G}$. A route log records the stage, average score, decision, reason, and next node. Repair history records issue fingerprints, actions, scene versions, charged attempts, and outcomes in a separate record.

The graph factory registers layout, asset, placement, physics, critique, export, and orchestrator nodes. Layout is the entry; every construction/validation stage except export returns to the orchestrator, whose registered successors are those six stages or termination. Export is terminal. The nominal path is layout, asset service, placement, physics, critique, and export, with scoped asset/placement revisits for deferred children or repairs. Graph adjacency is fixed in the current factory. Runtime configuration provides models, backends, room/house mode, validation flags, thresholds, and iteration budgets; it does not accept an arbitrary adjacency specification. Changing topology requires editing graph construction, while routing chooses an already permitted successor.

B.2 Model context, prompt, and authority

The top-level model is called only after completed placement or nonterminal critique. Its current role default is GPT-5.2, selected through the configurable orchestrator model field, with temperature 0.1 at this call site. The system prompt identifies a scene-pipeline scheduler, describes continuation and rollback choices, advises placement rollback for low scores attributed to spatial logic and layout rollback for severe layout faults, and supplies the global rollback count. The requested response is a JSON decision and a one-sentence reason. After placement, the advertised options are continue and rollback to layout; after critique, rollback to placement is also advertised. The code interprets these recommendations through stage-specific branches and hard gates rather than granting the model direct scene-editing tools.

The user prompt contains the current stage, critique dimensions and mean, physics stability rate, number of removed objects, and the three most recent errors. The full serialized scene, repair history, route log, checkpoints, and rejection history remain available to deterministic scheduling and bookkeeping. History-conditioned no-op detection is implemented in the repair scheduler. A failed model call yields a continuation recommendation, while geometry, physics, required-object, and terminal gates still execute.

Specialist model settings are separate from this top-level call. The current configuration supplies role-specific designer, critic, parser, asset-router, and validator model fields, with GPT-5.2 defaults. Local

loops have their own prompts, tool limits, critique histories, convergence criteria, and regression handling. For example, the layout designer can create or revise rooms and openings through architectural tools; its critic evaluates the current layout, and its planner chooses revision, finish, or rollback. These local roles do not acquire the top-level controller's cross-stage authority. A local stop caused by convergence or iteration exhaustion is not, by itself, strict world acceptance.

B.3 Node inputs, editable state, and returned evidence

The stage contracts below use the scene-side asset request adapter. Candidate ranking, generator selection, canonicalization, and collision processing remain internal to the asset service.

Node	Inputs and editable state	Outputs and feedback
Layout	Text and room/house mode; room geometry, openings, architectural relations, and object requests	Layout, core requests, deferred manipuland requirements, or layout diagnostics/failure.
Asset adapter	Core, deferred, or repair request workset; named acquisition records	Validated asset results, failed requests, rejection evidence, and request identities.
Placement	Scene, validated assets, core/manipuland scope, target IDs, optional repair patch	Instantiated poses/support lineage, placement residuals, deferred requests, or unresolved repair issues.
Physics	Current instantiated scene and collision/physical representations; settling and permitted local corrections	Settled state, instability/removal/displacement evidence, and residual geometry violations.
Critique	Current scene and rendered evidence; local improvement within its execution contract	Scores, feedback, modifications, continuity/history, quality state, and stop reason.
Export	Final scene and validation evidence; package construction and terminal checks	JSON/USD/MJCF artifacts, validation/export status, and explicit package errors.

Construction and repair use different permissions. Initial placement may create the requested scene content. A routed repair of existing objects carries a patch with base scene version, issue IDs, named targets/owners, permitted actions, and verification requirements. The implemented action whitelist is move and rescale. Asset geometry must remain verified, and support-dependent edits require a verified parent surface. Before and after the edit, the scope guard computes support descendants of the named targets. It rejects unrelated-object changes, object additions/removals, and architectural changes, restoring the pre-edit scene. Export checks are backend-specific: emitting MJCF does not establish target-simulator loading or execution parity.

B.4 Routing, recurrence, rollback, and termination

Structured issues contain category, severity, owner, affected request/object/room/support identifiers, and evidence. Missing assets route to the scene asset adapter. Uninstantiated objects, invalid support, placement faults, and physical faults route to placement with a core or manipuland scope. An invalid-support issue without usable child lineage routes to core placement so the owning furniture can be repaired before real-surface child planning resumes. The routing helper also recognizes layout and export ownership. Recognition by the helper does not create a universal export-recovery loop: the registered export node terminates the top-level graph.

For example, a required cup with no acquired candidate produces a missing-asset issue, request-scoped acquisition, and a return to placement. If the asset exists but has not been instantiated, the next owner is placement rather than acquisition. A child with no valid parent surface requires a support-owner repair before another child placement proposal. Critic additions are resolved against request identities and validated assets before the scheduler selects acquisition or placement.

The issue fingerprint combines category with semantic/request, room, furniture, and surface identities. Scheduling increments a charged-attempt count; repeating the same action for the same scene version is recorded as a no-op and does not reset the budget. The current exhaustion threshold is three charged scheduling requests, with exhaustion marked before another owner execution is scheduled. Independently, the top-level rollback branches allow fewer than two previous rollbacks before

scheduling another. Required failures that exhaust their repair path block successful completion; nonrequired enrichment has distinct degradation handling.

After placement, residual hard geometry or missing required content blocks physics progression. After physics, residual violations return to placement, including a manipuland scope when required children are missing. After critique, structured owner repairs are handled before an export decision, and current placement metrics are recomputed before promotion. A terminal critique budget permits at most one terminal owner repair and retains the degraded-quality status. These controls prevent repeated unresolved issues from being counted as successful completion merely because a model recommends continuation.

Placement rollback restores an available scoped snapshot, preferring manipuland-specific or core-specific checkpoints as appropriate; unavailable required snapshots cause failure. Layout rollback clears dependent asset and later-stage results. Placement rollback clears stale physics and critique evidence. This invalidation is stage-based and can exceed the smallest object dependency closure. Final package validation remains necessary after local repair. Equation (11) states a sufficient condition for evidence reuse under complete dependencies; the implementation does not establish that all gate read sets are tracked.

B.5 Boundary of the demonstration interface

PolicyForge is not a node or successor in this graph. It consumes accepted worlds, task goals and predicates, robot/runtime adapters, replayable resets, scripted or motion-planned rollouts, and trajectory validation. Resolving $B = \mathrm{Bind}(W, Q, \Gamma)$ requires selecting a task-family teacher, mapping target roles to exported object/link/joint identities, and supplying motion parameters, reset conditions, and success/termination predicates. For articulated substitution, these include the replacement's joint type, axis, limits, initial/target state, and interaction region; a retained open/close goal does not determine the new trajectory.

The current scene implementation does not contain a PolicyForge teacher registry or an automatic natural-language-to-teacher resolver. Teacher selection and parameter binding are interface requirements of the downstream module. Additional task families and robots need compatible teachers, predicates, and adapters. No automatic rollout-to-scene repair edge is included.

C Scene-evaluation protocols and metrics

C.1 Evaluation strata and source protocols

The room-level stratum contains 179 prompts: 94 SceneEval-100 room prompts, 50 type-diversity prompts, 25 object-density prompts, and 10 themed prompts. We use the SceneEval protocol for this stratum, including its semantic fidelity and spatial plausibility measurements (Tam et al., 2026; Pfaff et al., 2026). The house-level stratum contains 31 multi-room prompts: six SceneEval-100 house prompts and 25 house-level prompts. These scenes are evaluated with SceneSmith's house-level multi-room protocol and its reported aggregation rules (Pfaff et al., 2026). The main table keeps the source column names for readability, but the two strata are reported separately and are never pooled or re-normalized.

C.2 Metric dictionary

Table 4 gives the source-defined meanings used by the main scene-quality table. Rates are computed over the eligible objects, relations, or scenes specified by the evaluator; higher is better except for COL and OOB.

SceneEval obtains CNT, ATR, OOR, OAR, and ACC by checking the generated scene against the prompt constraints; ACC is accessibility, not architectural faithfulness. NAV measures connected free-floor space. SceneSmith augments the semantic evaluator with physics checks: COL counts objects with more than 1 mm of prohibited penetration after collision processing, while STB runs a 5-second gravity settling simulation with a 1 ms step and accepts an object when its displacement is below 1 cm and its rotation change below 0.1 rad. OOB is computed from boundary sampling and requires at least 99% of sampled rays to hit the permitted floor region. These operational details

Table 4: Source-defined metrics used in Table 1. Room-level values follow SceneEval; house-level values follow SceneSmith's house-level protocol.

Metric	Definition	Direction
#Obj	Number of instantiated objects, reported as content scale rather than a quality objective.	–
CNT	Satisfaction of requested object-count constraints.	↑
ATR	Satisfaction of requested object-attribute constraints.	↑
OOR	Satisfaction of requested object–object relations.	↑
OAR	Satisfaction of requested object–architecture relations.	↑
ACC	Accessibility of functional sides and relevant free space.	↑
NAV	Ratio of the largest connected free-floor region to total free floor space.	↑
COL	Fraction of evaluated objects with prohibited penetration after physics processing.	↓
STB	Fraction of evaluated objects that satisfy the post-settling pose and support thresholds.	↑
OOB	Fraction of evaluated objects outside the permitted floor or scene boundary.	↓

follow the SceneEval/SceneSmith protocol (Tam et al., 2026; Pfaff et al., 2026) and are not replaced by the local diagnostic formulas below.

C.3 SimForge diagnostics and acceptance accounting

The following formula describes a SimForge internal relation diagnostic; it is not used to reinterpret published source rows. For a required relation set R^* and realized relations R, the diagnostic is $\mathrm{OOR}_{\mathrm{local}} = 100|R \cap R^*|/\max(|R^*|, 1)$. When a published row uses a different evaluator or denominator, its value is reported as supplied rather than silently recomputed. CNT in the main tables is always the source-defined count-constraint rate; no alternate normalized count formula is used to replace it.

Scene acceptance uses all requested scenes as its denominator, including empty outputs, timeouts, and requests that exhaust the repair budget. For request i, let $a_i = 1$ only when the final artifact passes every required scene gate (required content and cardinality, geometry/support, collision and settling, structural/critique, and export/package validation), and let $a_i = 0$ otherwise. The scene-level acceptance rate is

$$\mathrm{Accepted} = 100\,\frac{\sum_{i=1}^{N_{\mathrm{req}}} a_i}{N_{\mathrm{req}}}. \tag{6}$$

An empty output, timeout, or exhausted repair budget contributes zero to the numerator but remains in N_{req}. This quantity is separate from object-level quality, valid reset, teacher success, and demonstration yield.

Let $\mathcal{O}_e$ be eligible instantiated objects and $\mathcal{P}_{\mathrm{bad}}$ the prohibited penetrating pairs after settling. Intended support and declared joint contacts are excluded by the fixed contact rule. SimForge also records object- and pair-level collision diagnostics,

$$\mathrm{COL}_{\mathrm{obj}} = 100\frac{|\{i \in \mathcal{O}_e : \exists j, \{i,j\} \in \mathcal{P}_{\mathrm{bad}}\}|}{|\mathcal{O}_e|}, \qquad \mathrm{COL}_{\mathrm{pair}} = 100\frac{|\mathcal{P}_{\mathrm{bad}}|}{|\mathcal{P}_e|},$$

where $\mathcal{P}_e$ is the eligible tested-pair set. These diagnostics are kept distinct from the source COL definition in the main table. Support validity includes a parent identifier, a measured support surface, a local pose, footprint containment, intended contact, and post-settling stability; articulated checks additionally inspect joints, limits, reset, and required motion.

C.4 Routing, resource accounting, and yield

A local-fix case is accepted when the current specialist resolves its violation; a cross-node case is accepted when another scene node changes the relevant state. A repeated unresolved attempt returns to an equivalent unresolved issue after repair. Regression records a newly violated protected condition. Unrecoverable requests terminate explicitly.

For $N > 0$ requests, accepted count A, and total scene costs c_i,

$$C_{\mathrm{request}} = \frac{\sum_{i=1}^{N} c_i}{N}, \qquad C_{\mathrm{accepted}} = \frac{\sum_{i=1}^{N} c_i}{A} \quad (A > 0). \tag{7}$$

Each c_i includes orchestration, asset-service calls charged to scene construction, specialist execution, repair, validation, and export. Wall-clock time, tokens, and GPU time remain in their native units. PolicyForge execution and policy training are separate from this scene-control ledger.

An augmentation record contains required targets, protected content, accepted edits, and the resulting validation state. A demonstration record contains the episode request, reset result, rollout result, and packaged observation–state–action trajectory. The event chain is described in Equation (5); its attempt-level estimator is given in Equation (17).

D Recovery Cases and Failure Analysis

The recovery taxonomy separates three diagnostic situations. In a local-fix case, the current specialist can repair the violated condition within its permissions. In a cross-node case, a different scene node must change the relevant state, such as repairing an absent support before retrying a deferred child. In a same-symptom/different-cause pair, an observed placement failure may arise from a poor pose or from incompatible support geometry. The categories distinguish interventions that act on different state owners.

A separate unrecoverable stratum contains contradictory requirements, unavailable mandatory assets, or requests beyond the permitted tools and budget. Correct behavior is an explicit non-success or stop. Deleting a required object, relaxing the requested cardinality, or suppressing a failed gate cannot count as recovery. Multiple valid paths may exist for recoverable cases, so route agreement with a single reference trace is secondary to final validity and resource use.

Case records contain the starting checkpoint, controller-visible diagnostics, selected node and reason, directly edited records, rechecked neighbors, checkpoint restoration, and final outcome. Protected conditions are recorded before and after repair, together with issue fingerprints and charged attempts.

Natural-run analysis uses the complete normal-generation set and records asset, support, placement, stability, critique, and export failures together with their subsequent routes.

E Asset Reuse and Augmentation Contracts

This appendix specifies the records and acceptance predicates used by support-aware augmentation and by the downstream transfer adapters. The contracts keep asset identity, scene-instance state, and task bindings separate; they do not turn a qualitative case into an aggregate success rate.

E.1 Canonical assets, instances, and support records

An asset record stores a stable identifier, semantic category, canonical visual and collision representations, dimensions, orientation, materials, physical parameters, and optional link and joint metadata. A scene instance adds its own identifier, world pose, parent and support identifiers, surface-local pose, task role, and settled state. Thus two copies of the same table can still have different support owners and independent task roles.

A verified support record is tied to one parent instance and stores a measured two-dimensional hull, a local frame, usable area, surface orientation, clearance margin, and a semantic compatibility prior. The local frame is defined on the instantiated parent geometry at its current pose. Tables, counters, shelves, workbenches, cabinet tops, and bedside surfaces may be eligible supports; a support inferred only from a prompt phrase or a bounding-box top is diagnostic evidence rather than a geometric support record. For a compatible support S_j with area A_j and compatibility prior π_j, sampling may use

$$w_j \propto A_j \pi_j,$$

with zero weight for an incompatible asset–support pair.

E.2 Capacity screens and surface-local poses

Before expensive pose checks, two conservative screens expose impossible requests. If U is the compatible asset set and each asset can be copied at most $k_{\max}$ times, the asset-side bound is

$$C_{\text{asset}} = |U| k_{\max}.$$

For measured usable support area A_{sup}, reference footprint A_{ref}, packing efficiency η, and spacing reserve $\rho \geq 1$, an area screen is

$$C_{\text{sup}} = \frac{A_{\text{sup}} \eta}{A_{\text{ref}} \rho}.$$

These bounds only guide request sizing; final admissibility still requires pose-level collision checks and simulator settling.

A candidate pose is sampled in the selected support frame and mapped to the world frame by

$$T_{WA} = T_{WS_j} T_{S_j A}.$$

The surface-local transform is retained after settling together with T_{WA}, so subsequent editing and task relabeling can recover the parent relation. A compact support predicate is

$$\chi_{\text{sup}}(u, S_j) = \mathbb{1}_{\text{inside}}(u, S_j) \mathbb{1}_{\text{contact}}(u, S_j) \mathbb{1}_{\text{collision}}(u) \mathbb{1}_{\text{stable}}(u).$$

Footprint containment, intended-parent contact, collision validity, and post-settling stability are separate factors: a collision-free object with no contact to its declared support is not accepted.

E.3 Settling, neighbor revalidation, and exact cardinality

A geometrically screened candidate is settled in the simulator with the current scene, including previously accepted neighbors. Acceptance checks candidate–support contact, residual penetration, displacement, tilt, and contacts with non-target layout geometry. Every neighbor whose support, clearance, or stability may have changed is revalidated at its settled pose before the candidate is committed. A later object can displace an earlier object even when the later footprint itself fits its support, so checking only the new candidate is insufficient.

For an augmented basis S_0 and accepted additions A_{acc}, the result is

$$S = S_0 \oplus A_{\text{acc}}, \qquad |A_{\text{acc}}| = N_{\text{tar}}.$$

The supplied fixed-cardinality study uses $N_{\text{tar}} = 30$ and measures cardinality after settling rather than after proposals are sampled. If a stability pass removes an object, sampling resumes within the remaining bounded budget until the target is restored or explicit exhaustion is recorded. A failed candidate is not replaced by an arbitrary visual proxy.

E.4 Material and appearance substitution

Material/appearance substitution changes a compatible asset's texture, color, shader, or other declared surface appearance while preserving its canonical geometry, collision representation, pose, support, settled state, and task identity. The replacement material must be registered with the simulator and the export validator; a purely visual change therefore reopens appearance and export checks without requiring geometric reconstruction. If a material also changes physical parameters such as friction or restitution, the corresponding physics and settling checks are reopened as well. This operation is distinct from rigid substitution, which changes geometry and collision binding, and from a scene cousin, which changes a scene-level field. Changing a support or pose creates a geometric cousin and reopens the affected validation gates. Figure 3 gives a qualitative paired view of this reuse-and-edit interface.

E.5 Rigid and articulated transfer acceptance

For a rigid transfer, the adapter aligns canonical orientation, scale, collision and inertial properties, reset pose, and task identity. If p_A is a point in the canonical asset frame and p_B is the receiving benchmark frame, the mapping is

$$p_B = s R_{BA} p_A + t_{BA},$$

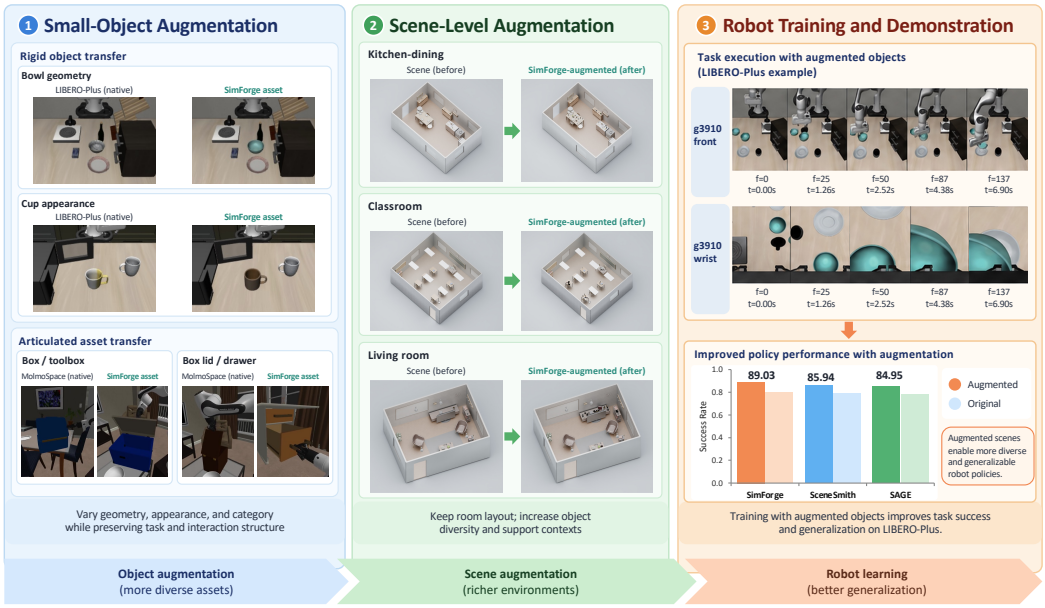

Figure 3: Examples of the reuse and augmentation interface. The three panels show rigid, material, and articulated asset transfer; scene-level edits that preserve the surrounding layout; and the downstream demonstration and learning interface. The panels summarize supported operations; quantitative evaluation is reported in the corresponding tables.

with the same transform applied to visual and collision representations. The rigid acceptance predicate requires a valid reset, grasp acquisition, lift clearance, collision-compatible transport, and terminal placement:

$$\Phi_{\text{rigid}} = I_{\text{reset}} I_{\text{grasp}} I_{\text{lift}} I_{\text{transport}} I_{\text{place}}.$$

A transfer therefore cannot be accepted from appearance similarity alone.

For an articulated transfer, the replacement record provides its own parent and child links, joint type, origin and axis, limits, initial state, collision geometry, and interaction region. The adapter rebinds these fields; the original joint type and trajectory are not assumed to carry over. For limits $q_{\min}$ and $q_{\max}$, a target state may be selected as

$$q^* = q_{\min} + \lambda(q_{\max} - q_{\min}), \qquad \lambda \in [0, 1].$$

The articulated predicate requires reproducible reset, interaction acquisition, collision-valid swept motion, and a terminal joint interval. A retained high-level open/close goal therefore still requires the replacement's own joint and interaction parameters.

F Additional Robot-Benchmark Details

This appendix gives the benchmark protocol, training-data provenance, and the complete LIBERO-Plus traces used to interpret Table 2. The LIBERO-Plus results use the same evaluation subset across training-data conditions. The three paired augmentation comparisons are kept separate: SAGE-10k is compared with its SimForge augmentation, SceneSmith without augmentation is compared with its matched augmentation, and SimForge without augmentation is compared with SimForge with augmentation. These labels describe data provenance, not difficulty levels. We omit the unrelated intermediate cohorts from the main comparison.

F.1 Benchmark protocol and training settings

The policy trained for the reported downstream comparisons is a dual-stream World Action Model (WAM). The Video DiT and Action DiT are both initialized from Wan2.2-TI2V-5B. Video DiT has

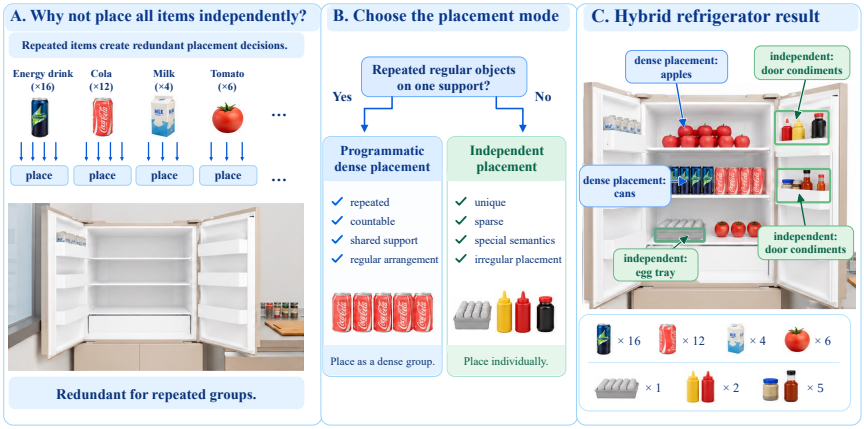

Figure 4: Hybrid placement example. Repeated objects are expanded in batches while irregular or semantically distinct objects retain independent placement.

Table 5: Policy architecture used for downstream benchmark training.

Component	Configuration
Video stream	Video diffusion transformer, initialized from Wan2.2-TI2V-5B
Action stream	Action diffusion transformer with flow-matching objective
Visual input	Two RGB views at 224×224 resolution
Proprioception	8-dimensional state vector
Action output	7-DoF end-effector action: Δxyz, Δrpy, and gripper command

30 layers with hidden size 3072, FFN size 14336, and 24 heads; Action DiT has 30 layers with hidden size 1024, FFN size 4096, and 24 heads. A frozen UMT5-XXL text encoder supplies precomputed embeddings with context length 128. The input contains front and wrist RGB views at 224×224 each (concatenated to 224×448), 33 video frames, and an 8-dimensional proprioceptive state. The output is a 7-DoF end-effector action consisting of six delta pose coordinates and one absolute gripper command (0=closed, 1=open). Proprioception is min–max normalized; the diffusion objective uses flow matching with training and inference shifts of 5.0 and 1,000 timesteps, with action-loss weight 1.0. This policy is kept fixed across data-source rows, so the comparison measures the effect of the generated demonstrations rather than a change in policy architecture.

All comparisons use the same policy architecture, optimizer, precision, batch construction, evaluation suites, and a 1:1 effective native–synthetic frame ratio within a benchmark. Main LIBERO-Plus comparisons use the common 13,000-step checkpoint; the appendix retains later checkpoints only to show learning curves. The only intended change is the training-data condition. The SAGE-10k, SceneSmith, and SimForge paired rows use the same augmentation operator; no value is inferred from legacy condition names.

F.2 Complete LIBERO-Plus learning traces

The following table reports every recorded LIBERO-Plus checkpoint for the relevant cohorts. A dash means that the run did not record that checkpoint; it is not a zero score. The main table uses the common 13,000-step checkpoint for every listed condition. The final column is retained only to show the latest recorded checkpoint for each cohort.

F.3 Complete LIBERO-Pro checkpoint traces

The recorded LIBERO-Pro runs use 1,000 episodes overall and 200 episodes per reported dimension. The main table retains the declared checkpoint for each cohort. Because the five dimensions have equal episode counts, the Native overall value in the main table is the episode-weighted mean at its final 15,000-step checkpoint (46.7 percent). The 45.9 value remains only at 13,000 steps in the

Table 6: Evaluation scale for the downstream robot benchmarks.

Benchmark	Simulator	Scale	Breakdown
LIBERO-Plus	MuJoCo	Fixed evaluation subset	Object, Spatial, Goal, LIBERO-10
LIBERO-Pro	MuJoCo	1,000 episodes	200 configurations $\times$ 5 trials; five generalization dimensions

Table 7: Training settings held fixed within each benchmark comparison.

Setting	LIBERO-Pro	LIBERO-Plus
GPUs	8 (1 node $\times$ 8)	32 (4 nodes $\times$ 8)
Batch size per GPU	16	24
Global batch size	128	768
Learning rate	1×10^{-4}	2.449×10^{-4}
Schedule / epochs	Cosine / 10	Cosine / 10
Optimizer / precision	AdamW (ZeRO-1) / bf16	AdamW (ZeRO-1) / bf16
Weight decay / accumulation	10^{-2} / 1	10^{-2} / 1
Checkpoint interval	2,000 steps	1,000 steps
Max steps	15,000	14,580
Effective native:synth. frames	1:1	1:1

learning trace and is not mixed with the final row. The complete checkpoint sequence is retained below; values are success rates in percent.

F.4 LIBERO-Pro base-suite learning traces

The preceding Pro table reports the five generalization dimensions used in the main comparison. The following table gives the separate base-suite breakdown from the same recorded runs: Object, Spatial, Goal, and LIBERO-10 each contain 250 episodes per checkpoint. "Baseline" is the native-data control. The last row is our augmented SimForge data; the source run records use a separate internal condition name.

F.5 LIBERO-Pro reporting boundary

Table 2 now includes the directly matched LIBERO-Pro measurements for all three paired cohorts. The main-table rows use SceneSmith without augmentation at step 12,000 (45.3 overall), SceneSmith + matched augmentation at step 14,580 (48.0 overall), and SAGE-10k + SimForge augmentation at step 14,580 (48.1 overall), alongside the Native, SAGE-10k, SimForge without augmentation, and SimForge with augmentation references. The complete checkpoint traces and dimension-level scores for those cohorts are given in Table 11. The final SimForge row denotes our augmented SimForge data. Its result is reported at the recorded 13,000-step checkpoint: 48.7 overall with 81.0/1.5/98.0/18.5/44.5 on Object/Swap/Language/Task/Environment. The 14,580-step SceneSmith matched-augmentation row remains separate at 48.0. The two 13,000-step rows share the same displayed aggregate and dimension values because they come from separate source records; the corresponding manifests and run identifiers should be retained with the released evaluation logs.

The transfer check uses scenes generated and validated in Isaac Sim and then exported through the scene adapter for MuJoCo evaluation. It tests the scene interface across simulators; it is not a claim that every task or robot is simulator-independent.

G Derivations and Conditions for Validity-Preserving Reuse

The following derivations characterize the geometric and accounting contracts used in Section 4. They separate exact identities under stated assumptions from their implementation through sampled poses, owner-specific repair, and recorded outcomes.

Table 8: Training-data provenance used for the robot benchmark comparison.

Row in Table 2	Source cohort	Augmentation state
Baseline	Native benchmark demonstrations	No added data
SAGE-10k	SAGE-10k aligned cohort	Before our augmentation
SAGE-10k + SimForge augmentation	SAGE-10k aligned cohort	Same SimForge augmentation pipeline
SceneSmith without augmentation	SceneSmith original demonstrations	Before the matched augmentation
SceneSmith + matched augmentation	Original SceneSmith demonstrations	Same SimForge augmentation pipeline
SimForge without augmentation	SimForge original demonstrations	Before SimForge augmentation
SimForge with augmentation	Original SimForge demonstrations	SimForge augmentation and validated variants

Table 9: LIBERO-Plus overall success rate (%) across the recorded training checkpoints.

Training-data condition	1K	3K	5K	7K	9K	11K	13K	Final
Native	35.49	62.61	64.91	71.39	77.17	76.57	75.97	74.28
SAGE-10k	48.65	59.52	69.89	74.48	77.17	76.27	78.76	78.66
SAGE-10k + SimForge augmentation	47.06	68.49	75.87	81.85	84.35	85.54	85.14	84.95
SceneSmith without augmentation	–	–	70.09	71.09	78.76	78.86	78.96	78.96
SceneSmith + matched augmentation	47.76	76.87	76.07	77.47	82.85	85.94	86.24	85.94
SimForge without augmentation	36.89	63.11	74.38	72.78	79.46	79.36	81.06	80.56
SimForge with augmentation	**59.02**	**76.97**	**82.35**	**86.64**	**87.84**	**88.83**	**87.84**	**89.03**

G.1 From footprint containment to feasible translations

Fix an asset, one measured support region Ω_j, and yaw θ. Assume the footprint $F_i \subset \mathbb{R}^2$ is nonempty and compact, expressed about the asset's placement origin in the support plane. R_θ is a planar rotation, $t \in \mathbb{R}^2$ is translation, and $B_\delta = \{b : \|b\|_2 \le \delta\}$ reserves a nonnegative planar margin. For sets U, V, define $U \oplus V = \{u + v : u \in U, v \in V\}$ and $U \ominus V = \{t : t + V \subseteq U\}$. Then

$$\begin{aligned} t + (R_\theta F_i \oplus B_\delta) \subseteq \Omega_j &\iff t + R_\theta f + b \in \Omega_j \quad \text{for all } f \in F_i,\ b \in B_\delta \\ &\iff t \in \bigcap_{f \in F_i,\, b \in B_\delta} (\Omega_j - R_\theta f - b) \\ &\iff t \in \Omega_j \ominus (R_\theta F_i \oplus B_\delta). \end{aligned} \tag{8}$$

Here $\Omega_j - a = \{x - a : x \in \Omega_j\}$. Equation (8) proves Equation (3) by applying the containment requirement to every footprint point. No convexity assumption is needed for this set identity. If the footprint is an outer proxy for the true projected geometry, containment of the proxy is a sufficient geometric screen for that projection, relative to the supplied support region. An inaccurate support region can still admit a physically unsupported pose.

For the special case of a convex polygon $\Omega_j = \{x : a_\ell^\top x \le b_\ell,\ \ell = 1, \dots, L\}$ with nonzero outward normals a_ℓ, define the support function $h_{F_i}(a) = \sup_{f \in F_i} a^\top f$. The maximum extent along each polygon normal gives

$$\begin{aligned} \sup_{f \in F_i,\, b \in B_\delta} a_\ell^\top (t + R_\theta f + b) &= a_\ell^\top t + h_{F_i}(R_\theta^\top a_\ell) + \delta \|a_\ell\|_2, \\ \mathcal{F}_{ij}(\theta) &= \bigcap_{\ell=1}^{L} \left\{ t : a_\ell^\top t \le b_\ell - h_{F_i}(R_\theta^\top a_\ell) - \delta \|a_\ell\|_2 \right\}. \end{aligned} \tag{9}$$

This follows because translation adds $a_\ell^\top t$, rotation changes the queried normal to $R_\theta^\top a_\ell$, and a radius-δ disk has maximum projection $\delta \|a_\ell\|_2$. For a rectangular support of widths w_s, l_s and an axis-aligned rectangular footprint of widths w_f, l_f, both centered at their origins, this reduces to

$$|t_x| \le (w_s - w_f)/2 - \delta, \qquad |t_y| \le (l_s - l_f)/2 - \delta.$$

A negative right-hand side makes this fixed-yaw screen infeasible; a zero value can still admit boundary placements under the non-strict inequality. Positive support area alone cannot guarantee a fit.

These relations formalize the screen rather than introduce a new solver. The inspected placement path generates finite candidates inside inset bounds and checks a dimension-based footprint against a measured hull. That proxy and its boundary-margin handling need not equal the continuous set

Table 10: LIBERO-Plus per-suite success rates (%) at the common 13,000-step checkpoint. Columns follow the main-table order.

Training-data condition	Overall	Object	Spatial	Goal	LIBERO-10
Native	75.97	84.13	81.25	54.83	84.52
SAGE-10k	78.76	86.51	85.83	58.30	85.32
SAGE-10k + SimForge augmentation	85.14	73.75	90.87	84.52	92.08
SceneSmith without augmentation	78.96	65.25	87.70	82.54	80.83
SceneSmith + matched augmentation	86.24	73.36	90.08	90.08	92.08
SimForge without augmentation	81.06	64.09	87.70	82.14	91.25
SimForge with augmentation	**87.84**	**80.31**	**90.48**	**86.11**	**95.00**

Table 11: LIBERO-Pro success rates (%) at the recorded checkpoints. Overall uses 1,000 episodes; each dimension uses 200 episodes. The augmented SimForge row denotes our data.

Cohort	Step	Overall	Object	Swap	Language	Task	Environment
SceneSmith without augmentation	12000	45.3	80.5	0.0	92.5	20.0	33.5
SceneSmith + matched augmentation	7000	45.5	73.5	1.0	93.5	18.0	41.5
SceneSmith + matched augmentation	9000	45.9	75.0	2.5	96.5	16.5	39.0
SceneSmith + matched augmentation	11000	47.4	79.5	0.5	98.5	16.5	42.0
SceneSmith + matched augmentation	13000	48.7	81.0	1.5	98.0	18.5	44.5
SceneSmith + matched augmentation	14580	48.0	80.5	1.0	97.0	16.5	45.0
SAGE-10k + SimForge augmentation	7000	46.7	77.0	2.5	97.0	19.5	37.5
SAGE-10k + SimForge augmentation	9000	48.6	82.0	4.0	94.5	20.0	42.5
SAGE-10k + SimForge augmentation	11000	48.2	81.0	2.5	99.0	16.5	42.0
SAGE-10k + SimForge augmentation	13000	48.2	82.0	2.5	96.0	18.5	42.0
SAGE-10k + SimForge augmentation	14580	48.1	81.5	2.5	97.5	17.5	41.5
SimForge with augmentation	13000	48.7	81.0	1.5	98.0	18.5	44.5

above for arbitrary rotated shapes. The support-frame mapping $T_{WA} = T_{WS_j} T_{S_j A}$ places each candidate in the world, while semantic compatibility, intended-parent contact, collision, settling, and neighbor revalidation remain separate requirements. Containment does not establish static equilibrium, dynamic stability, reachability, or the existence of a successful manipulation trajectory.

G.2 Dependency closure and conditional reuse of gate evidence

Let $\mathcal{Z}$ be a finite universe of named records, including scene fields, derived support geometry, task bindings, and any simulator/configuration inputs read by a gate. Absence of a record is itself a value, so insertion and deletion count as changes. A directed dependency $u \to v$ means that changing u may invalidate v; mutual interaction can therefore require edges in both directions. For a repair, the relation must conservatively cover dependencies in both the old and candidate states, including new contacts. A spatial-neighbor list that misses a newly created interaction is insufficient for the following argument.

The closure used by the main text is

$$\mathcal{C}_0 = \Delta, \qquad \mathcal{C}_{k+1} = \mathcal{C}_k \cup \{v : \exists u \in \mathcal{C}_k,\ (u, v) \in E_d\}, \qquad \mathcal{C}_\Delta = \bigcup_{k \geq 0} \mathcal{C}_k. \tag{10}$$

Starting from the directly edited records Δ, Equation (10) is monotone: $\mathcal{C}_k \subseteq \mathcal{C}_{k+1} \subseteq \mathcal{Z}$. Each strict expansion adds at least one record, so the sequence reaches a fixed point after at most $|\mathcal{Z}|$ expansions. It is the least dependency-closed superset of Δ: for any closed set U containing Δ, induction gives $\mathcal{C}_0 \subseteq U$ and $\mathcal{C}_k \subseteq U \Rightarrow \mathcal{C}_{k+1} \subseteq U$. Hence $\mathcal{C}_\Delta \subseteq U$. Cycles change the resulting scope but do not prevent termination.

Suppose a cached gate g has a complete read set $\mathcal{R}_g$ and can be written as a deterministic function $\widetilde{g}$ of those records. The gate's version, thresholds, random inputs if any, and external simulator state must either be fixed or included among those inputs. Assume also that the repair preserves record values outside $\mathcal{C}_\Delta$. Restriction to the read set then yields the main-text preservation condition

$$\mathcal{R}_g \cap \mathcal{C}_\Delta = \varnothing \quad \Longrightarrow \quad g(\mathbf{z}') = g(\mathbf{z}). \tag{11}$$

and the more explicit identity

$$\begin{aligned}\mathcal{R}_g \cap \mathcal{C}_\Delta = \varnothing &\Longrightarrow \mathbf{z}'|_{\mathcal{R}_g} = \mathbf{z}|_{\mathcal{R}_g} \\ &\Longrightarrow g(\mathbf{z}') = \widetilde{g}(\mathbf{z}'|_{\mathcal{R}_g}) = \widetilde{g}(\mathbf{z}|_{\mathcal{R}_g}) = g(\mathbf{z}).\end{aligned} \tag{12}$$

If $g(\mathbf{z}) = 1$, its previous acceptance evidence remains valid under these conditions. The result says nothing about gates whose inputs intersect the closure; they must be reconsidered even if they ultimately return the same value. An untracked input, changed gate definition, or hidden physical coupling invalidates the reuse argument. Repeating a stochastic gate with fresh randomness is not covered by deterministic equality.

The execution graph in Section 3 schedules stages, whereas E_d expresses data dependence. The implementation realizes selected parts of the contract through parent-descendant traversal, owner routing, checkpoint preservation, and guards against unrelated mutations. The formal closure can encompass the entire scene, and final physical and package checks remain necessary.

G.3 When saved work exceeds repair overhead

For the additive comparison in Equation (13), fix one requested revision and the same acceptance requirements, comparing executions with equivalently accepted outcomes. Partition reusable rebuild/check work into non-overlapping units $\mathcal{V}$, excluding common final checks. Let $\mathcal{A}_\Delta$ contain the units that depend on invalidated records, and let $c_v \geq 0$ be the cost of unit v under matched conditions. Assume these costs are the same in the full and scoped executions. Let $h \geq 0$ be the additional scoped dependency-tracking, snapshot, and bookkeeping cost relative to the full execution. Under these assumptions, the common final-check cost cancels in the comparison:

$$\begin{aligned}C_{\text{full}} &= C_{\text{final}} + \sum_{v \in \mathcal{V}} c_v, \\ C_{\text{scope}} &= C_{\text{final}} + h + \sum_{v \in \mathcal{A}_\Delta} c_v, \\ C_{\text{scope}} < C_{\text{full}} &\Longleftrightarrow h < \sum_{v \in \mathcal{V} \setminus \mathcal{A}_\Delta} c_v.\end{aligned} \tag{13}$$

Equivalently, subtracting the matched totals gives:

$$C_{\text{full}} - C_{\text{scope}} = \sum_{v \in \mathcal{V} \setminus \mathcal{A}_\Delta} c_v - h. \tag{14}$$

This gives the stated break-even condition. If $\mathcal{A}_\Delta = \mathcal{V}$, the model predicts no saving for nonnegative overhead. A large unaffected region also need not save much if its checks are cheap. The shared final checks cancel in the difference but still contribute to both totals, limiting the relative benefit of avoiding earlier work.

Actual costs can differ when a repair changes convergence, cache behavior, concurrency, or the number of retries. Writing path-specific unit costs $c_v^{\text{full}}, c_v^{\text{scope}}$ and final-check costs $C_{\text{final}}^{\text{full}}, C_{\text{final}}^{\text{scope}}$ gives instead

$$\begin{aligned}C_{\text{full}} - C_{\text{scope}} = &\sum_{v \in \mathcal{V}} c_v^{\text{full}} - \sum_{v \in \mathcal{A}_\Delta} c_v^{\text{scope}} - h \\ &+ C_{\text{final}}^{\text{full}} - C_{\text{final}}^{\text{scope}}.\end{aligned} \tag{15}$$

The additive model describes total charged work or serial execution time. Parallel wall-clock time additionally depends on scheduling and resource contention.

G.4 Success probabilities and attempts

Let E_0, E_1, E_2, and E_3 denote world acceptance, valid reset, fixed-teacher success, and valid trajectory packaging. With $\mathcal{E}_k = \bigcap_{\ell=0}^{k} E_\ell$,

$$\Pr(\mathcal{E}_k) = \Pr(\mathcal{E}_{k-1}) \Pr(E_k \mid \mathcal{E}_{k-1}), \quad k = 1, 2, 3. \tag{16}$$

On attempt b, define $Y_b = \prod_{k=0}^{3} \mathbf{1}[E_{bk}]$. For N_{req} attempts with at most one retained episode per attempt,

$$\widehat{Y} = \frac{N_{\text{use}}}{N_{\text{req}}}, \qquad N_{\text{use}} = \sum_{b=1}^{N_{\text{req}}} \prod_{k=0}^{3} \mathbf{1}[E_{bk}]. \tag{17}$$

For a fixed number N_{req} of attempts, linearity of expectation gives

$$\mathbb{E}[N_{\text{use}}] = \sum_{b=1}^{N_{\text{req}}} \Pr(Y_b = 1), \qquad \mathbb{E}[\widehat{Y}] = \frac{1}{N_{\text{req}}} \sum_{b=1}^{N_{\text{req}}} \Pr(Y_b = 1). \tag{18}$$

When the number of launched attempts is bounded by $B_{\max}$, let L_b indicate that attempt b is launched. Then $N_{\text{use}} = \sum_{b=1}^{B_{\max}} L_b Y_b$ and $N_{\text{req}} = \sum_b L_b$.

H THE USE OF LARGE LANGUAGE MODELS

We used large language models (GPT and Claude) only as general-purpose writing assistants to refine the clarity and fluency of the manuscript, such as improving grammar, style, and consistency of academic tone. All scientific ideas, methodological designs (including the framework, algorithms, and experiments), data collection, and analysis were conceived and executed entirely by the authors. Any LLM-generated or suggested text was critically reviewed and edited to ensure accuracy and faithfulness to our original contributions.