SCENEWISE: LEARNING A DECISION-CONSEQUENCE WORLD MODEL FOR 3D INDOOR SCENE GENERATION

Anonymous authors
Paper under double-blind review

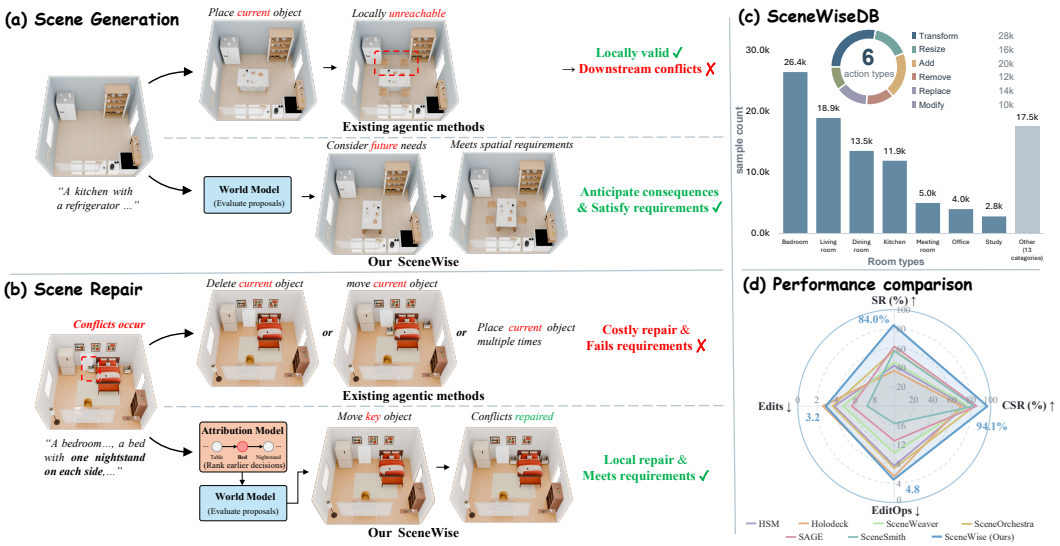


Figure 1: **SceneWise predicts decision consequences to guide scene generation and targeted repair.** (a) During generation, the world model predicts how candidate decisions affect later spatial and functional requirements, helping avoid locally valid choices that may cause downstream conflicts. (b) During repair, the attribution model ranks earlier decisions for revision, and the same world model evaluates replacements. Execution verifies local repairs before they are accepted, preserving valid scene content where possible. (c) SceneWiseDB provides decision–consequence and matched-intervention records across diverse rooms and six operations to train consequence prediction and historical attribution. (d) SceneWise achieves higher generation success and constraint satisfaction with fewer editing operations than the compared methods.

ABSTRACT

Recent agentic frameworks for 3D indoor scene generation integrate generation, placement, and editing tools to construct scenes from natural language instructions, refining intermediate layouts through feedback. However, they overlook how currently valid decisions can prevent subsequent spatial and functional requirements from being met, leading to delayed failures and costly revisions. We propose SceneWise, a framework that learns a decision-consequence world model to guide scene generation and repair. During generation, the world model predicts how candidate decisions affect placement feasibility and constraint satisfaction, guiding agents to avoid downstream conflicts. During repair, an attribution model uses the failure context and historical decision graph to rank earlier decisions for revision. The same world model evaluates replacement candidates to guide local regeneration while limiting changes to valid scene content. We further introduce SceneWiseDB, a dataset of 100k records across 20 room categories that provides consequence and intervention supervision for both models. Experiments show that SceneWise improves task success and constraint satisfaction while reducing repeated editing, and enables effective repair with limited changes to existing scenes.

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

1 INTRODUCTION

Generating 3D indoor scenes from natural language supports scalable environment creation for embodied intelligence and interactive simulation Yang et al. (2024b); Pfaff et al. (2026); Xia et al. (2026). To fulfill a user's request, a generated scene must accommodate the specified objects, satisfy their spatial relationships, and provide sufficient space for their intended use Sun et al. (2025); Yang et al. (2024a). These requirements are closely coupled: adding an object changes the space and access available to others, while layout adjustments can affect already satisfied requirements Raistrick et al. (2024). Completing such scenes reliably is therefore essential to automated scene generation. Unmet requirements and repeated revisions increase the effort needed to obtain usable scenes and limit generation scalability.

Recent agentic frameworks have progressed substantially toward this goal Feng et al. (2023); Pun et al. (2025). SceneWeaver Yang et al. (2025), SceneSmith Pfaff et al. (2026), and SAGE Xia et al. (2026) combine scene construction tools with iterative feedback to improve semantic alignment and physical validity. SceneOrchestra He et al. (2026) further improves efficiency by learning to generate complete tool-call trajectories. Nevertheless, scene construction remains vulnerable to conflicts emerging only after several interdependent decisions. For example, a dining table may fit within a room without collisions, yet leave insufficient space for six chairs and a surrounding passage. When this conflict surfaces, chair adjustments alone may fail because the earlier table choice or placement requires revision. A currently valid decision can thus obstruct later requirements, and the object exhibiting failure may differ from that requiring change.

These dependencies raise a challenge not directly resolved by current scene validity and overall trajectory quality: determining how individual construction decisions affect subsequent requirements. Before execution, a generator must anticipate which choices may restrict later placements or compromise functional requirements. After failure, it must identify earlier decisions worth revisiting and establish whether changing them actually enables recovery. World models support anticipating action outcomes and guiding decisions with predicted futures Hafner et al. (2025); Bar et al. (2025). We study both questions through decision-consequence learning: learning relationships between construction decisions and their downstream effects to guide generation and repair. This perspective makes individual decisions' effects explicit and testable to complete scenes more reliably while reducing unsuccessful adjustments and unnecessary local regeneration.

We propose SceneWise, a framework built around a learned decision-consequence world model. Given a partial scene, its construction history, remaining requirements, and a candidate decision, the model predicts changes in placement feasibility and spatial or functional constraint margins, alongside potential failures and affected objects. During generation, these predictions guide candidate selection, with shortlisted choices verified through limited execution before the next decision is committed. When a conflict occurs, an attribution model identifies historical decisions as promising revision candidates. SceneWise uses the same world model to guide subsequent construction and verifies each proposed revision by comparing outcomes of retaining and replacing the original decision under matched conditions. It then regenerates the scene's affected portion while preserving unaffected content where possible. Together, these mechanisms connect forward prediction with scene repair through a shared decision-consequence model.

Learning these capabilities requires supervision connecting decisions to subsequent outcomes. Completed scenes alone do not directly reveal counterfactual outcomes of alternative object choices and placements, or whether revising a particular historical decision would have resolved a failure. We therefore propose SceneWiseDB, a dataset organized around both evidence types. Its decision–consequence records associate alternative decisions from the same partial scene with subsequent construction outcomes, including delayed failures following initially valid choices. Its matched-intervention records associate historical revisions with observed repair effects, covering effective and ineffective changes. These records supervise consequence prediction and historical decision attribution and provide evaluation tasks for candidate selection and failure recovery.

We evaluate SceneWise on 94 room-level prompts from SceneEval Tam et al. (2026) and 100 tasks with coupled spatial and functional requirements. Comparisons with six scene-generation baselines show that SceneWise improves complete requirement satisfaction while reducing repeated editing during generation. For failed scenes, attribution-guided repair improves recovery success while limiting changes to existing objects. Ablation studies further demonstrate the contributions of

consequence-guided generation, learned attribution, and consequence-guided repair. We additionally evaluate the execution-verification calls and runtime required for repair.

Our contributions are threefold:

- We formulate decision-consequence learning for sequential indoor scene generation and develop a world model that predicts structured consequences of construction decisions for subsequent placement feasibility and spatial and functional constraint satisfaction.
- We propose SceneWise, which reuses the consequence model across generation and repair, combining forward candidate selection, learned historical decision attribution, and matched-intervention verification to recover from failures while limiting scene changes.
- We introduce SceneWiseDB, comprising 100,000 decision–consequence and matched-intervention records across 20 room categories, supporting learning and evaluation of consequence prediction, candidate selection, and failure recovery.

2 Related Work

3D Indoor Scene Generation. Indoor scene generation spans learned layout synthesis, language-guided construction, and agentic workflows. Autoregressive and diffusion models learn spatial patterns from scene datasets to generate plausible object arrangements Deitke et al. (2022); Paschalidou et al. (2021); Tang et al. (2024); Yang et al. (2024a). Language-guided methods introduce large language models' semantic knowledge, letting users specify diverse environments through natural language Yang et al. (2024b); Lin & Mu (2024). Recent agentic approaches integrate object generation, placement, and editing tools with visual and physical feedback, iteratively refining scene quality and feasibility Pfaff et al. (2026); Xia et al. (2026). SceneOrchestra He et al. (2026) improves orchestration by training on complete tool-call trajectories with trajectory preferences. Nevertheless, early construction decisions can constrain subsequent placement. SceneSmith reports earlier objects leaving insufficient space for later additions, with subsequent stages unable to readily revise previous choices Pfaff et al. (2026). We explicitly predict how candidate object choices and placements affect remaining spatial and functional requirements.

World Models for Planning. World models' predictive environment representations support action evaluation and sequential decisions. In model-based control, learned dynamics support planning through visually inferred future latent states Hafner et al. (2019), while trajectory models predict state evolution across robotic systems Wang et al. (2026a). For task-level search-based planning, language models can provide commonsense world knowledge and suggest actions Zhao et al. (2023). These approaches guide decision selection with action-outcome predictions spanning physical dynamics to task-level knowledge. For indoor scene construction, decisions' effects on unmet requirements are important targets. Our learned decision-consequence world model predicts candidate decisions' effects on later scene construction, guiding generation and repair.

Failure Attribution and Repair. Recent research traces failures to responsible agents and decisions in execution histories Zhang et al. (2025); Sun et al. (2026), addressing decisive error localization, while subsequent benchmarks examine execution-context requirements for reliable attribution Chen et al. (2026). Beyond localization, intervention-based diagnosis can guide targeted repair of generated tools Wang et al. (2026b), and visual failure analysis can guide robotic manipulation corrections Zeng et al. (2026). These studies link failure diagnosis to correction across agentic and embodied tasks. We examine delayed scene construction conflicts: initially valid object choices or placements can obstruct later requirements. Historical attribution ranks earlier decisions for revision; consequence prediction guides local regeneration with effects verified through matched execution, preserving valid scene content where possible.

3 Method

3.1 Problem Formulation

Given a natural-language description P, SceneWise constructs a scene satisfying semantic, spatial, and functional requirements through sequential placement and editing. A decision $d_{t+1} =$

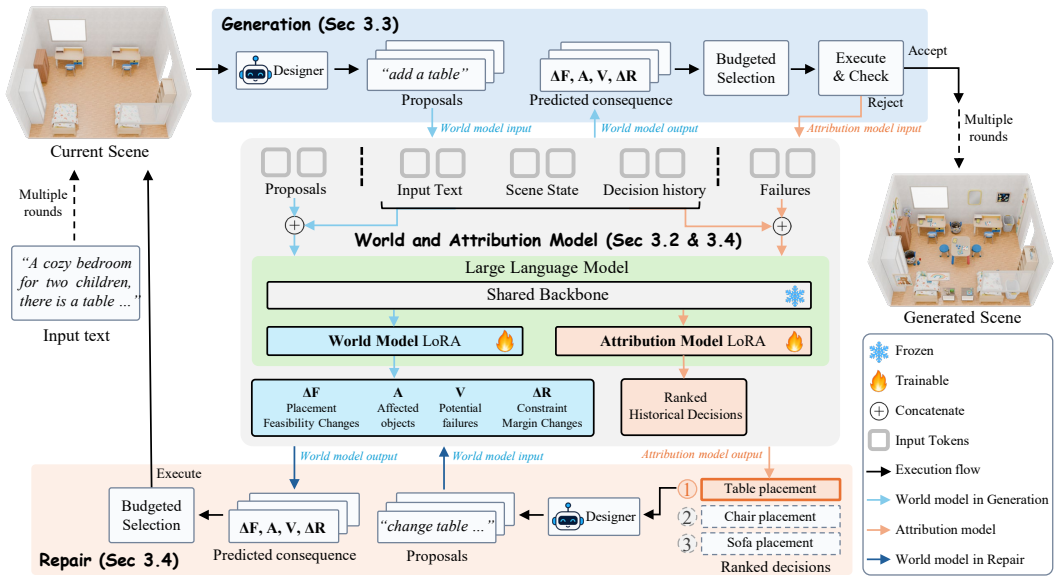


Figure 2: **SceneWise's generation and repair pipeline.** The Designer proposes candidates, the world model predicts consequences, and execution verifies shortlisted candidates. After failure, the attribution model ranks earlier decisions for revision, and the world model evaluates replacements to guide local scene repair. Both models share a frozen backbone with separate LoRA adapters.

$(a, u, \boldsymbol{\eta}, \mathcal{R})$ specifies an operation, target object identifier, operation-specific parameters, and relational constraints. For addition, u identifies a new instance and $\boldsymbol{\eta}$ specifies its asset, position, orientation, and scale; edits target existing instances. Execution gives $S_{t+1} = \text{Execute}(S_t, d_{t+1})$. We track remaining requirements Q_t and a historical decision graph $G_t = (D_t, E_t)$, where $D_t = \{d_1, \ldots, d_t\}$. Edges record spatial or support dependencies, or geometrically identified restrictions on later placement regions. Figure 2 connects consequence-guided generation with historical attribution and scene repair; both use the same world model.

3.2 Decision-Consequence World Model

The world model predicts a candidate decision's consequences under a fixed continuation policy π over at most H decisions, including the candidate:

$$\widehat{\mathcal{C}}_t^H(d) = W_\phi(S_t, Q_t, G_t, P, d), \qquad \mathcal{C}_t^H(d) = (\Delta F, \Delta R, V, A). \tag{1}$$

Hats distinguish predictions from execution-derived measurements.

Structured consequences. ΔF (*Placement Feasibility Changes*) measures changes in the fraction of valid poses for pending objects. For a candidate pose set Ω_o,

$$f_o(S) = \frac{1}{|\Omega_o|} \sum_{p \in \Omega_o} \mathbf{1}[\text{Valid}(o, p; S)], \qquad \Delta f_o(d) = f_o(S^d_{\text{eval},o}) - f_o(S_t). \tag{2}$$

Here, $\Delta F = \{\Delta f_o(d)\}_{o \in \mathcal{O}(Q_t)}$, with $\mathcal{O}(Q_t)$ denoting pending objects. $S^d_{\text{eval},o}$ is the state just before placing o, or the last valid branch state if it remains unplaced, excluding its own occupancy from the measurement. Branches share pose samples and validation rules.

ΔR (*Constraint Margin Changes*) records changes in normalized relational and functional margins, such as available minus required passage width; positive margins indicate satisfaction and negative margins indicate violation. V (*Potential Failures*) records collisions, blocked access, clearance or relational violations, and inability to continue. A (*Affected Objects*) identifies objects whose feasibility or margins change beyond thresholds, or that participate in failures; it does not establish historical causes. Measurement details are in Appendix A.2.

Training. The world model and attribution model share a frozen LLM backbone with independent LoRA Hu et al. (2021) adapters ϕ and θ. The consequence adapter learns structured outputs from alternative candidates executed in copies of the same partial scene, using token-level negative log-likelihood. Targets contain measured consequences and distinguish completion, horizon termination, and failure. Supervision and inference-time verification use the same π and H. Training objectives and configurations are given in Appendix A.1.

3.3 Lookahead Scene Generation

The *Designer* proposes K decisions from the current task and scene. Immediate collision, boundary, and hard-constraint checks produce U_t^{valid}. The world model ranks surviving candidates by predicted risk:

$$R(\mathcal{C}) = \alpha R_F(\mathcal{C}) + \beta R_R(\mathcal{C}) + \gamma R_V(\mathcal{C}), \qquad R_F = \frac{\sum_{o\in\mathcal{O}(Q_t)}[-\Delta f_o]_+}{|\mathcal{O}(Q_t)|}. \tag{3}$$

Here, $[z]_+ = \max(z, 0)$; R_R averages decreases in comparable constraint margins, and R_V penalizes failures. Weights and thresholds are selected on validation data; detailed scoring conventions are in Appendix A.2.

Budgeted Selection retains up to B low-risk candidates. *Execute & Check* verifies each through an H-step rollout and selects the lowest measured-risk candidate among failure-free branches $\mathcal{K}_t^{\text{ok}}$:

$$\mathcal{K}_t = \text{TopB}_{d\in U_t^{\text{valid}}}[-R(\widehat{\mathcal{C}}_t^H(d))], \qquad d_{t+1}^* = \arg\min_{d\in\mathcal{K}_t^{\text{ok}}} R(\mathcal{C}_t^H(d)). \tag{4}$$

Branches violating hard constraints or unable to continue are excluded. Failure-free horizon termination is admissible but does not establish task completion. Only d_{t+1}^* is committed; subsequent rollout decisions serve evaluation. We update S_t, Q_t, and G_t and repeat until all requirements and final validation pass. Scene repair is invoked if bounded retries yield no valid candidate, all shortlisted branches fail, or post-execution checks detect a violation.

3.4 Historical Attribution and Repair

SceneWise repairs a failed, possibly incomplete state S_T by revising historical decisions and regenerating affected content. Scene repair denotes this overall process; local regeneration is the step that rebuilds content within the selected scope. A historical revision may change an operation or its parameters; asset replacement is only one possible operation.

Historical attribution. We retrieve decisions associated with failed objects, regions, and constraints and their ancestors in G_T. Given the task, failed scene, history, remaining requirements, graph, and failure description, the attribution adapter ranks these decisions for revision. It learns whether at least one tested revision was effective under a fixed testing budget. Negative labels indicate that completed tests found no effective replacement; untested decisions are masked. Scores determine testing order, while execution establishes effectiveness (Appendix A.1).

Consequence-guided replacements. The Designer proposes replacements for prioritized decisions. After removing content designated for regeneration, the same world model evaluates each valid replacement using the retained scene, outstanding and regeneration requirements, and updated graph. Predicted risks shortlist replacements, and local regeneration follows Section 3.3; the input schema and output meanings remain unchanged (Appendix A.3).

Matched interventions. For decision d_i, a reference branch retains it and an intervention branch applies $\widetilde{d}_i$. Both use the same continuation, requirements, regeneration scope, budget, validators, and controllable randomness, with identical content outside the scope. The effect is

$$\Delta L_i(\widetilde{d}_i) = L(S_{T,i}^{\text{ref}}) - L(S_{T,i}^{\text{int}}(\widetilde{d}_i)), \tag{5}$$

where L measures target violation severity. A replacement is effective only if the reference still fails and the intervention removes the failure or reduces its loss beyond a threshold. Improvement from reference replay alone is not credited to replacement.

Local regeneration. Let $I \subseteq \{1, \ldots, T\}$ index the decisions to revise and $\widetilde{d}_I = \{\widetilde{d}_i\}_{i \in I}$ their replacements. Local regeneration starts from $\mathcal{B}_0(I) = I \cup \mathrm{Desc}_{G_T}(I)$, where Desc returns descendant decision indices. Boundary conflicts expand both comparison branches to a common scope $\mathcal{B}(I, \widetilde{d}_I)$. For $S' = S_T^{I \leftarrow \widetilde{d}_I}$, we seek

$$\begin{aligned} \min_{I, \widetilde{d}_I} \quad & \lambda_I |I| + \lambda_B |\mathcal{B}(I, \widetilde{d}_I)| + \lambda_D D(S_T, S') \\ \text{s.t.} \quad & L(S') \leq \epsilon, \qquad \mathrm{Valid}_{\mathrm{all}}(S') = 1. \end{aligned} \tag{6}$$

The costs count direct revisions, regenerated decisions, and scene changes D, including pose changes, additions, removals, and replacements. Validation checks currently evaluable task and hard constraints; unmet requirements remain active. Search tests single-decision replacements before joint revisions and commits the lowest-cost verified feasible repair found within budget. The budgeted search does not guarantee a globally optimal repair. If no feasible repair is found, no unverified changes are committed. Appendix A.3 details scope expansion and search.

3.5 SceneWiseDB

SceneWiseDB contains 100,000 records across 20 room categories and six operations: transform, resize, add, remove, replace, and modify. All six are permitted during generation and repair and use the decision schema in Section 3.1. Decision–consequence records $\mathcal{D}_{\mathrm{con}}$ group alternative branches from the same partial scene, recording contexts, continuations, structured consequences, and termination status. They include immediately valid decisions that later fail. Matched-intervention records $\mathcal{D}_{\mathrm{int}}$ link historical revisions to verified effects, regeneration scopes, scene-change costs, and validity outcomes. Both effective and ineffective completed tests are retained; interrupted or untested interventions receive no effectiveness label. Operation definitions, dataset statistics, replay metadata, and annotation details are provided in Appendix B.

Table 1: **General scene quality on room-level prompts.** SceneWise scores highest on OOR and OAR. All scores are percentages. The complete results are in Table 11.

Method	CNT↑	ATR↑	OOR↑	OAR↑	ACC↑	NAV↑
HSM	60.6±4.2	61.5±7.3	30.6±5.7	66.2±6.6	**88.9±1.7**	99.4±0.4
Holodeck	44.2±4.3	44.4±7.2	16.8±4.6	38.0±7.1	84.9±1.4	**99.6±0.2**
SceneWeaver	41.8±3.9	29.1±6.7	15.3±4.2	31.6±6.5	77.5±2.7	98.1±1.2
SceneOrchestra	50.0±4.0	40.0±6.5	25.0±5.0	42.0±6.0	82.0±2.0	98.5±0.8
SAGE	80.0±3.5	72.0±6.0	62.0±5.8	75.0±5.5	86.0±1.8	98.0±0.9
SceneSmith	82.9±3.4	**74.4±6.2**	67.6±5.8	80.6±5.5	83.4±1.5	97.6±1.1
SceneWise (Ours)	**83.0±3.3**	74.0±6.0	**74.0±5.5**	**82.0±5.2**	88.0±1.5	98.5±0.8

4 Experiments

4.1 Experimental Setup

Data and comparisons. We train both adapters on their respective records in SceneWiseDB's 80,000-record training split. Evaluation covers 94 SceneEval room prompts and 100 coupled-requirement tasks, each with a fixed, manually verified checklist. HSM (Pun et al., 2025), Holodeck (Yang et al., 2024b), SceneWeaver (Yang et al., 2025), SceneOrchestra (He et al., 2026), SAGE (Xia et al., 2026), and SceneSmith (Pfaff et al., 2026) retain their pipelines and share task inputs and the evaluator. Controlled ablations test generation guidance, learned attribution, and repair guidance; world-model ablations test history inputs and multi-branch data (Appendix C.2).

Metrics. General quality uses SceneEval Tam et al. (2026) metrics. We report SR (all original requirements satisfied), CSR (mean satisfied fraction per attempt), EditOps (editing operations after initial placement), Edits (distinct edited objects), and prediction AUROC and failure-event F1. Repair also reports $\mathrm{SR}_{E \leq 5}$ (success with at most five checkpoint objects edited), execution-verification Calls, and elapsed Time in minutes. Appendix C.3 gives full definitions.

Table 2: **Generation under coupled spatial and functional requirements.** On 100 prompts, SceneWise achieves the highest SR and CSR with the fewest editing operations. Results include controlled ablations.

Method	SR↑	CSR↑	EditOps↓	Edits↓
Comparison with scene-generation methods				
HSM	42.0±8.4	74.1±3.0	7.6±1.1	3.8±0.6
Holodeck	36.7±8.0	69.8±3.4	5.4±0.8	**2.6±0.4**
SceneWeaver	45.3±8.5	76.4±3.1	10.3±1.5	4.7±0.7
SceneOrchestra	58.0±8.3	83.2±2.7	7.1±1.0	3.6±0.5
SAGE	62.0±8.1	85.3±2.5	12.8±1.8	5.6±0.8
SceneSmith	57.3±8.4	82.8±2.8	16.4±2.2	7.2±1.0
SceneWise (Ours)	**84.0±6.4**	**94.1±1.8**	**4.8±0.7**	3.2±0.5
Controlled ablations				
w/o consequence-guided generation	76.0±7.4	90.8±2.1	11.7±1.6	5.8±0.8
w/o learned attribution	78.0±7.1	91.6±2.0	9.6±1.3	5.0±0.7
w/o generation guidance and attribution	68.0±8.0	86.9±2.6	16.1±2.1	7.4±1.0
w/o consequence-guided repair	79.3±7.0	92.5±1.9	7.8±1.1	4.3±0.6
SceneWise (Full)	**84.0±6.4**	**94.1±1.8**	**4.8±0.7**	**3.2±0.5**

Table 3: **Consequence prediction and lookahead selection.** The full model scores highest on all four metrics. Prediction uses held-out records; selection uses 200 shared partial scenes with identical candidates, continuation policy, and budgets. AUROC is on a 0–1 scale; F1, SR, and CSR are percentages.

Predictor	Consequence prediction		Lookahead selection	
	AUROC↑	F1↑	SR↑	CSR↑
Prompted backbone	0.72±0.04	61.4±3.5	73.3±5.5	88.2±2.3
w/o decision history	0.83±0.03	75.6±2.9	80.7±4.7	91.7±1.9
w/o multi-branch data	0.80±0.03	71.2±3.2	77.3±5.1	90.1±2.1
Full consequence model	**0.93±0.02**	**87.9±2.0**	**88.3±3.8**	**96.0±1.4**

Protocol. Each task uses three seeds. Coupled-task comparisons share the verification interface and budget; controlled ablations also share the backbone, proposal generator, and tools. Repair retains the original checklist. All attempts, including failures, enter task-level rates and edit/cost averages. Tables report means with 95% confidence intervals; bold and underlining mark the best and second-best results within each section. Implementation details are in Appendices A and C.

4.2 Scene Generation Results

Table 4: **Repair success and cost.** Each method is evaluated on 100 shared failed checkpoints with a 60-call verification budget per attempt. SR, CSR, and $\mathrm{SR}_{E\leq 5}$ are percentages; the latter requires success with at most five checkpoint objects edited. Time is in minutes.

Repair strategy	SR↑	CSR↑	$\mathrm{SR}_{E\leq 5}$ ↑	Edits↓	Calls↓	Time↓
Latest-dec. repair	48.0±8.4	82.4±2.9	35.3±8.0	4.6±0.6	22.2±1.8	9.4±0.9
Visible-conf. repair	54.0±8.4	85.7±2.6	43.3±8.3	3.9±0.5	21.3±1.7	9.4±0.8
Hist. search w/o attr.	69.3±7.8	90.1±2.2	51.3±8.4	4.4±0.6	24.3±2.0	10.8±1.0
SceneWise (Ours)	**86.0±6.1**	**96.2±1.5**	**78.0±7.1**	**2.6±0.4**	**16.0±1.4**	**8.4±0.7**

Quantitative comparison. Table 1 shows improved spatial relations and accessibility. Relative to SceneSmith, SceneWise raises OOR from 67.6% to 74.0% and ACC from 83.4% to 88.0%. Appendix D.1 reports results for all metrics. On coupled tasks (Table 2), SceneWise reaches 84.0%

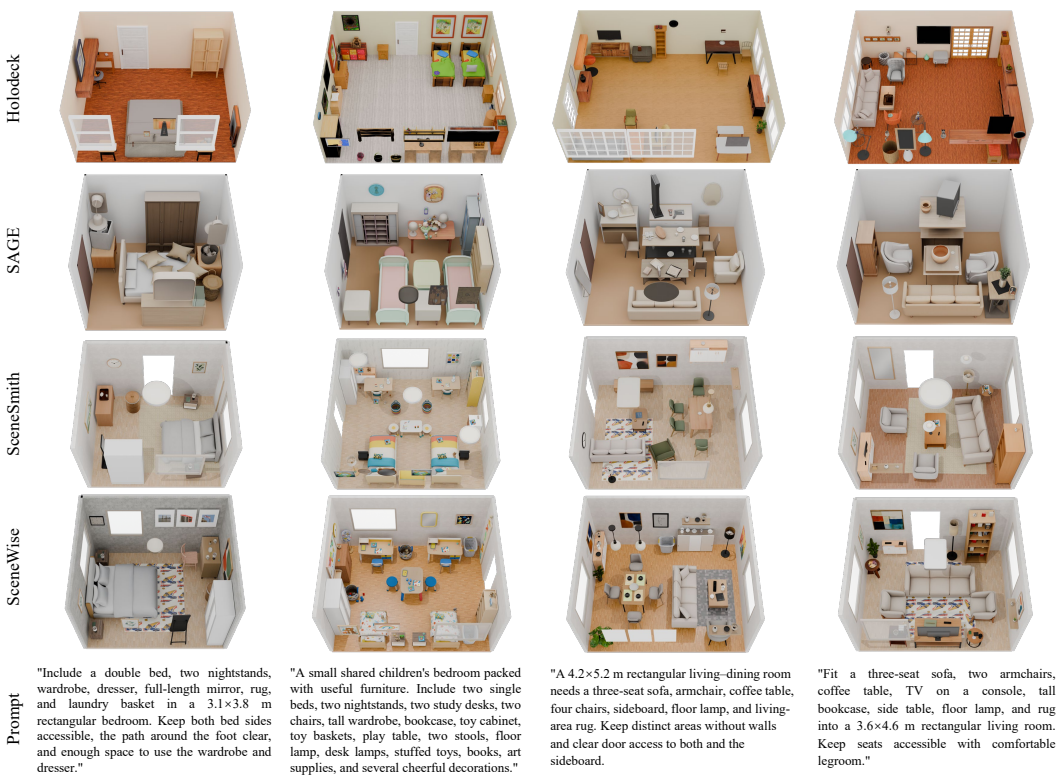


Figure 3: **Qualitative comparison.**. SceneWise better accommodates the requested furniture while organizing distinct functional areas and reserving space for circulation and furniture use, particularly in compact rooms with dense furnishing requirements; Appendix E provides more examples.

SR versus 62.0% for SAGE, while reducing editing operations from 12.8 to 4.8 and distinct edited objects from 5.6 to 3.2. Holodeck edits fewer objects but succeeds much less often, underscoring the need to interpret edits jointly with success. Removing generation guidance or learned attribution reduces SR to 76.0% or 78.0%; removing both lowers it to 68.0%. Removing repair guidance yields 79.3% SR. Each removal also increases editing, supporting the contributions of anticipating downstream conflicts, ranking historical revisions, and evaluating replacements.

Qualitative Comparison. Figure 3 compares methods on identical prompts. SceneWise preserves an open central area in the shared bedroom and separates seating and dining areas in the combined room, illustrating coordinated placement under coupled spatial requirements.

4.3 Consequence Prediction and Lookahead Selection

Table 3 links prediction quality to downstream selection. The full model achieves 0.93 AUROC and 87.9% failure-event F1, versus 0.72 and 61.4% for the prompted backbone. On 200 shared partial scenes with identical candidates, continuation policy, and verification budgets, selection SR increases from 73.3% to 88.3%. Removing history or multi-branch data weakens both prediction and completion, supporting their role in learning useful decision consequences.

4.4 Scene Repair

Table 4 compares repair on 100 shared failed checkpoints with 60 verification calls per attempt. The baselines are latest-decision repair (Latest-dec. repair), visible-conflict repair (Visible-conf. repair), and history search without learned attribution (Hist. search w/o attr.). SceneWise reaches 86.0% SR, compared with 69.3% when the prompted backbone ranks the same historical candidates and 48.0% or 54.0% for latest-decision or visible-conflict repair. These comparisons support learned prioriti-

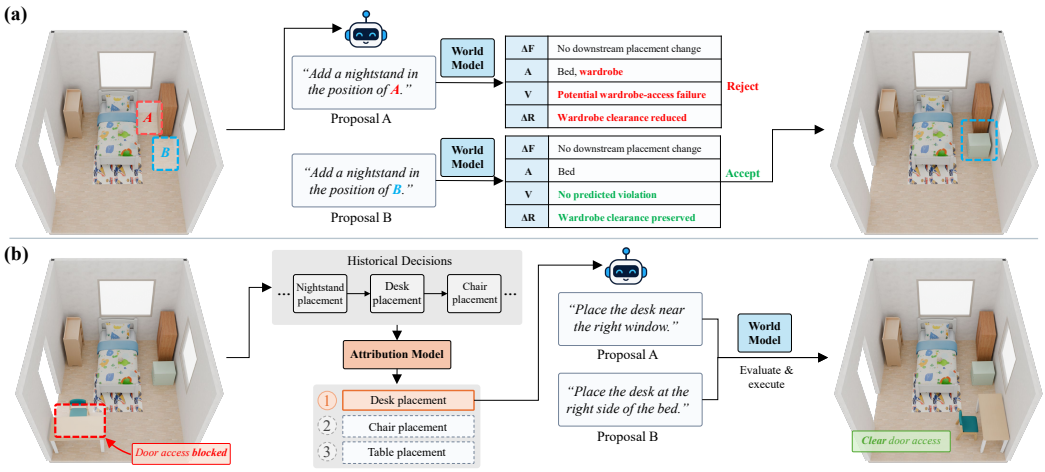


Figure 4: Examples of consequence-guided generation and attribution-guided repair. (a) The world model predicts that nightstand placement A would restrict wardrobe access and selects placement B, which preserves the required clearance. (b) When door access is blocked, the attribution model prioritizes the earlier desk-placement decision for revision. The world model then evaluates alternative placements to guide local repair, restoring door access while preserving the rest of the layout.

zation of earlier revisions. SceneWise also achieves the highest CSR (96.2%) and $\mathrm{SR}_{E\leq 5}$ (78.0%) among the compared methods. It has the lowest mean edit count, verification-call count, and runtime: 2.6 objects, 16.0 calls, and 8.4 minutes, respectively. Figure 4 illustrates consequence-guided placement and historical revision, preserving wardrobe clearance during generation and restoring door access through local repair.

4.5 Repair Success with Limited Edits

We compare repair strategies from shared failed-scene checkpoints using $\mathrm{SR}_{E\leq k}$, the percentage of all attempts that succeed while modifying at most k distinct objects. Figure 5 shows that SceneWise achieves higher success under the same edit threshold and requires fewer modifications to reach comparable success rates, better preserving the existing layout. At $k = 5$, SceneWise achieves 78.0% success, compared with 51.3% for history search without learned attribution.

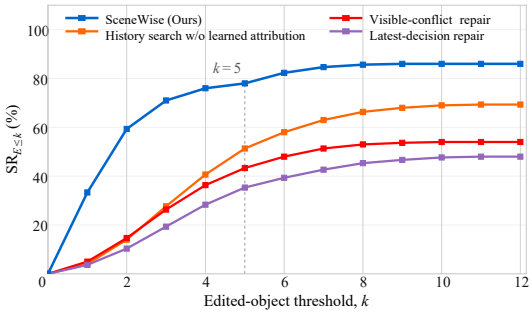


Figure 5: **Repair success with at most k edited objects.** The dashed line marks $k = 5$.

5 Conclusion

We presented SceneWise, a framework that predicts how construction decisions affect later spatial and functional requirements in indoor scenes. The same world model guides candidate selection during generation and evaluates replacements during repair. An attribution model ranks earlier decisions for revision, while execution verifies proposed changes before local repair is accepted. We also introduced SceneWiseDB, providing decision–consequence and matched-intervention records for learning these capabilities. Experiments show improved requirement satisfaction with fewer editing operations and effective repair with limited changes to existing objects. These results highlight the value of anticipating decision consequences for both scene generation and repair, particularly when spatial and functional requirements are closely coupled.

6 AI USE STATEMENT

In this work, we used generative AI tools for research ideation, methodological and experimental design, and method implementation. Additionally, we used generative AI tools for literature discovery, manuscript drafting and editing, and figure design. We reviewed and revised AI-assisted research suggestions and manuscript drafts for consistency with the proposed method and experimental setup. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

REFERENCES

Amir Bar, Gaoyue Zhou, Danny Tran, Trevor Darrell, and Yann LeCun. Navigation world models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15791–15801. IEEE, 2025.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905, 2026.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36:18225–18250, 2023.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Yun He, Kelin Yu, and Matthias Zwicker. Sceneorchestra: Efficient agentic 3d scene synthesis via full tool-call trajectory generation. *arXiv preprint arXiv:2604.19907*, 2026.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. *URL https://arxiv.org/abs/2106.09685*, 2106, 2021.

Chenguo Lin and Yadong Mu. Instructscene: Instruction-driven 3d indoor scene synthesis with semantic graph prior. *arXiv preprint arXiv:2402.04717*, 2024.

Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in neural information processing systems*, 34:12013–12026, 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. Hsm: Hierarchical scene motifs for multi-scale indoor scene generation. *arXiv preprint arXiv:2503.16848*, 2025.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, et al. 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.

Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 29469–29478, 2025.

Kai Sun, Wenqiang Li, Bo Dong, Yuxin Lin, Jingyao Zhang, and Bin Shi. Scope delineation before localization: A two-stage framework for enhancing failure attribution in multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33108–33116, 2026.

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.

Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 20507–20518, 2024.

Yuchen Wang, Jiangtao Kong, Sizhe Wei, Xiaochang Li, Haohong Lin, Hongjue Zhao, Tianyi Zhou, Lu Gan, and Huajie Shao. Westworld: A knowledge-encoded scalable trajectory world model for diverse robotics. In *ICLR 2026 the 2nd Workshop on World Models: Understanding, Modelling and Scaling*, 2026a.

Zhangyi Wang, Jiexiang Xu, Bingnan Yu, and Zongze Li. Architect: Uncertainty-aware dynamic tool learning via causal intervention for open-world agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36819–36829, 2026b.

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.

Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. Physcene: Physically interactable 3d scene synthesis for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16262–16272, 2024a.

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *arXiv preprint arXiv:2509.20414*, 2025.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16227–16237, 2024b.

Xianchao Zeng, Xinyu Zhou, Youcheng Li, Jiayou Shi, Tianle Li, Liangming Chen, Lei Ren, and Yong-Lu Li. Diagnose, correct, and learn from manipulation failures via visual symbols. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 42386–42395, 2026.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. *Advances in neural information processing systems*, 36:31967–31987, 2023.

7 APPENDIX

A IMPLEMENTATION DETAILS

This section supplements the implementation of SceneWise, including the shared model architecture, consequence-guided generation, historical attribution, and local scene repair. All notation follows the main paper.

A.1 MODEL CONFIGURATION AND TRAINING

The consequence model and attribution model share a frozen pretrained language-model backbone with independent LoRA adapters parameterized by ϕ and θ, respectively. Each adapter is inserted into selected backbone layers and activated for its corresponding task. All inputs are serialized as structured text, and outputs use persistent object identifiers and task-specific numerical or label fields.

The consequence adapter is trained on decision–consequence records from the training split of SceneWiseDB. We serialize the task description, partial scene, remaining requirements, decision graph, and candidate decision as input x. The target sequence y contains execution-derived consequences and branch termination status. Training minimizes token-level negative log-likelihood:

$$\mathcal{L}_{\text{con}} = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{con}}^{\text{train}}} \sum_{j} \log p_\phi(y_j \mid x, y_{<j}). \tag{7}$$

Updates are restricted to ϕ.

The attribution adapter is trained on matched-intervention records. Its failure context z contains P, S_T, the generation history, Q_T, G_T, and the failure description. Each historical decision is tested with a fixed budget of proposed revisions under matched replay conditions. Let $\mathcal{J}(z)$ contain the indices of decisions with completed tests, and let $a_i = 1$ if at least one tested revision is effective and $a_i = 0$ otherwise. The adapter predicts $q_\theta(i \mid z)$ through binary label-token probabilities and minimizes

$$\mathcal{L}_{\text{attr}} = -\mathbb{E}_{z\sim\mathcal{D}_{\text{int}}^{\text{train}}} \frac{1}{|\mathcal{J}(z)|} \sum_{i\in\mathcal{J}(z)} \left[a_i \log q_\theta(i \mid z) + (1-a_i)\log(1-q_\theta(i \mid z))\right]. \tag{8}$$

Only θ is updated. Untested decisions and interrupted tests are masked, and contexts without completed tests are excluded. Multiple decisions may be positive. A negative label means that completed tests found no effective revision, rather than that no possible revision could work. These labels describe intervention effectiveness, not satisfaction of every original task requirement.

The two adapters use their corresponding training records, while validation records are used to select scoring weights and thresholds. Local training and inference use at most two NVIDIA A6000 GPUs. The continuation policy and horizon used for consequence supervision match those used for inference-time verification.

A.2 CONSEQUENCE PREDICTION AND LOOKAHEAD GENERATION

Decision context and prediction targets. At generation step t, the world model receives (S_t, Q_t, G_t, P, d), where d is a candidate decision and the remaining inputs describe its generation context. The predicted consequence tuple is $\widehat{\mathcal{C}}_t^H(d) = (\widehat{\Delta F}, \widehat{\Delta R}, \widehat{V}, \widehat{A})$. The prediction concerns candidate execution and continuation under policy π for at most H decisions, including the candidate itself. Thus, H specifies the extent of the evaluated continuation. For supervised records, alternative candidates are executed in independent copies of the partial scene, and validators Ψ extract their consequences:

$$\tau_t^H(d) = \text{Rollout}_H^\pi(S_t, d, Q_t), \qquad \mathcal{C}_t^H(d) = \Psi(\tau_t^H(d), S_t, Q_t). \tag{9}$$

Immediate geometric violations are filtered directly; the targets describe the consequences of candidate execution and subsequent generation.

Table 5: Training and inference hyperparameters of SceneWise.

Parameter	**Value**
Backbone	Qwen3.8-27B
Hardware	$2\times$ RTX A6000 (48 GB each)
Fine-tuning	4-bit QLoRA
LoRA rank / alpha / dropout	32 / 64 / 0.05
Optimizer / learning rate	AdamW / 1×10^{-4}
Effective batch size	16
Maximum sequence length	4096
Maximum training steps (world model / attribution)	5000 / 2000
Lookahead $K/B/H$	5 / 2 / 3
Maximum additional retries	2
Risk weights (α, β, γ)	$(0.25, 0.25, 0.50)$
Repair cost weights $(\lambda_I, \lambda_B, \lambda_D)$	$(1.0, 0.5, 0.1)$
Intervention threshold / repair tolerance	10% relative loss reduction / $\epsilon = 0$
Execution budget per repair	60

Placement feasibility. For each pending object o, placement feasibility is measured using the fraction of valid poses in its candidate set Ω_o, as defined in Equation 2. Branches from the same partial scene share pending objects, pose samples, and validation rules. If an object is placed during a branch, its feasibility is measured immediately before placement. Otherwise, it is measured in the last valid branch state. This convention avoids counting the object's own occupancy as a reduction in its available placement space.

Constraint margins and failure events. Relational and functional margins retain the sign convention of the main paper: positive values indicate satisfaction and negative values indicate violation. Margin changes are compared over constraints that are measurable in both states, while unmeasurable entries are masked. Violations of newly measurable constraints contribute to the failure-event component. Recorded failures include collisions, entrance obstruction, insufficient clearance, relational violations, and inability to continue generation. Affected objects are identified from feasibility changes, margin changes, and participation in failures.

Candidate ranking. The Designer proposes K candidate decisions from the current task context. Immediate collision, boundary, and hard-constraint checks produce the valid candidate set U_t^{valid}. The world model evaluates its members using the risk function in Equation 3. The feasibility term averages reductions in feasible placement fractions, the margin term averages decreases over comparable constraints, and the failure term assigns weighted penalties to predicted violations. Empty averages are defined as zero. Scoring weights and thresholds are selected using validation data.

Execution-based selection. SceneWise retains up to B candidates with the lowest predicted risks and verifies each through a rollout of at most H decisions. Branches that violate hard constraints or cannot continue generation are excluded from the final selection. Among the remaining branches, the candidate with the lowest execution-measured risk is selected. Both completed branches and failure-free branches that reach the horizon are admissible, but horizon termination alone does not establish task completion.

Only the first decision of the selected branch is committed. The remaining rollout decisions provide evidence for selection and are not directly committed to the scene. SceneWise then updates S_t, Q_t, and G_t and generates new proposals from the updated state. Each selection round therefore verifies at most B branches of length H.

Historical diagnosis is invoked when bounded proposal retries yield no immediately valid candidate, all shortlisted branches fail, or post-execution checks detect a violation. Generation terminates successfully only after the complete task requirements have been satisfied and the final scene passes validation.

A.3 Historical Attribution and Repair

Historical candidate retrieval. Given a failed checkpoint S_T, SceneWise retrieves decisions associated with the failed objects, regions, and constraints, together with their ancestors in G_T. The attribution context includes the task description, failed scene, generation history, remaining requirements, decision graph, and failure description. The attribution adapter scores these candidates to determine the order of intervention testing.

The attribution score prioritizes decisions for revision. Its training label records whether an effective replacement was found among the completed tests for that decision. Multiple historical decisions can receive positive labels for the same failure context. A negative label means that the completed tests did not identify an effective replacement within the prescribed testing configuration; it does not establish that every possible replacement would be ineffective.

Consequence-guided replacements. For each prioritized decision, the Designer proposes replacements. After removing content designated for regeneration, SceneWise constructs the repair context $(S^{\mathrm{rep}}, Q^{\mathrm{rep}}, G^{\mathrm{rep}})$. These components describe the retained scene, outstanding and regeneration requirements, and updated decision graph. The same consequence adapter evaluates each immediately valid replacement $\widetilde{d}$:

$$\widehat{\mathcal{C}}^{H}_{\mathrm{rep}}(\widetilde{d}) = W_{\phi}(S^{\mathrm{rep}}, Q^{\mathrm{rep}}, G^{\mathrm{rep}}, P, \widetilde{d}). \tag{10}$$

The input schema and output meanings remain consistent with forward generation; only the decision context changes.

Predicted risks shortlist replacements for execution-based testing. Subsequent local regeneration follows the lookahead procedure in Section A.2, updating the repair context after each committed local decision.

Matched intervention verification. For a historical decision d_i, the reference branch retains d_i, while the intervention branch applies $\widetilde{d}_i$. The two branches regenerate affected content using the same continuation procedure, task requirements, regeneration scope, search budget, validators, and controllable randomness. Content outside the scope is held identical. The intervention effect is measured by

$$\Delta L_i(\widetilde{d}_i) = L(S^{\mathrm{ref}}_{T,i}) - L(S^{\mathrm{int}}_{T,i}(\widetilde{d}_i)). \tag{11}$$

A replacement is effective when the reference branch still exhibits the target failure and the intervention eliminates that failure or reduces its loss beyond the prescribed threshold. Improvement produced by reference replay alone is not attributed to the replacement. The original trajectory is used as the reference only when it satisfies the same comparison conditions.

Adaptive regeneration scope. Let I contain the indices of historical decisions to revise. The initial regeneration scope contains these indices and the indices of their descendants in the decision graph. Boundary checks identify conflicts between regenerated and retained content and expand the scope to include conflicting decisions and necessary successors. If either comparison branch requires expansion, both branches are replayed within the common expanded scope. Repeated execution contributes to the intervention-testing budget.

Repair commitment. Search first tests ranked single-decision replacements and considers joint interventions when necessary. The repair objective accounts for the number of direct interventions, the number of regenerated decisions, and changes to the existing scene, including pose changes, additions, removals, and replacements. SceneWise commits the lowest-cost verified feasible repair found within the budget.

For incomplete scenes, repair validation checks currently evaluable constraints, while outstanding requirements remain active for subsequent generation. An effective intervention can improve the target failure without completing the entire task; final success still requires satisfaction of the complete original specification. If no feasible repair is found, SceneWise reports failure without committing unverified changes.

A.4 Prompts and Input–Output Examples

SceneWise uses the original task description throughout generation and repair. The remaining requirement list changes as construction progresses, while the original specification remains the reference for final evaluation. The Designer, consequence model, and attribution model receive different structured views of this shared task context.

Template scope. The templates below are used by the Designer, consequence model, and attribution model in our implementation. Input fields are populated with serialized task and scene records. The examples in this section and Section B.3 report recorded model outputs and execution feedback.

Candidate proposal. The Designer receives the task, current scene, and remaining requirements and proposes object choices or scene modifications. A decision $d = (a, u, \boldsymbol{\eta}, \mathcal{R})$ specifies the operation type, target object identifier, operation-specific parameters, and relational constraints, following Section 3.1. During repair, the context also identifies the historical decision being revised and the retained scene. A compact instruction template is:

```
Given the original task, current scene, and remaining
requirements, propose K candidate scene decisions.
Use the operation, target ID, parameters, and relations
schema. Permitted operations are transform, resize,
add, remove, replace, and modify. Preserve object IDs and
include all parameters needed by the proposed operation.
For repair, also use the supplied revision target and scope.
Input: {P, S, Q, G, K, optional_repair_context}
Output: {candidate_decisions}
```

Geometric validity and downstream feasibility are checked by the subsequent filtering and verification stages; a proposal alone does not establish validity.

Consequence prediction. A consequence-model input contains (S_t, Q_t, G_t, P, d) and the continuation configuration. Scene records specify object and asset identifiers, geometry and poses, room boundaries, and applicable constraints; G_t preserves the history and its dependencies. The continuation policy and horizon are fixed consistently with training and verification. Execution-derived outcomes from the candidate branch are prediction targets, not input fields. The instruction template is:

```
Predict the consequences of candidate d and continuation
under policy pi for at most H decisions, including d.
Return delta_F, delta_R, V, A, and termination_status.
delta_F: feasibility changes keyed by pending-object ID;
evaluate an object just before placement, or at the last
valid state if it remains unplaced.
delta_R: normalized margin changes for constraints that
are measurable both before and after continuation.
V: predicted failure events with object and constraint IDs.
A: IDs of objects substantially affected or in failures;
this is not a list of established causes.
Distinguish completion, horizon termination, and failure.
Use the supplied schema, masks, and persistent IDs.
Input: {P, S, Q, G, d, continuation_configuration}
```

The first four output fields represent the consequence tuple; termination status is supplementary metadata. Unmeasurable constraints are masked rather than assigned a zero margin change. Violations of newly measurable constraints can still appear in V.

Input context. The output below uses a compact bedroom example. The original task P requests a bed and a wardrobe while preserving entrance access. The partial scene S_t contains the room boundary and `door_01`, but neither furniture object has been committed. The pending objects are $\mathcal{O}(Q_t) = \{\texttt{bed_01}, \texttt{wardrobe_01}\}$; G_t has no committed furniture decisions. Candidate d_A adds `bed_01` using a specified bed asset and pose A. Its relation requirements preserve the entrance

and allow subsequent wardrobe placement. Pose A is immediately valid but may restrict the later wardrobe. An alternative d_B uses the same asset at pose B. Both candidates use the same initial scene, pending objects, pose samples, continuation policy, horizon, and validation rules. Asset and pose records are abbreviated here; these identifiers stand for complete geometry and transformation records in the input serialization.

Recorded model output. The following JSON is a recorded structured response from the consequence model for candidate d_A, which adds `bed_01` while `wardrobe_01` remains to be placed. Under the fixed continuation policy and horizon, the model predicts reduced wardrobe-placement feasibility and an entrance obstruction during subsequent construction.

```
{
  "delta_F": {"bed_01": 0.00, "wardrobe_01": -0.25},
  "delta_R": {"door_01.passage_clearance": -0.30},
  "V": [
    {
      "type": "entrance_obstruction",
      "object_ids": ["bed_01", "wardrobe_01"],
      "constraint_id": "door_01.passage_clearance"
    }
  ],
  "A": ["bed_01", "wardrobe_01"],
  "termination_status": "failure"
}
```

The first four fields correspond to $\widehat{\Delta F}$, $\widehat{\Delta R}$, $\widehat{V}$, and $\widehat{A}$, respectively. The predicted wardrobe feasibility change of -0.25 corresponds to a valid-pose fraction decreasing from 0.60 to 0.35. The bed entry is zero because its feasibility is evaluated immediately before the root candidate executes, in the same state as the baseline. The predicted normalized entrance-margin change of -0.30 corresponds to a margin decreasing from $+0.20$ to -0.10; this value is not a distance in meters. The failure event identifies the predicted obstruction, and the affected-object set records changed feasibility or participation in the violation, rather than a causal attribution. The termination field reports predicted failure within the continuation horizon. These predictions guide candidate selection; execution-based verification remains necessary before commitment.

Historical attribution. An attribution input contains the failure context z and a historical decision under consideration. The context includes the original task, failed scene, decision history, pending requirements, decision graph, and observed failure. Its instruction template is:

```
Given failure context z and historical decision i, predict
whether an effective replacement can be found under the
prescribed intervention-testing configuration.
Use the binary effectiveness labels defined by training.
A positive label requires a completed matched test in
which retaining i still fails and replacing i removes
the failure or reduces its loss beyond the threshold.
A negative label describes completed unsuccessful tests;
it does not prove that every replacement is ineffective.
Input: {P, failed_scene, history, Q, G, failure, decision_i}
Output: {effectiveness_label}
```

The probability assigned to the positive label supplies $q_\theta(i \mid z)$ for ranking decisions; the score is read from label-token probabilities rather than requested as a freely generated numerical justification. Actual reference and intervention outcomes are used to construct training labels and to verify proposed repairs, not supplied as future evidence during attribution. Untested decisions and interrupted tests are masked during training. A high score establishes testing priority only; commitment still requires execution-based verification.

B SceneWiseDB Details

B.1 Data Construction and Annotation

SceneWiseDB provides supervision for consequence prediction and historical attribution through decision–consequence records $\mathcal{D}_{\text{con}}$ and matched-intervention records $\mathcal{D}_{\text{int}}$. The records connect decisions to executed continuations and historical revisions to verified repair effects.

Decision–consequence records. We sample partial scenes from offline generation trajectories and execute alternative candidates in independent scene copies. Each branch continues under policy π for at most H decisions, including the candidate. Validators extract the measured consequence tuple $(\Delta F, \Delta R, V, A)$ and termination status. Branches originating from the same partial scene share the pending objects, candidate pose samples, and validation rules used to assess their consequences.

The records include successful and failed continuations, with particular attention to immediately valid decisions that introduce delayed failures. Each record retains the decision context, candidate, continuation, structured consequences, and termination status. A branch reaching the horizon is distinguished from a branch completing the original task.

Matched-intervention records. Intervention records start from failed checkpoints and compare historical replacements against matched reference replays. Both branches use the same continuation configuration, regeneration scope, task requirements, search budget, validators, and controllable randomness. The records retain the tested replacement, outcomes of both branches, violation-loss reduction, regeneration scope, scene-change cost, and global validity outcome.

Completed tests are retained whether or not the replacement is effective. Historical-decision labels are aggregated over the completed replacement tests: a decision is positive when at least one replacement is effective and negative when none of the completed tests is effective. Untested decisions and interrupted tests without completed outcomes are masked. This labeling supports learning from both effective and ineffective interventions without treating unobserved outcomes as failures.

Replay metadata. Replay information includes recoverable scene states, persistent object identifiers, asset references, random seeds, and execution settings. These fields connect each supervision target to the scene and comparison conditions that produced it. When a regeneration scope expands, the recorded comparison reflects the common expanded scope and the repeated execution required for both branches.

B.2 Dataset Statistics and Splits

SceneWiseDB contains 100,000 samples spanning 20 room categories and six action types: `transform`, `resize`, `add`, `remove`, `replace`, and `modify`. Its two record types support consequence prediction and historical decision attribution.

Operation definitions. Transform changes position or orientation; resize changes size or shape; add and remove insert or delete objects; and replace exchanges an object's asset. Modify changes material, color, physical properties, or articulation and fixation states, such as opening a door or drawer. All six operations are permitted during generation and repair and use the decision schema in Section 3.1. Offline consequence branches and intervention replays record the same operations. Generation uses them to construct and revise a partial scene; repair uses them for historical revisions and local regeneration.

Record definition. One decision–consequence record consists of a partial-scene context, a candidate decision, and the consequences measured from its executed continuation. Different candidates evaluated from the same partial scene count as separate records. One matched-intervention record consists of a failed-scene context, a historical decision selected for revision, a proposed replacement, and the paired outcomes of retaining and replacing that decision under matched replay conditions. These two branches jointly count as one intervention record. Different tested replacements count separately, even when they share a reference replay. Intermediate rollout states and individual annotation fields do not count as additional records.

Composition and splits. The dataset comprises 70,000 decision–consequence records and 30,000 matched-intervention records derived from 5,000 source-scene groups. Each group contains one source scene and its closely related variants. The training, validation, and test partitions contain 80,000, 10,000, and 10,000 records, respectively, as summarized in Table 6. Record counts refer to supervision records rather than unique simulator executions.

Table 6: Composition and partitioning of SceneWiseDB. Each matched reference–intervention pair counts as one intervention record.

Split	Source-scene groups	Consequence	Intervention	Total
Training	4,000	56,000	24,000	80,000
Validation	500	7,000	3,000	10,000
Test	500	7,000	3,000	10,000
Total	5,000	70,000	30,000	100,000

Action distribution. Table 7 gives the distribution across the six action types. Each record receives one action label: the evaluated candidate action for a consequence record, or the tested repair action for an intervention record. Additional actions executed during continuation or replay do not contribute separate counts to this distribution.

Table 7: Action distribution in SceneWiseDB. Percentages are computed over all 100,000 records.

Action type	Records	Proportion (%)
`transform`	28,000	28.0
`resize`	16,000	16.0
`add`	20,000	20.0
`remove`	12,000	12.0
`replace`	14,000	14.0
`modify`	10,000	10.0
Total	100,000	100.0

Partitioning protocol. Source-scene groups are partitioned before derived trajectories and interventions are collected. All prefixes, candidate branches, failed checkpoints, and complete intervention pairs from the same group remain in the same partition. Training records supervise the two adapters, while validation records are used to select thresholds and scoring weights. Test records are reserved for held-out evaluation.

The generalization settings exclude designated layout families, asset shape subtypes, or requirement combinations from both training and validation records. These exclusions define the conditions under which the learned generation and repair guidance is evaluated. External baselines retain their original generation pipelines; the designation "unseen" refers specifically to the SceneWiseDB partition.

B.3 Annotated Examples

The examples below use recorded model outputs and execution feedback to connect the input interfaces in Section A.4 to consequence labels and intervention outcomes. They show how predictions guide candidate selection and how execution verifies decisions and repairs.

Decision–consequence trace. This run uses the bedroom context in Section A.4. Both candidate bed placements d_A and d_B pass immediate checks. Before either branch, the wardrobe feasible-pose fraction is 0.60 and the normalized entrance margin is $+0.20$. The same 50 sampled wardrobe poses are used for both branches, with 30 initially valid poses. The pose counts reported here refer to this run.

For d_A, the recorded output in Section A.4 predicts a wardrobe feasibility change of -0.25, an entrance-margin change of -0.30, an entrance obstruction, and the affected set $\{\texttt{bed_01}, \texttt{wardrobe_01}\}$. For d_B, the model returns the same four fields as follows:

```
{
  "delta_F": {"bed_01": 0.00, "wardrobe_01": 0.00},
  "delta_R": {"door_01.passage_clearance": 0.00},
  "V": [],
  "A": [],
  "termination_status": "horizon"
}
```


The empty affected-object set predicts no substantial change to the monitored quantities and no failure participation; it does not mean that adding the bed leaves the scene geometry unchanged. Both bed-feasibility changes are zero because each root candidate's own feasibility is evaluated before that candidate executes.

Both candidates are executed within the verification budget. Table 8 compares their model predictions with measured execution outcomes. In branch A, 15 of the original 50 wardrobe poses remain valid at its evaluation state, giving $0.30 - 0.60 = -0.30$, and the normalized entrance margin changes to -0.15, giving $-0.15 - 0.20 = -0.35$. The branch reports entrance obstruction. In branch B, the corresponding values remain 0.60 and $+0.20$, and no failure is detected before the horizon, with the wardrobe still uncommitted. If the wardrobe is placed in either branch, its feasible fraction is measured immediately before that placement, following Section A.2.

Table 8: **Recorded predictions and execution feedback.** The model predicts an obstruction for d_A and no failure within the horizon for d_B; execution confirms these outcomes. The numeric columns report predicted and measured changes in wardrobe-placement feasibility and normalized entrance margin.

Candidate	Stage	ΔF	ΔR	Event / termination
d_A	Prediction	-0.25	-0.30	Obstruction / failure
d_A	Execution feedback	-0.30	-0.35	Obstruction / failure
d_B	Prediction	0.00	0.00	None / horizon
d_B	Execution feedback	0.00	0.00	None / horizon

Execution feedback for branch A records the obstruction in V and the bed and wardrobe in A; branch B has empty V and A for this example. Thus, the model output and execution-derived target contain the same four consequence components while remaining distinct records. The failed branch A is excluded. Branch B is admissible, so only its initial bed-placement decision is committed; its simulated continuation is not copied into the scene. Horizon termination indicates that no failure was detected within the tested continuation, not that all original requirements have been satisfied. Generation resumes from the committed state.

Matched-intervention trace. In a separate recorded repair case, an earlier desk placement restricts door access. The attribution input identifies the failure context and the historical desk-placement decision. Its positive-label probability determines when that decision is tested; the score alone is not evidence that the desk is the cause. The Designer proposes a revised desk placement, and the consequence model evaluates the replacement in its repair context.

The intervention record stores the decision identifier, original and replacement parameters, common regeneration scope, continuation settings, random seed, verification budget, and both replay outcomes. The reference replay retains the desk placement; the intervention replay changes it. The reference replay still exhibits the target obstruction, while the intervention eliminates it and passes the applicable consistency checks. This completed test supplies a positive effectiveness label for that historical decision. Any scope expansion is applied to both branches before comparison, and all repeated execution counts toward the budget. If both branches recover through replay alone, the test does not establish replacement effectiveness; interrupted tests remain unlabeled.

A verified replacement may then be committed if it is selected by the budgeted repair search. Removing this obstruction does not establish full task success unless the complete original requirement checklist also passes.

C Experimental Setup and Evaluation

C.1 Evaluation Tasks and Requirement Checklists

The evaluation covers scene generation, consequence prediction, lookahead selection, scene repair, and generalization. General scene quality is evaluated on 94 SceneEval room prompts. An additional 100 tasks assess generation under coupled spatial and functional requirements. Consequence prediction uses held-out execution records from SceneWiseDB, while lookahead selection starts from 200 shared partial scenes.

The repair evaluation uses 100 shared failed-scene checkpoints and allocates 60 execution-verification calls to each repair attempt. Generalization is evaluated on three sets of 60 tasks covering unseen layouts, asset shapes, and requirement combinations. Each task is evaluated with three random seeds.

Table 9: Evaluation tasks and controlled conditions.

Evaluation	Task set	Controlled conditions
General scene quality	94 room prompts	Shared evaluation protocol
Coupled generation	100 tasks	Common task inputs, evaluator, and call budget
Consequence prediction	Held-out execution records	Execution-derived labels
Lookahead selection	200 partial scenes	Identical candidate sets, continuation policy, and execution budget
Scene repair	100 failed checkpoints	Common starting states and 60 verification calls per attempt
Generalization	Three sets of 60 tasks	Fixed model parameters and scoring settings

Each coupled task has a fixed requirement checklist derived from its original specification and manually verified before generation. The checklist includes the requested objects and their spatial and functional requirements, such as preserving circulation routes, maintaining furniture access, allowing cabinet operation, and placing objects on appropriate supporting surfaces. All methods receive the same task inputs and are evaluated against the complete original checklist.

Final success includes requirements satisfied before candidate selection or repair. A repair that resolves the reported conflict while violating another original requirement is unsuccessful. Attempts that exhaust the evaluation budget without satisfying the complete checklist remain in the task-level results and cost averages.

C.2 Baseline and Ablation Configurations

External generation baselines. We compare with six external scene-generation methods. Each retains its own generation pipeline and uses the same task inputs and evaluator. Coupled-task comparisons also share the execution-verification interface and call budget. The shared-backbone setting applies to controlled SceneWise ablations.

HSM Pun et al. (2025) generates scenes hierarchically, using recurring spatial motifs to organize both room-level furniture and smaller objects on supporting surfaces.

Holodeck Yang et al. (2024b) uses an LLM to select objects and specify spatial relations, then optimizes object placement to satisfy these constraints.

SceneWeaver Yang et al. (2025) uses an LLM planner to select scene-generation tools and iteratively refine scenes based on physical, visual, and semantic feedback.

SceneOrchestra He et al. (2026) generates complete tool-call trajectories with a learned orchestrator, using discriminator feedback during training to improve trajectory quality.

SAGE Xia et al. (2026) combines layout and object generation with semantic, visual, and physical critics to iteratively build scenes from task instructions.

SceneSmith Pfaff et al. (2026) coordinates Designer, Critic, and Orchestrator agents across architectural layout, furniture placement, and small-object placement, integrating asset generation and physical validation.

System-level component ablations. Controlled SceneWise ablations share the backbone, proposal generator, and tools. Table 10 states the scope of each removal and the functionality that remains. All variants retain the original requirement checklist and execution-based validity checks. The variant without learned attribution still ranks the same historical decision candidates using the prompted backbone; it does not remove historical reasoning or restrict repair to the latest decision. Removing forward consequence guidance is distinct from removing its use during repair.

Table 10: Scope of the controlled SceneWise component ablations. Designer ranking replaces forward consequence guidance, and prompted-backbone ranking replaces learned attribution, as detailed below.

Variant	Removed behavior	Retained behavior
w/o consequence-guided generation	Learned consequence ranking during forward generation	Designer ranking; historical attribution and consequence-guided repair
w/o learned attribution	Learned ranking of historical intervention targets	Prompted-backbone historical ranking; consequence-guided generation and repair
w/o generation guidance and attribution	Both forward consequence ranking and learned historical ranking	Designer ranking and prompted-backbone attribution; consequence-guided repair
w/o consequence-guided repair	Learned consequence ranking of replacement and local-regeneration candidates	Forward consequence guidance and historical attribution

Without consequence-guided generation. The same Designer proposes candidate decisions, and the same collision, boundary, and applicable hard-constraint checks filter them. The world model is not called to predict downstream consequences for forward candidate ranking. Instead, the Designer ranks valid candidates from the current scene and task requirements. The same number of shortlisted candidates is passed to execution-based verification, with the verification budget unchanged. Historical attribution and consequence-guided repair remain active.

Without learned attribution. The historical decision candidate set is unchanged. The trained attribution LoRA is not loaded; the original pretrained backbone is prompted with the failure information and decision history to rank which decisions should be revised first. Replacement-candidate generation, world-model evaluation, and repair verification remain unchanged. This comparison tests the learned attribution adapter relative to prompted attribution by the backbone. The joint variant combines this prompted-backbone ranking with Designer-based forward candidate ranking.

Consequence-prediction ablations. The prompted backbone, the variant without decision history, the variant without multi-branch data, and the full consequence model use the same consequence-output definitions and held-out evaluation records. Their lookahead-selection comparison uses identical candidate sets, continuation policy, and execution budget, so a change in predictor does not redefine the execution target.

The prompted-backbone comparison removes task-specific consequence-adapter training and elicits the structured output from the pretrained backbone. The no-history comparison removes decision-history information from the predictor input while retaining the task, current scene, pending requirements, and candidate decision.

Without multi-branch data. The full consequence-training set contains multiple alternative candidate decisions and their respective executed consequences from the same partial-scene context. The *w/o multi-branch data* variant retains only one decision–consequence record per partial scene and supplements the set with records from other partial scenes in the training split, each also contributing a single candidate. The total number of consequence-training records, batch size, and

training steps match the full model. Supplementary records respect the source-scene partitioning in Section B.2; no validation or test records are used.

Only the composition of consequence-training data changes. The sequence-likelihood objective in Equation 7 remains unchanged, and the attribution model's matched-intervention training data are unchanged. The comparison contrasts multiple candidate outcomes per context with single-branch records from more contexts at a fixed training size; it does not remove a pairwise training loss or hold the number of distinct partial-scene contexts fixed.

Historical-repair baselines. The repair comparison in Table 4 includes latest-decision repair, visible-conflict repair, historical search without learned attribution, and SceneWise. Latest-decision repair targets the most recent decision, whereas visible-conflict repair targets objects implicated in the observed violation. Historical search without learned attribution uses the same historical candidate set and the prompted pretrained backbone to rank revisions, with the attribution LoRA disabled as described above. All strategies begin evaluation from the same failed-scene checkpoints and are assessed against the complete original task requirements under the stated call budget. The prompted-backbone ordering replaces learned attribution only; downstream replacement proposals, world-model evaluation, and execution verification are retained.

Alternative methods may propose repairs absent from the intervention dataset; those proposals are executed and assessed using the same validators. Reference replays, proposed replacements, and repeated execution caused by scope expansion are included in the repair-testing budget. Failed attempts remain included in success rates and cost averages, as specified in Section C.3.

C.3 Task-Specific Evaluation Metrics

Let N denote the total number of evaluation attempts, including all random seeds and unsuccessful attempts. For attempt i, let $\mathcal{Q}_i = \{q_{i1}, \ldots, q_{iM_i}\}$ be the complete original requirement checklist, and let $c_{ij} \in \{0, 1\}$ indicate whether requirement q_{ij} is satisfied in the evaluated scene. Requirements that remain unmet or cannot be verified as satisfied receive $c_{ij} = 0$. The checklist remains fixed throughout generation and repair.

Success rate (SR). An attempt succeeds only when every original requirement is satisfied. We define its success indicator s_i and the overall success rate as

$$s_i = \prod_{j=1}^{M_i} c_{ij}, \qquad \mathrm{SR} = \frac{100}{N} \sum_{i=1}^{N} s_i. \tag{12}$$

For repair, the checklist includes requirements already satisfied at the failed-scene checkpoint, so resolving the target failure while violating another original requirement does not count as success.

Constraint satisfaction rate (CSR). CSR measures the average fraction of satisfied requirements:

$$\mathrm{CSR} = \frac{100}{N} \sum_{i=1}^{N} \left(\frac{1}{M_i} \sum_{j=1}^{M_i} c_{ij} \right). \tag{13}$$

Requirements are first averaged within each attempt, and the resulting fractions are then averaged across attempts. This gives each attempt equal weight regardless of its checklist length. Unsuccessful attempts can still receive nonzero CSR.

Object-edit operations (EditOps). Let $\mathcal{A}_i$ denote the sequence of object-level modification events recorded in attempt i, including moves, rotations, rescaling, reshaping, replacements, removals, and other object modifications after initial placement. Then

$$\mathrm{EditOps} = \frac{1}{N} \sum_{i=1}^{N} |\mathcal{A}_i|. \tag{14}$$

Repeated modifications of the same object count separately. An operation modifying multiple objects contributes one object-level event per modified object. Initial placement is excluded, and modifications that are subsequently undone remain part of the editing history.

Distinct edited objects (Edits). Let $\mathrm{objid}(e)$ denote the persistent object identity associated with modification event e, and define

$$\mathcal{U}_i = \{\mathrm{objid}(e) : e \in \mathcal{A}_i\}. \tag{15}$$

For generation, $E_i = |\mathcal{U}_i|$. For repair, let $\mathcal{O}_i^0$ be the set of objects present at the failed-scene checkpoint; only modification events after this checkpoint are considered, and $E_i = |\mathcal{U}_i \cap \mathcal{O}_i^0|$. The reported metric is

$$\mathrm{Edits} = \frac{1}{N}\sum_{i=1}^{N} E_i. \tag{16}$$

Each object is counted once regardless of how often it is modified. A replacement is associated with the original object's counting identity, preventing repeated replacements from inflating the distinct-object count.

Success rate at an edited-object threshold ($\mathrm{SR}_{E\le k}$). For repair, we jointly measure complete requirement satisfaction and the number of checkpoint objects modified:

$$\mathrm{SR}_{E\le k} = \frac{100}{N}\sum_{i=1}^{N} s_i\,\mathbf{1}[E_i \le k]. \tag{17}$$

The denominator includes all repair attempts, rather than only successful attempts or attempts satisfying the edited-object threshold. Table 4 uses $k = 5$. We compute the repair-success curve in Figure 5 by varying k using the same attempt outcomes. This threshold concerns objects present at the checkpoint and does not impose a bound on newly added objects.

Execution-verification calls (Calls). Let C_i be the number of calls issued through the common execution-verification interface during attempt i. We report

$$\mathrm{Calls} = \frac{1}{N}\sum_{i=1}^{N} C_i. \tag{18}$$

Repeated calls, failed calls, and calls used to verify candidate branches are included. World-model predictions are excluded because they do not invoke execution verification. Reference replays, intervention branches, and repeated execution caused by scope expansion contribute to the repair-testing budget.

Elapsed time (Time). Let t_i^{start} and t_i^{end} be the wall-clock timestamps, measured in seconds, at the start and termination of attempt i. Average runtime in minutes is

$$\mathrm{Time} = \frac{1}{60N}\sum_{i=1}^{N}\left(t_i^{\mathrm{end}} - t_i^{\mathrm{start}}\right). \tag{19}$$

Runtime includes model inference, world-model predictions, asset processing, execution, and verification. For repair, timing starts from the shared failed-scene checkpoint. Wall-clock duration is measured directly, avoiding double counting when processing stages overlap. All editing and cost metrics are averaged over successful and unsuccessful attempts.

C.4 Other Metrics, Validators, and Statistical Reporting

Consequence prediction. AUROC and failure-event F1 assess predictions on held-out labeled execution outcomes. AUROC is reported on a 0–1 scale, and F1 is reported as a percentage. Task completion, failure-free horizon termination, and execution failure remain distinct, so horizon termination is not interpreted as evidence that all later requirements will be satisfied.

General scene quality. The scene-quality evaluation reports count satisfaction (CNT), attributes (ATR), object–object relations (OOR), object–architecture relations (OAR), accessibility (ACC), free-space connectivity (NAV), and out-of-bounds objects (OOB). Following the SceneSmith evaluation protocol Pfaff et al. (2026), it additionally reports collision rate (COL), stability (STB), and average object count (#Obj). Object count describes scene density and is not ranked as a quality objective.

Validation rules. Immediate checks assess collisions, room boundaries, and applicable hard constraints. Continuation validation additionally detects entrance obstruction, insufficient clearance, relational violations, and inability to continue generation. Relational and functional margins provide signed measures of constraint satisfaction. Constraints that cannot yet be measured are masked during consequence comparison and remain subject to evaluation when the necessary scene content becomes available.

Repair validation checks the regenerated content and its consistency with the retained scene. For incomplete scenes, currently evaluable constraints are checked while unmet requirements remain active. Final validation assesses the complete original task specification.

Statistical reporting. Each task is evaluated with three random seeds. Unless otherwise specified, results are reported as means with 95% confidence intervals. Comparisons based on shared partial scenes or failed checkpoints use the same starting conditions. Generalization evaluations keep model parameters and scoring settings fixed across held-out tasks.

D Additional Quantitative Results

D.1 Complete Scene-Quality Results

Table 11 provides the complete version of Table 1 on the same 94 SceneEval room prompts. In addition to the six metrics shown in the main paper, it includes mean object count (#Obj), collision rate (COL), stability (STB), and out-of-bounds rate (OOB).

Table 11: **Complete scene-quality results on room-level prompts.** SceneWise scores highest on OOR and OAR. Scores are percentages except #Obj (mean object count, not ranked).

Method	#Obj	CNT↑	ATR↑	OOR↑	OAR↑	ACC↑	NAV↑	COL↓	STB↑	OOB↓
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**
SceneOrchestra	16.0±1.5	50.0±4.0	40.0±6.5	25.0±5.0	42.0±6.0	82.0±2.0	98.5±0.8	10.0±2.0	45.0±4.0	0.1±0.1
SAGE	50.0±6.0	80.0±3.5	72.0±6.0	62.0±5.8	75.0±5.5	86.0±1.8	98.0±0.9	2.0±0.7	95.0±1.8	0.2±0.2
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
SceneWise (Ours)	70.0±12.0	**83.0±3.3**	74.0±6.0	**74.0±5.5**	**82.0±5.2**	88.0±1.5	98.5±0.8	1.5±0.6	95.0±1.8	0.2±0.2

SceneWise and SceneSmith produce similar mean object counts (70.0 and 71.1, respectively). SceneSmith retains slightly better collision and stability scores, while both methods have an out-of-bounds rate of 0.2%.

D.2 Generalization to Held-Out Conditions

Table 12 tests unseen layouts, asset shapes, and requirement combinations, with 60 tasks per condition. These conditions are excluded from SceneWiseDB training and validation; model and scoring settings remain fixed. External baselines retain their original pipelines, so the held-out designation applies specifically to SceneWiseDB.

SceneWise achieves SRs of 75.6%, 73.3%, and 68.9%, exceeding SAGE by 20.0–24.5 percentage points. Higher CSR and fewer edits across all three conditions indicate that its guidance remains effective under these specified distribution shifts.

D.3 Evaluation with Different World-Model Backbones

Table 13 evaluates SceneWise with InternVL3.5-8B, Gemma4-31B, and Qwen3.6-27B as the world-model backbone. Only the world model changes across backbone groups; the Designer and attribution components use the same configurations in each group. Evaluation uses 100 coupled-requirement prompts with three random seeds per prompt, sharing task inputs, validators, and execution-verification budgets.

Table 12: **Generalization to unseen layouts, assets, and requirement combinations.** SceneWise achieves higher SR and CSR with fewer edits across all three conditions. Each condition uses 60 tasks and is held out relative to SceneWiseDB. SR and CSR are percentages.

Method	**SR↑**	**CSR↑**	**EditOps↓**	**Edits↓**
Unseen layouts				
SAGE	55.6±10.3	82.1±3.1	14.6±2.1	6.3±0.9
SceneSmith	52.2±10.4	80.4±3.3	18.7±2.6	7.9±1.1
SceneWise (Ours)	**75.6±8.9**	**91.8±2.3**	**5.9±0.9**	**3.8±0.6**
Unseen assets				
SAGE	52.2±10.4	80.6±3.3	15.8±2.3	6.8±1.0
SceneSmith	48.9±10.5	78.9±3.5	20.4±2.8	8.5±1.2
SceneWise (Ours)	**73.3±9.1**	**90.7±2.5**	**6.5±1.0**	**4.1±0.6**
Unseen requirement combinations				
SAGE	44.4±10.4	76.8±3.6	18.2±2.6	7.5±1.1
SceneSmith	41.1±10.3	74.9±3.8	23.1±3.1	9.3±1.3
SceneWise (Ours)	**68.9±9.6**	**88.6±2.7**	**7.6±1.1**	**4.7±0.7**

For each backbone, we compare SceneWise with the joint ablation without consequence-guided generation and learned attribution, abbreviated as *w/o Gen. & Attr.* The ablation uses Designer ranking during generation and prompted-backbone ranking of historical decisions, while retaining consequence-guided repair, as defined in Appendix C.2.

Table 13: **Evaluation with different world-model backbones.** SceneWise achieves higher SR and CSR with fewer edited objects and verification calls and lower runtime for all three backbones. SR and CSR are percentages, and Time is in minutes. The joint ablation removes consequence-guided generation and learned attribution.

World-model backbone	Configuration	SR↑	CSR↑	Edits↓	Calls↓	Time↓
InternVL3.5-8B	w/o Gen. & Attr.	58.7	83.5	6.1	50.6	57.8
	SceneWise	74.7	90.0	4.3	34.1	39.4
Gemma4-31B	w/o Gen. & Attr.	64.0	86.1	5.5	46.8	69.4
	SceneWise	81.3	92.8	3.5	28.5	43.8
Qwen3.6-27B	w/o Gen. & Attr.	66.0	86.9	5.2	45.2	65.0
	SceneWise	83.3	93.7	3.3	26.3	40.2

Across the three backbones, SceneWise improves SR by 16.0–17.3 percentage points and CSR by 6.5–6.8 percentage points relative to the joint ablation. It also modifies fewer distinct objects, reduces verification calls by 32.6–41.8%, and lowers mean runtime by 31.8–38.2%. These improvements are observed for every tested backbone, supporting the benefits of generation guidance and learned attribution across these world-model configurations.

D.4 Generation Time Analysis

Table 14 reports end-to-end wall-clock time for SceneWise and its controlled ablations. Each variant is evaluated on 94 SceneEval room prompts and 100 coupled-requirement prompts, with three random seeds per prompt. Timing covers model inference, asset processing, scene execution, verification, and any repair performed before termination. All attempts, including unsuccessful ones, are included. The variants share the backbone, tools, and execution-verification budget.

Table 14: **End-to-end scene-generation time in minutes.** SceneWise has the lowest mean runtime and standard deviation among the compared variants in both task groups. Statistics include all attempts, including failures; Std denotes standard deviation, and Min and Max denote the minimum and maximum.

Method	Mean	Std	Min	Max
SceneEval prompts: 94 prompts × 3 seeds				
SceneWise (Full)	72.4	20.6	29.6	156.8
w/o consequence-guided generation	88.7	27.4	31.2	207.5
w/o learned attribution	81.9	24.8	30.1	191.4
w/o generation guidance and attribution	101.6	34.9	33.8	258.6
w/o consequence-guided repair	78.8	23.7	29.8	183.2
Coupled requirements: 100 prompts × 3 seeds				
SceneWise (Full)	38.6	12.4	14.2	87.5
w/o consequence-guided generation	51.3	18.2	16.1	131.6
w/o learned attribution	46.7	16.5	15.3	118.4
w/o generation guidance and attribution	62.4	23.1	18.6	167.9
w/o consequence-guided repair	44.9	15.6	14.8	110.7

SceneWise averages 72.4 minutes on SceneEval prompts and 38.6 minutes on coupled-requirement tasks. Removing both generation guidance and learned attribution increases these means to 101.6 and 62.4 minutes, respectively; the full framework reduces mean runtime by 28.7% and 38.1% relative to this joint ablation. Removing either component individually, or removing consequence-guided repair, also increases mean runtime in both groups. The full framework has the lowest standard deviation in each group, indicating less variation across the evaluated runs.

These end-to-end measurements include the inference cost of consequence prediction and attribution. Together with the generation results in Table 2, they show improved requirement satisfaction with lower mean runtime within the controlled comparisons. Generation timing starts at task initialization, whereas the repair-only times in Table 4 start at shared failed-scene checkpoints.

D.5 Structured Consequences versus Direct Scoring

Controlled comparison. We compare three prediction targets using the held-out consequence records and the 200 shared partial scenes from Section 4.3, with three seeds per scene. All trained variants receive the same inputs (S_t, Q_t, G_t, P, d) and use the same backbone, adapter configuration, training records, optimizer, batch size, and training steps. They differ only in the predicted target and the resulting candidate ranking. Historical attribution and repair settings remain fixed.

The *direct-risk predictor* outputs a scalar target $r = R(\mathcal{C}_t^H(d))$, computed from the execution-derived consequence record using the same risk weights and normalization as the full model. The *failure-only predictor* outputs V and ranks candidates using R_V. The *full consequence model* outputs $(\Delta F, \Delta R, V, A)$ and ranks candidates using R. Each variant also predicts the same termination-status metadata and uses token-level negative log-likelihood, with the same loss-averaging convention and fixed numerical serialization precision. Equal predicted scores are resolved by the original Designer proposal order.

All variants share the candidate sets, immediate validity checks, continuation policy, and execution-verification budget, with $K = 5$, $B = 2$, and $H = 3$. The shortlisted branches are verified with the same execution-measured risk and acceptance rules. SR and CSR are evaluated after genera-

tion terminates from each partial scene, using the complete original checklist; failure-free horizon termination alone does not count as success.

Table 15: **Comparison of candidate-scoring targets.** Training settings, candidate sets, and execution-verification budgets are held fixed. SR and CSR are reported as percentages over all attempts.

Predictor	SR↑	CSR↑
Direct-risk predictor	85.0	94.6
Failure-only predictor	83.3	93.5
Full consequence model	88.3	96.0

As shown in Table 15, the full consequence model achieves 88.3% SR and 96.0% CSR, exceeding direct-risk prediction by 3.3 and 1.4 percentage points, respectively. Its gains over failure-only prediction are 5.0 and 2.5 percentage points. These results support structured consequence prediction for improving candidate selection under the same execution-verification budget.

Prediction quality. We separately compare the prompted backbone and the trained full model on the same held-out consequence records, with both producing all four fields. We report mean absolute error (MAE) for ΔF and ΔR, matching pending-object and constraint identifiers and applying the execution-derived measurement masks. MAE is averaged over valid matched numerical entries. Numerical validity reports the percentage of all required numerical entries returned with a valid value; missing or malformed entries reduce this percentage and remain in the coverage denominator. Records with no applicable numerical targets contribute no entries to these numerical metrics.

For V, a predicted event matches a measured event only when its type, constraint identifier, and unordered set of participating object identifiers agree. For A, matching uses persistent object identifiers. Both F1 scores are micro-averaged from true positives, false positives, and false negatives accumulated across all test records. A nonempty prediction against an empty target contributes false positives; two empty sets contribute no counts. F1 is reported as not applicable if the entire evaluation set contains neither positive targets nor positive predictions.

Table 16: **Prediction quality of the four consequence outputs.** MAE uses the original feasible-fraction and normalized-margin units. F1 and numerical validity are percentages.

Predictor	ΔF MAE↓	ΔR MAE↓	V F1↑	A F1↑	Num. valid↑
Prompted backbone	0.12	0.18	64.0	66.0	95.0
Full consequence model	0.06	0.09	85.0	83.0	99.0

Table 16 shows improved prediction across all four consequence fields. Training reduces both numerical MAEs by 50%, from 0.12 to 0.06 for ΔF and from 0.18 to 0.09 for ΔR. F1 increases by 21.0 percentage points for failure events and 17.0 points for affected objects, while numerical validity improves from 95.0% to 99.0%.

E Additional Qualitative Results

Figure 6 presents additional comparisons of Holodeck, SAGE, SceneSmith, and SceneWise on bedroom, dining-room, kitchen, and office tasks. Each prompt combines a specified set of objects with requirements for access, operating space, or circulation. These examples extend the qualitative evaluation to layouts in which the placement of one object affects the space available for others.

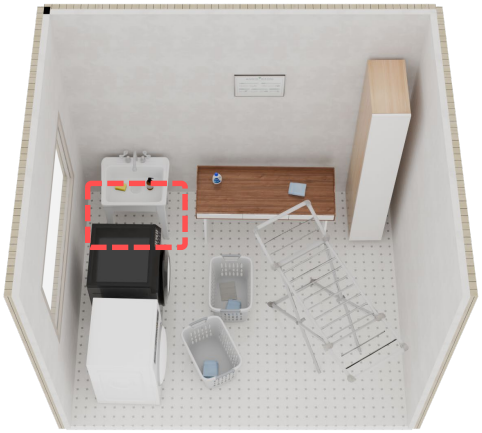
The bedroom task combines a double bed, two nightstands, storage furniture, a bench, and a reading area within a fixed room. It requires access to both sides of the bed and the storage fronts while preserving the route around the foot of the bed. The dining-room task combines a round table, four separate chairs, and storage furniture, with requirements for chair movement and cabinet access.

The kitchen task requires facing cabinet and worktop runs, several appliances, and a clear central aisle. The office task requires four individual workstations together with shared storage and equip-



Figure 6: Additional qualitative comparisons of Holodeck, SAGE, SceneSmith, and SceneWise. Columns show a bedroom, a dining room, a kitchen, and an office, with the corresponding input prompts below. SceneWise better balances dense furnishing requirements with functional organization, reserving space for bedside access, chair movement, and circulation between workstations.

"Set up a compact laundry room with a washer, dryer, utility sink, folding counter, tall storage cabinet, two laundry baskets, and a freestanding drying rack. Leave enough space to load both machines, open the cabinet doors, and use the sink and counter. **Keep the walkway clear.**"



(a) Text input (b) Generated scene

Figure 7: Failure case under coupled spatial and functional requirements in a compact laundry room. (a) Input prompt. (b) Scene generated by SceneWise. The requested objects are present, but clearance in front of the utility sink remains insufficient (red dashed box).

ment, while maintaining access to each desk and space to pull back its chair. These requirements make furniture dimensions and placement interdependent.

The rendered results show differences in object coverage, furniture arrangement, and the allocation of open space. The SceneWise dining-room example organizes seating around the central table with storage near the room boundaries, while the office example arranges workstations around a shared central area. Complete requirement satisfaction and functional clearance are evaluated using the original checklists and validators described in Sections C.3 and C.4.

F Failure Cases and Limitations

Coupled requirements in compact spaces. Compact rooms impose competing demands on object placement, circulation, and operating clearance. Figure 7 illustrates a remaining access conflict despite the inclusion of all requested objects. In such settings, clearing one functional area can reduce the space available for another, making coordinated adjustments across multiple objects necessary.

Finite candidate and verification budgets. The search space is constrained by available assets, proposed alternatives, and the execution-verification budget. Resolving tightly coupled conflicts may require exploring additional object configurations and coordinated revisions beyond the tested candidates. Broader candidate coverage and adaptive verification budgets could further improve recovery in these settings.