SpatialGuard: Harness-Guided Verifiable Spatial Reasoning for Text-to-Image Generation

Anonymous ACL submission

Abstract

Complex 3D spatial text to image generation requires models to convert natural language into stable visual geometry, not merely semantic appearance. Existing prompt-driven or layout-conditioned methods improve controllability, but often lack an optimizable and verifiable spatial intermediary before visual sampling. As a result, object relations, occlusion, visibility, and camera constraints can decay during multi-round generation. This paper presents SpatialGuard, a structured layout-guided framework for complex 3D spatial text-to-image generation. SpatialGuard parses prompts into image synthesis-oriented 3D layouts through a Spatial Layout Architect, realizes them as visual conditions and candidate images through a Visual Realizer, and uses a Visual Alignment Critic to validate consistency among prompt, layout, and image. To keep constraints stable across iterations, SpatialGuard introduces a Layout Harness that organizes rule constraints, tool invocation, shared knowledge, and feedback loops around the editable layout state. This design turns complex spatial generation from implicit prompt following into a verifiable process of planning, realization, validation, and repair. Comprehensive experiments show that SpatialGuard achieves state-of-the-art performance in complex 3D spatial layout generation and improves spatial faithfulness over existing text-to-image and layout control baselines.

1 Introduction

Complex 3D spatial text-to-image generation is a key capability for bridging natural-language interaction and controllable visual content production, with important applications in game-asset previewing (Husen et al., 2025; Li et al., 2025b), film storyboard design (Zhang et al., 2025a; Wang et al., 2026a), and embodied-intelligence simulation (Feng et al., 2026). These applications require models to translate spatial intent in natural language into stable visual geometry, rather than merely generating semantically related appearance content. Traditional prompt-driven methods (Pan et al., 2026a; Qiu et al., 2025; Han et al., 2026; Jiang et al., 2026) mainly rely on text-to-image models to infer spatial structure from text implicitly. Thus, in relation-dense, view-sensitive, and layout-constrained scenes, they often satisfy only local semantics and struggle to maintain global spatial consistency. Recent methods (Venkatesh et al., 2026; Qin et al., 2026; Agrawal et al., 2026; Zhang et al., 2025b) introduce layout conditions, hierarchical generation, 3D control signals, and multi-step planning to make spatial structure more explicit during generation, significantly improving the controllability of complex scenes. However, existing methods (Parihar et al., 2025; Li et al., 2025a) still face two key limitations. First, the generation process usually lacks an image-synthesis-oriented spatial layout intermediary that is optimizable and verifiable. As a result, complex spatial intent lacks a stable carrier before visual sampling, and text alone fails to provide sufficiently explicit and executable spatial constraints for subsequent synthesis. Second, when complex spatial intent requires multiple rounds of planning, generation, evaluation, and correction, large models are prone to context forgetting and constraint decay. Spatial requirements established in early stages are difficult to preserve in later interactions, weakening fidelity to the original prompt and degrading generation quality.

These limitations highlight two key requirements for complex spatial text-to-image gen-

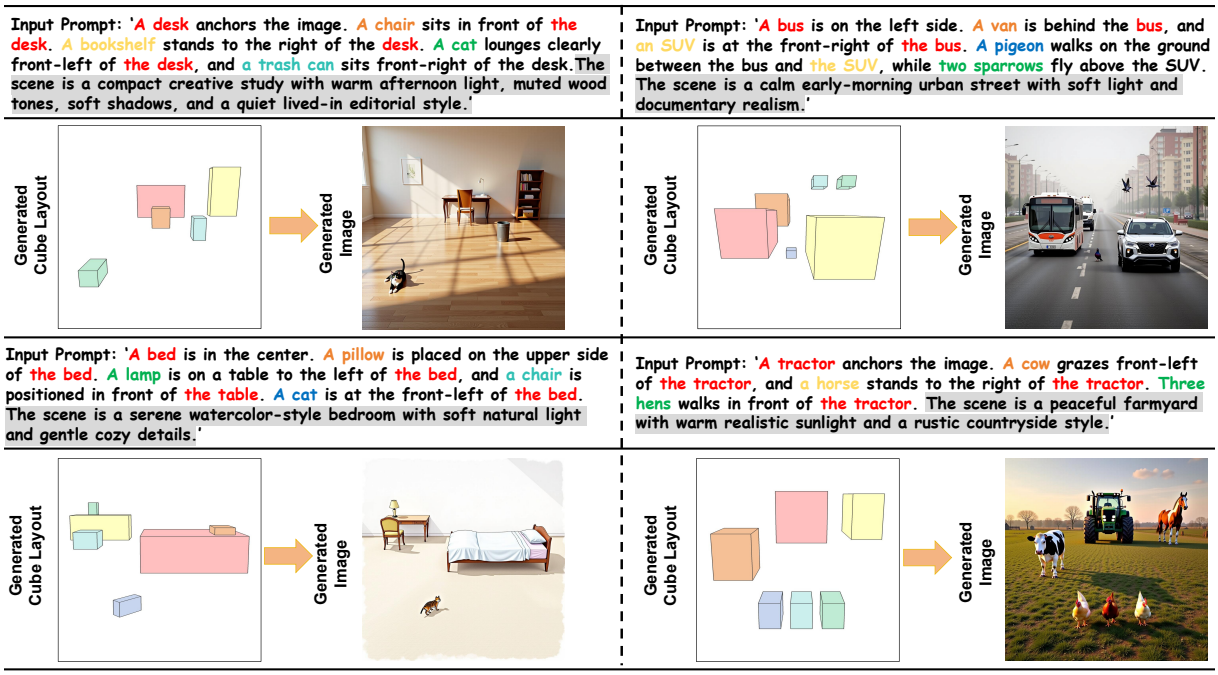

Figure 1: 1111.

eration. First, the model needs to form a verifiable layout representation before image sampling, allowing spatial constraints in text to be explicitly encoded, continuously optimized, and checked before generation. Second, the system needs a cross-round structured constraint enforcement mechanism, so that spatial requirements do not remain merely in prompts or scoring signals but are stably preserved, actively retrieved, and updated according to feedback across planning, execution, verification, and revision. The former provides explicit spatial targets for image synthesis, while the latter ensures that these targets remain effective in long-chain interactions despite memory decay. Together, they indicate a stronger generation paradigm: complex 3D layouts should not only be prompted by language, but also explicitly designed, executed, and maintained.

To address these issues, we propose SpatialGuard, a structured-layout-guided generation framework for complex 3D spatial text-to-image generation. It unifies verifiable layout representation and persistent constraint enforcement within a single generation process. SpatialGuard consists of three collaborative modules. The Spatial Layout Architect parses input text into an image-synthesis-oriented 3D structured layout, providing a checkable and editable representation of complex spatial intent before sampling. The Visual Realizer converts this layout into visual conditions and synthesizes images, bringing spatial targets into pixel space. The Visual Alignment Critic evaluates consistency among text, layout, and image, and feeds deviations back to subsequent planning. These modules operate around a shared layout state, preventing text understanding, image realization, and result evaluation from becoming decoupled across multi-round interaction.

Moreover, to prevent memory decay of spatial constraints during multi-round agent planning and interaction, we first introduce the idea of an agent harness (Lin et al., 2026; Li et al.; Pan et al., 2026b) into text-to-image generation and distill it into four core mechanisms: rule constraints, tool invocation, shared knowledge, and feedback loops. Rule constraints convert spatial requirements in language into checkable layout boundaries. Tool invocation turns layout decisions into externally executable operations. Shared knowledge preserves layout versions, evaluation judgments, and revision trajectories across stages. Feedback loops reinject detected spatial deviations into layout planning and image generation. Based on the automated project workflow, SpatialGuard generates an initial 3D layout

from text, performs deterministic verification according to the object list and spatial relations, iteratively invokes 3D layout rendering and image synthesis, and produces executable repair plans when spatial violations are detected. Thus, it shifts complex spatial text-to-image generation from prompt-driven implicit generation to verifiable generation guided by structured layouts and maintained by constraint mechanisms.

The main contributions of this paper are as follows:

- We propose SpatialGuard, an agentic layout generation framework for complex 3D spatial text-to-image generation, enabling closed-loop modeling from textual spatial intent to verifiable visual generation.
- We introduce the concept of agent harness into the text-to-image pipeline for the first time and build a cross-round spatial constraint enforcement mechanism to mitigate memory decay in complex spatial generation by large models.
- Comprehensive quantitative and qualitative experiments show that SpatialGuard achieves new state-of-the-art performance in complex 3D spatial text-to-image generation and significantly outperforms existing text-to-image and layout-control baselines.

2 Related Work

2.1 Text to Image Generation

Text-to-image generation has progressed from high-fidelity synthesis to more deliberate modeling of semantic and spatial intent. General models improve prompt adherence through reasoning, reward optimization, autoregressive modeling, and spatial evaluation signals. Janus-Pro-R1 (Pan et al., 2026a) and T2I-R1 (Jiang et al., 2026) connect visual comprehension with generation through reinforcement learning and chain of thought style planning, while NextStep-1 (Han et al., 2026) explores scalable autoregressive generation with continuous visual tokens (Han et al., 2026). SelfCross (Qiu et al., 2025) reduces subject mixing among similar objects, and SpatialScore provides a reward model for spatial relation evaluation and reinforcement learning (Tang et al., 2026). Recent benchmarks show that compositional and spatial alignment have become central axes for evaluating text-to-image models (Wang et al., 2026b,c; Huang et al., 2025). These works strengthen semantic alignment, but spatial composition is still largely learned or rewarded through implicit model behavior. A complementary line of work makes spatial control more explicit through planning, layout, pose, and 3D conditions. LayoutGPT (Feng et al., 2023) uses language models for compositional visual planning, while MCCD (Li et al., 2025a) and CREA (Venkatesh et al., 2026) explore multi-agent collaboration for complex composition and creative generation. LayerCraft (Zhang et al., 2025b) decomposes generation into layered reasoning and object integration. Compass-Control (Parihar et al., 2025) focuses on object orientation in multi-object scenes, SceneDesigner (Qin et al., 2026) models 9 DoF object pose control, and SeeThrough3D (Agrawal et al., 2026) introduces occlusion-aware 3D layout conditioning with camera control. These studies show that layouts, poses, layers, and planning improve controllability. However, existing methods still lack an optimizable and verifiable spatial layout intermediary, leaving complex spatial intent without a stable carrier before visual sampling and weakening executable spatial constraints for synthesis.

2.2 Harness for Agentic Execution

Agent harness research studies the execution layer around a foundation model, covering tools, state, orchestration, validation, observability, and recovery. Recent surveys organize harness design around execution environment, tool interface, context management, lifecycle control, verification, and governance (Li et al.; Meng et al., 2026). Agentic Harness Engineering evolves harness components through structured observability over files, trajectories, and decisions (Lin et al., 2026). Natural Language Agent Harnesses separate readable harness policy from runtime mechanisms (Pan et al., 2026b). Meta Harness treats harness optimization as an outer loop over code, traces, and scores (Lee et al., 2026). Beyond coding agents, harness-style designs have begun to appear in visual and embodied tasks. VASA

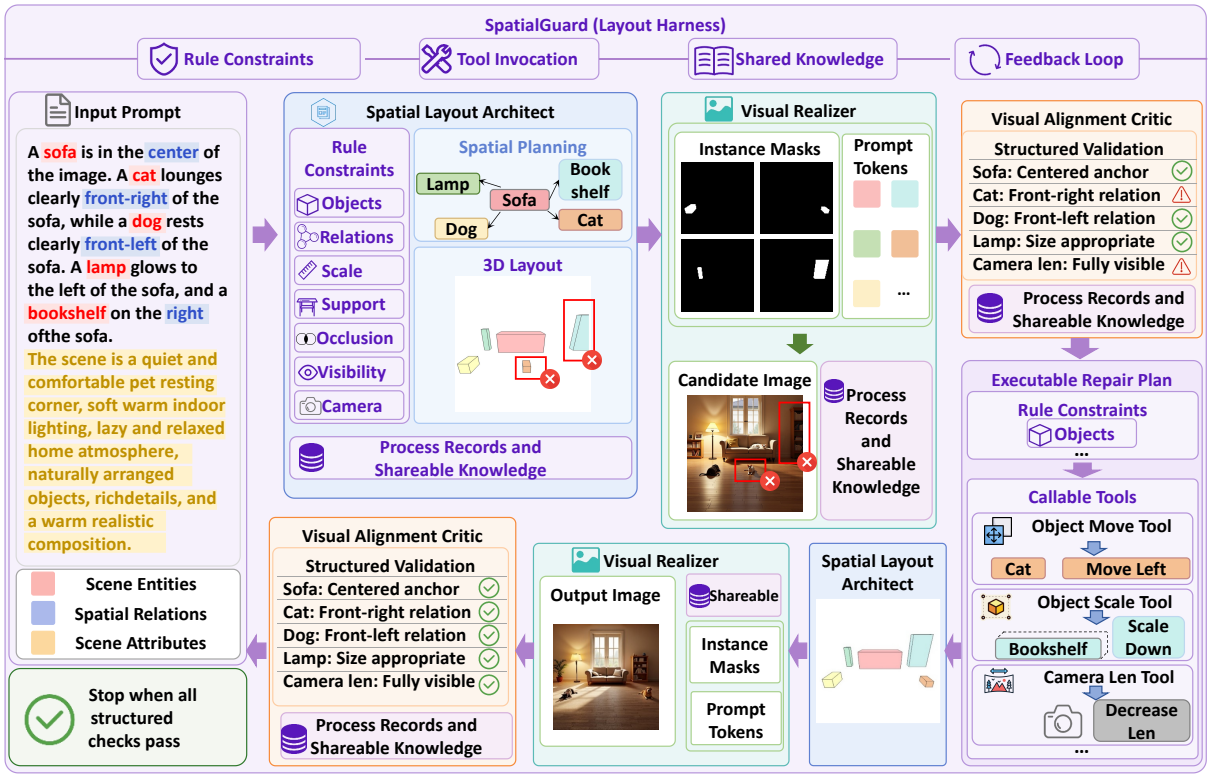

Figure 2: Overview of SpatialGuard. Given an input prompt, the Spatial Layout Architect parses objects, spatial relations, and scene appearance into a verifiable 3D layout under the Layout Harness. The Visual Realizer converts the layout into masks, token-aligned conditions, and a candidate image. The Visual Alignment Critic performs structured validation over object placement, relations, visibility, and camera constraints. Failed constraints are converted into executable repair actions and fed back to update the layout, while rule constraints, tool invocation, shared knowledge, and feedback loops preserve spatial intent across iterations until the final image satisfies the prompt.

maintains a persistent working mask for open ad hoc segmentation, turning visual progress into an inspectable state (Wang and Yu, 2026). SceneWeaver uses a reflective tool-based loop to synthesize 3D scenes under semantic and physical feedback (Yang et al., 2026). These systems indicate that explicit state, callable tools, validation gates, and shared memory can help complex layout text-to-image generation keep spatial intent readable, enforceable, and recoverable.

3 Methodology

3.1 Overview

Complex 3D spatial text-to-image generation requires preserving object identity, relative position, visibility, camera configuration, and realism from language understanding to image synthesis. Existing controllable methods use layouts, 3D controls, layered generation, or agentic planning to improve spatial grounding (Agrawal et al., 2026; Qin et al., 2026; Parihar et al., 2025; Zhang et al., 2025b; Li et al., 2025a), but spatial requirements often remain scattered across prompts, conditions, and evaluation signals. Relations understood early may weaken after synthesis or disappear during revision. SpatialGuard addresses this by combining agent-based spatial reasoning with a Layout Harness, as shown in Fig. 2.

Given an input prompt, SpatialGuard separates object mentions, spatial intentions, and appearance descriptions, then organizes them into a layout-centered generation loop. The system plans a structured scene, realizes it as visual evidence, checks whether the image follows the spatial intent, and repairs the layout when violations appear. In Fig. 2, the input enters the Spatial Layout Architect, the Visual Realizer produces a candidate image, the Visual Alignment Critic performs structured validation, and failed items return through an executable repair plan.

The Layout Harness keeps this loop stable. Rule constraints make object, relation, scale, support, occlusion, visibility, and camera requirements checkable. Tool invocation maps revisions to operations such as moving objects, rescaling objects, and adjusting the camera. Shared knowledge stores layout decisions, validation records, and repair traces across rounds. Feedback loops send detected spatial errors back to planning rather than leaving them as passive scores. SpatialGuard therefore turns prompt-only sampling into a verifiable process that repeatedly plans, realizes, validates, and repairs the scene.

3.2 SpatialGuard Framework

SpatialGuard instantiates the common interface in Sec . 3.1 as an agentic layout generation framework. Prior layout-driven and 3D-controlled text-to-image methods make spatial control more explicit (Feng et al., 2023; Agrawal et al., 2026; Qin et al., 2026; Parihar et al., 2025), but often pass the layout as a static generator condition. SpatialGuard instead treats the layout as an editable state that links language understanding, visual synthesis, and result validation, moving complex 3D spatial intent from implicit prompt attention into a state that can be planned, grounded, and inspected.

Following Sec. 3.1, $\boldsymbol{p}$ denotes the input prompt, $\boldsymbol{\mathcal{O}} = \{\boldsymbol{o}_i\}_{i=1}^{N}$ denotes the set of N mentioned objects,$\boldsymbol{o}_i$ denotes the ith object. $\boldsymbol{\mathcal{C}}$ denotes the spatial constraint set extracted from $\boldsymbol{p}$, and $\boldsymbol{a}$ denotes the appearance and environment description. At round t, the layout state is written as

$$\boldsymbol{L}^{(t)} = \left(\left\{ \left(\boldsymbol{c}_i, \boldsymbol{x}_i^{(t)}, \boldsymbol{s}_i^{(t)}, \boldsymbol{\alpha}_i^{(t)} \right) \right\}_{i=1}^{N}, \boldsymbol{\kappa}^{(t)} \right), \tag{1}$$

where $\boldsymbol{c}_i$ is the category of object $\boldsymbol{o}_i$, $\boldsymbol{x}_i^{(t)}$ is the 3D position of object $\boldsymbol{o}_i$ at round t, $\boldsymbol{s}_i^{(t)}$ is its layout-readable size at round t, $\boldsymbol{\alpha}_i^{(t)}$ is its azimuth at round t, and $\boldsymbol{\kappa}^{(t)}$ is the camera parameter vector at round t. SpatialGuard operates on this state through three cooperative modules.

The **Spatial Layout Architect** converts the parsed spatial intent into an executable geometry carrier. It initializes the first layout and later revises the layout according to structured feedback:

$$\begin{aligned} \boldsymbol{L}^{(0)} &= \mathcal{A}_s \left(\boldsymbol{p}, \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{C}}, \boldsymbol{a} \right), \\ \boldsymbol{L}^{(t+1)} &= \mathcal{A}_s \left(\boldsymbol{p}, \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{C}}, \boldsymbol{a}, \boldsymbol{L}^{(t)}, \boldsymbol{\Delta}^{(t)} \right), \end{aligned} \tag{2}$$

where $\mathcal{A}_s$ denotes the Spatial Layout Architect, $\boldsymbol{L}^{(0)}$ denotes the initial layout state, $\boldsymbol{L}^{(t+1)}$ denotes the next layout state after revision, and $\boldsymbol{\Delta}^{(t)}$ denotes the repair feedback produced by the Visual Alignment Critic at round t. The architect resolves object mentions, normalizes categories and counts, decomposes spatial phrases into 3D and screen space requirements, and assigns positions, sizes, orientations, and camera parameters. For example, a front right relation is represented through depth ordering and lateral placement, keeping it checkable after projection.

The **Visual Realizer** maps the current layout state into visual conditions and synthesizes a candidate image. Given $\boldsymbol{L}^{(t)}$, it renders the layout, derives instance-level masks, aligns object regions with the corresponding prompt tokens, and invokes the generator:

$$\boldsymbol{I}^{(t)} = \mathcal{G} \left(\boldsymbol{p}, \boldsymbol{a}, \rho \left(\boldsymbol{L}^{(t)} \right), \psi \left(\boldsymbol{p}, \boldsymbol{L}^{(t)} \right) \right), \tag{3}$$

where ρ denotes the layout renderer, ψ denotes the operator that produces instance masks and token-aligned object regions, $\mathcal{G}$ denotes the layout-conditioned image generator, and $\boldsymbol{I}^{(t)}$ denotes the candidate image generated in round t. The rendered layout provides global geometry, masks preserve object boundaries, and token alignment keeps appearance phrases attached to targets, transferring the planned 3D arrangement into pixel space.

The **Visual Alignment Critic** closes the loop by judging whether the candidate image respects the prompt and the current layout:

$$\left(\boldsymbol{v}^{(t)}, \boldsymbol{\Delta}^{(t)} \right) = \mathcal{A}_c \left(\boldsymbol{p}, \boldsymbol{\mathcal{C}}, \boldsymbol{L}^{(t)}, \boldsymbol{I}^{(t)} \right), \tag{4}$$

where $\mathcal{A}_c$ denotes the Visual Alignment Critic, $\boldsymbol{v}^{(t)}$ is a structured validation record, and $\boldsymbol{\Delta}^{(t)}$ is the repair feedback for the next layout revision. The critic reads $\boldsymbol{\mathcal{C}}$, which contains the required object relations, support conditions, visibility requirements, and camera-related constraints extracted from $\boldsymbol{p}$. It localizes failures to concrete fields of $\boldsymbol{L}^{(t)}$, such as object position, size, azimuth, occlusion status,

or camera configuration, instead of collapsing the judgment into a single scalar score. If $\boldsymbol{v}^{(t)}$ contains no failed item, SpatialGuard returns $\boldsymbol{I}^{(t)}$ as the final output. Otherwise, $\boldsymbol{\Delta}^{(t)}$ is passed back to the Spatial Layout Architect to produce $\boldsymbol{L}^{(t+1)}$.

This separation gives each module a clear responsibility: the Spatial Layout Architect makes spatial intent explicit, the Visual Realizer turns it into visual evidence, and the Visual Alignment Critic converts deviations into actionable layout feedback. SpatialGuard, therefore, supports closed-loop modeling from textual spatial intent to verifiable visual generation without requiring one-pass implicit spatial inference by the image generator.

3.3 Layout Harness for Persistent Spatial Execution

Existing text-to-image systems make spatial control more explicit through layouts, 3D conditions, layered composition, and agent collaboration (Feng et al., 2023; Agrawal et al., 2026; Qin et al., 2026; Parihar et al., 2025; Zhang et al., 2025b; Li et al., 2025a). However, these signals are often used as local prompts, static conditions, or post hoc scores, which are fragile for relation-dense scenes. A relation parsed early may vanish during object repair, visibility adjustment, or camera change. The core challenge is therefore to preserve, retrieve, and enforce relations across planning, realization, validation, and revision.

Agent harness research provides an execution view for this challenge (Lin et al., 2026; Li et al.; Pan et al., 2026b). Although a harness has no single mathematical definition, prior work commonly treats it as an external layer that organizes model calls, tools, state, validation, recovery, and stopping conditions. From this view, we distill four properties for complex spatial generation: rule constraints, tool invocation, shared knowledge, and feedback loops. SpatialGuard instantiates them as a Layout Harness around the editable layout state introduced in Sec . 3.2.

At refinement round t, where t indexes the current iteration, the Layout Harness maintains

$$\boldsymbol{H}^{(t)} = \left(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{T}}, \boldsymbol{M}^{(t)}\right). \tag{5}$$

Here $\boldsymbol{H}^{(t)}$ is the harness state at round t, $\boldsymbol{\mathcal{C}}$ is the spatial constraint set extracted from the input prompt $\boldsymbol{p}$, $\boldsymbol{\mathcal{T}}$ is the executable layout tool library, and $\boldsymbol{M}^{(t)}$ is the shared knowledge state. $\boldsymbol{M}^{(t)}$ records the object manifest, parsed spatial predicates, previous layout states, validation records, and repair actions, giving all modules a durable reference beyond a single model call.

Rule constraints make linguistic requirements inspectable. SpatialGuard stores relations such as front, left, support, or full visibility in $\boldsymbol{\mathcal{C}}$ and checks them against $\boldsymbol{L}^{(t)}$ and $\boldsymbol{I}^{(t)}$. As defined in Sec. 3.2, $\boldsymbol{L}^{(t)}$ is the 3D layout state at round t, and $\boldsymbol{I}^{(t)}$ is the candidate image generated at round t. The Visual Alignment Critic produces $\boldsymbol{v}^{(t)}$ and $\boldsymbol{\Delta}^{(t)}$, where $\boldsymbol{v}^{(t)}$ lists passed and failed constraints and $\boldsymbol{\Delta}^{(t)}$ specifies the next layout revision.

Tool invocation turns a revision instruction into an executable operation. Given the validation result, the harness selects and applies a sequence of tool actions:

$$\begin{aligned} \boldsymbol{u}^{(t)} &= \Gamma\left(\boldsymbol{v}^{(t)}, \boldsymbol{\Delta}^{(t)}, \boldsymbol{H}^{(t)}\right), \\ \tilde{\boldsymbol{L}}^{(t+1)} &= \Omega\left(\boldsymbol{L}^{(t)}, \boldsymbol{u}^{(t)}\right), \\ \boldsymbol{M}^{(t+1)} &= \Xi\left(\boldsymbol{M}^{(t)}, \tilde{\boldsymbol{L}}^{(t+1)}, \boldsymbol{v}^{(t)}, \boldsymbol{u}^{(t)}\right). \end{aligned} \tag{6}$$

In this equation, $\boldsymbol{u}^{(t)}$ is the selected sequence of tool actions at round t, Γ is the tool selection function, Ω is the execution function that applies $\boldsymbol{u}^{(t)}$ to $\boldsymbol{L}^{(t)}$, $\tilde{\boldsymbol{L}}^{(t+1)}$ is the tool-executed intermediate layout state, Ξ is the memory update function, and $\boldsymbol{M}^{(t+1)}$ is the updated shared knowledge state. The tools include object translation, object scaling, support adjustment, occlusion control, visibility restoration, focal length adjustment, and camera elevation adjustment. Because they operate on explicit layout fields, a failed relation can be repaired while preserving constraints that already pass validation.

Shared knowledge keeps spatial decisions recoverable across rounds. The state $\boldsymbol{M}^{(t)}$ tells later modules which objects exist, which relations come from $\boldsymbol{p}$, which fields of $\boldsymbol{L}^{(t)}$ changed, and which failures remain. This avoids reliance on incomplete conversation history and makes each image traceable to a layout version, validation record, and repair action sequence.

Feedback loops connect validation to the next execution step. When $\boldsymbol{v}^{(t)}$ contains failed

Table 1: Evaluation results on the GenSpace benchmark for spatially-aware text-to-image generation. All metrics are better when higher. The best results are highlighted in bold and the second-best results are underlined.

Model	Spatial Pose				Spatial Relation				Spatial Measurement				Avg. Score ↑
	Camera ↑	Object ↑	Complex ↑	Avg. ↑	Ego. ↑	Allo. ↑	Intri. ↑	Avg. ↑	Size ↑	ObjDis ↑	CamDis ↑	Avg. ↑	
HunyuanImage-2.1 (Tencent, 2025)	41.78	31.06	<u>12.70</u>	<u>28.51</u>	69.42	<u>13.67</u>	23.18	35.42	30.31	28.62	26.42	28.45	<u>30.79</u>
SD-3.5-L (Esser et al., 2024)	<u>42.85</u>	<u>31.48</u>	5.90	26.74	**73.03**	11.15	<u>23.55</u>	<u>35.91</u>	<u>31.03</u>	**33.05**	24.83	29.64	30.76
FLUX.1-dev (Black Forest Labs, 2024)	40.42	31.11	12.28	27.94	63.39	13.17	19.40	31.99	29.16	30.72	**31.98**	<u>30.62</u>	30.18
Self-cross (Qiu et al., 2025)	24.83	18.74	7.16	16.91	41.57	8.36	10.92	20.28	17.85	19.46	16.73	18.01	18.40
T2I-R1 (Jiang et al., 2026)	29.64	22.81	8.93	20.46	47.23	9.74	12.86	23.28	20.58	21.94	18.22	20.25	21.33
Janus-pro-r1 (Pan et al., 2026a)	32.18	25.46	9.67	22.44	52.16	10.48	14.37	25.67	22.74	24.01	19.85	22.20	23.44
NextStep-1 (Han et al., 2026)	35.71	27.92	10.84	24.82	56.83	11.96	16.24	28.34	24.93	25.36	21.47	23.92	25.69
SpatialGuard(Ours)	**43.58**	**32.24**	**13.06**	**29.63**	<u>72.64</u>	**13.94**	**23.83**	**36.80**	**31.36**	<u>32.71</u>	<u>31.76</u>	**31.94**	**32.79**

constraints, $\boldsymbol{\Delta}^{(t)}$ is routed through Γ to obtain $\boldsymbol{u}^{(t)}$, and Ω produces the tool-executed intermediate layout state $\tilde{\boldsymbol{L}}^{(t+1)}$, which is then used by the Spatial Layout Architect to form the next layout state $\boldsymbol{L}^{(t+1)}$. When no failed constraint remains, SpatialGuard returns $\boldsymbol{I}^{(t)}$ as the final image. Through this harness, text to image generation becomes a persistent spatial execution process in which language constraints are externalized, checked, repaired, and remembered until the layout and image satisfy the requested spatial intent.

4 Experiments

4.1 Implementation Details

Implementation Details. SpatialGuard is a training-free framework, and all model weights remain frozen during evaluation. We conduct all experiments on a single NVIDIA H200 GPU with 143 GB of memory. Spatial Layout Architect and Visual Alignment Critic are implemented with GPT5 (OpenAI, 2025), where the former produces executable 3D layouts and the latter returns structured validation records and repair instructions. Visual Realizer is built on FLUX.1 dev (Black Forest Labs, 2024). We render layouts at 1024 resolution and use 512 resolution conditions for image synthesis. The default inference setting uses 25 denoising steps, a guidance scale of 3.5, and up to 4 layout verification and repair rounds.

4.2 Quantitative Evaluation

Table 1 reports quantitative results on GenSpace across spatial pose, spatial relation, and spatial measurement. SpatialGuard obtains the best overall score of 32.79, improving the strongest baseline HunyuanImage 2.1 by 2.00 points. The gain is consistent across the three metric groups: SpatialGuard ranks first on pose average 29.63, relation average 36.80, and measurement average 31.94. These results show that the improvement does not come from a single local metric, but from balanced gains in camera placement, object layout, relation grounding, and distance estimation. Compared with strong generative backbones and recent reasoning based text to image methods, SpatialGuard benefits from explicit layout planning and harness guided verification, which keep spatial constraints inspectable before synthesis and recoverable during iterative repair.

4.3 Qualitative Evaluation

Figure 3 compares SpatialGuard with recent text-to-image baselines on complex prompts containing object completeness, directional relations, and scene-level constraints. Baselines often produce visually plausible images, but their spatial structure is unstable. In the sofa scene, the prompt requires the cat and dog to appear in front of the sofa, while Self cross fails to preserve this layout. FLUX.1 dev (Black Forest Labs, 2024) generates only a partial bookshelf, and T2I-R1 (Jiang et al., 2026) omits the lamp and bookshelf in the first example. In the bus scene, Janus-pro-r1 (Pan et al., 2026a) misses the bird, breaking the requested back right relation. SpatialGuard avoids these failures by maintaining an explicit layout state and checking object, relation, and visibility constraints before accepting the output. This leads to images that are not only realistic but also faithful to the full spatial intent.

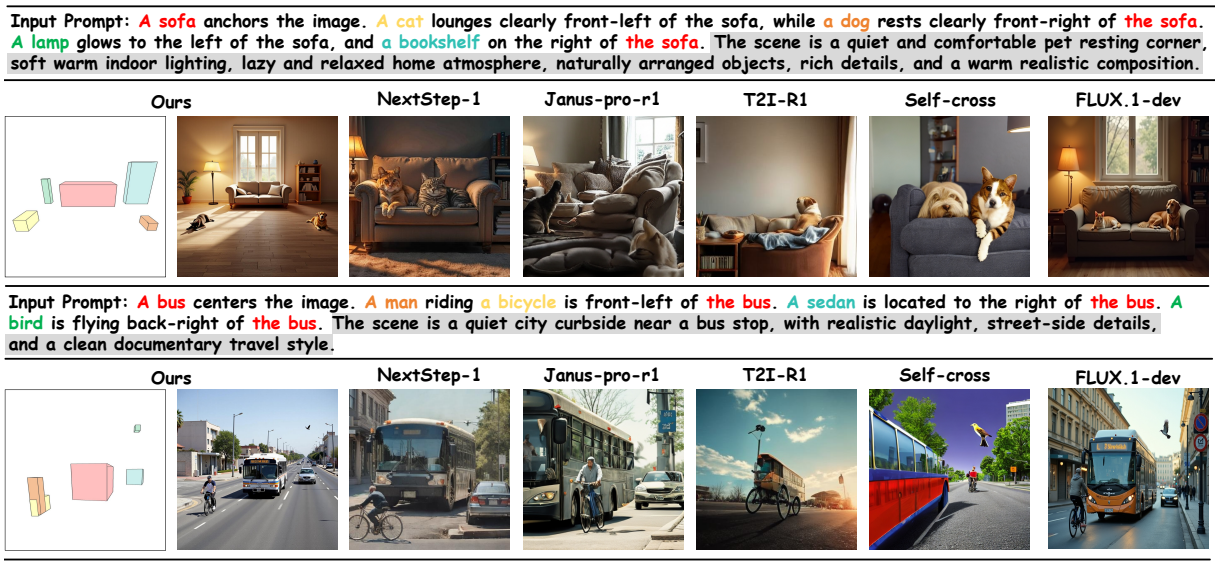

Figure 3: Qualitative comparison on complex spatial prompts. SpatialGuard preserves object completeness and relative layout more reliably than baselines, which often miss objects, truncate visible entities, or violate directional relations.

Table 2: Ablation study on the GenSpace benchmark for spatially-aware text-to-image generation. All metrics are better when higher. The best results are highlighted in bold and the second-best results are underlined.

Model	Spatial Pose				Spatial Relation				Spatial Measurement				Avg. Score ↑
	Camera ↑	Object ↑	Complex ↑	Avg. ↑	Ego. ↑	Allo. ↑	Intri. ↑	Avg. ↑	Size ↑	ObjDis ↑	CamDis ↑	Avg. ↑	
Ours w/o Spatial Layout Architect	39.86	29.08	11.45	26.80	70.31	13.12	22.48	35.30	30.05	30.72	30.16	30.31	30.80
Ours w/o Visual Realizer	40.24	29.47	11.63	27.11	70.58	13.24	22.61	35.48	30.11	30.79	30.24	30.38	30.99
Ours w/o Visual Alignment Critic	40.77	29.95	11.76	27.49	70.91	13.38	22.69	35.66	30.24	30.86	30.27	30.46	31.20
Ours w/o harness	41.31	30.62	12.10	28.01	71.23	13.49	22.78	35.83	30.58	31.04	30.32	30.65	31.50
Ours(full)	**43.58**	**32.24**	**13.06**	**29.63**	**72.64**	**13.94**	**23.83**	**36.80**	**31.36**	**32.71**	**31.76**	**31.94**	**32.79**

4.4 Ablation Studies

Table 2 studies the contribution of each component to GenSpace. Removing any module consistently weakens the overall score, confirming that SpatialGuard depends on both structured planning and closed-loop verification. The full model achieves 32.79, while removing the Layout Harness reduces the score to 31.50. This drop shows that rule constraints, shared state, tool invocation, and feedback are necessary for preserving spatial intent across rounds. Removing the Visual Alignment Critic further lowers the score to 31.20, indicating that post-generation validation is essential for converting visible spatial errors into targeted repairs. The strongest degradation appears when the Spatial Layout Architect is removed, where the score falls to 30.80 and the spatial pose average drops from 29.63 to 26.80. This confirms that an explicit 3D layout is the central carrier that makes object positions, relations, and camera constraints executable before image synthesis.

5 Conclusion

This paper introduces SpatialGuard, a structured layout-guided framework for complex 3D spatial text-to-image generation. SpatialGuard externalizes spatial intent into an editable 3D layout state and organizes generation as planning, realization, validation, and repair, rather than leaving spatial relations to implicit prompt interpretation. Its Layout Harness stabilizes this process with rule constraints, tool invocation, shared knowledge, and feedback loops, keeping object relations, visibility, and camera constraints recoverable across iterations. Experiments on GenSpace show clear gains in spatial pose, relation grounding, and measurement accuracy over strong text-to-image baselines. Qualitative and ablation results further show that these gains come from verifiable layout execution.

Limitations

SpatialGuard focuses on spatially grounded scenes that can be expressed through object lists, relations, visibility constraints, and camera parameters. Prompts with highly abstract artistic intent or intentionally ambiguous spatial descriptions may require additional interpretation rules. Since the framework performs planning, validation, and repair before producing the final image, its inference cost is higher than a single direct call to a text-to-image model. In addition, the final visual quality still depends on the capability of the underlying image generator. Future work can extend the tool library to finer physical interactions and improve efficiency through lighter validation strategies.

References

Vaibhav Agrawal, Rishubh Parihar, Pradhaan S Bhat, Ravi Kiran Sarvadevabhatla, and Venkatesh Babu Radhakrishnan. 2026. Seethrough3d: Occlusion aware 3d control in text-to-image generation.

Black Forest Labs. 2024. FLUX. https://github.com/black-forest-labs/flux. Accessed: 2026-05-26.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, and 1 others. 2024. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*.

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

Zhaohan Feng, Ruiqi Xue, Lei Yuan, Yang Yu, Ning Ding, Meiqin Liu, Bingzhao Gao, Jian Sun, Xinhu Zheng, and Gang Wang. 2026. Multi-agent embodied ai: Advances and future directions. *Science China Information Sciences*, 69(5):151202.

Chunrui Han, Guopeng Li, Jingwei Wu, Quan Sun, Yan Cai, Yuang Peng, Zheng Ge, Deyu Zhou, Haomiao Tang, Hongyu Zhou, and 1 others. 2026. Nextstep-1: Toward autoregressive image generation with continuous tokens at scale. In *The Fourteenth International Conference on Learning Representations*.

Kaiyi Huang, Chengqi Duan, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. 2025. T2i-compbench++: An enhanced and comprehensive benchmark for compositional text-to-image generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(5):3563–3579.

Dharma Hutama Husen, Wirawan Istiono, and 1 others. 2025. Procedural story generation for visual novels using large language models and text-to-image techniques. *Journal of Games, Game Art, and Gamification*, 10(3):106–114.

Dongzhi Jiang, Ziyu Guo, Renrui Zhang, Zhuofan Zong, Hao Li, Le Zhuo, Shilin Yan, Pheng-Ann Heng, and Hongsheng Li. 2026. T2i-r1: Reinforcing image generation with collaborative semantic-level and token-level cot. *Advances in Neural Information Processing Systems*, 38:39856–39890.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*.

Junjie Li, Xi Xiao, Yunbei Zhang, Chen Liu, Lin Zhao, Xiaoying Liao, Yingrui Ji, Janet Wang, Jianyang Gu, Yingqiang Ge, and 1 others. Agent harness engineering: A survey.

Mingcheng Li, Xiaolu Hou, Ziyang Liu, Dingkang Yang, Ziyun Qian, Jiawei Chen, Jinjie Wei, Yue Jiang, Qingyao Xu, and Lihua Zhang. 2025a. Mccd: Multi-agent collaboration-based compositional diffusion for complex text-to-image generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 13263–13272.

Ruihuang Li, Caijin Zhou, Shoujian Zheng, Jianxiang Lu, Jiabin Huang, Comi Chen, Junshu Tang, Guangzheng Xu, Jiale Tao, Hongmei Wang, and 1 others. 2025b. Hunyuan-game: Industrial-grade intelligent game creation model. *arXiv preprint arXiv:2505.14135*.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Xuanjing Huang, Hang Yan, Zhenhua Han, and Tao Gui. 2026. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*.

Qianyu Meng, Yanan Wang, Liyi Chen, Qimeng Wang, Chengqiang Lu, Wei Wu, Yan Gao, Yi Wu, and Yao Hu. 2026. Agent harness for large language model agents: A survey.

OpenAI. 2025. Gpt-5 system card. https://openai.com/index/gpt-5-system-card/. Accessed: 2026-05-26.

Kaihang Pan, Yang Wu, Wendong Bu, Shen Kai, Juncheng Li, Yingting Wang, Yunfei Li, Siliang Tang, Jun Xiao, Fei Wu, and 1 others. 2026a. Janus-pro-r1: Advancing collaborative visual comprehension and generation via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:60013–60041.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. 2026b. Natural-language agent harnesses. *arXiv preprint arXiv:2603.25723*.

Rishubh Parihar, Vaibhav Agrawal, Sachidanand VS, and Venkatesh Babu Radhakrishnan. 2025. Compass control: Multi object orientation control for text-to-image generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 2791–2801.

Zhenyuan Qin, Xincheng Shuai, and Henghui Ding. 2026. Scenedesigner: Controllable multi-object image generation with 9-dof pose manipulation. *Advances in Neural Information Processing Systems*, 38:133376–133400.

Weimin Qiu, Jieke Wang, and Meng Tang. 2025. Self-cross diffusion guidance for text-to-image synthesis of similar subjects. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 23528–23538.

Zhenyu Tang, Chaoran Feng, Yufan Deng, Jie Wu, Xiaojie Li, Rui Wang, Yunpeng Chen, and Daquan Zhou. 2026. Enhancing spatial understanding in image generation via reward modeling. *arXiv preprint arXiv:2602.24233*.

Tencent. 2025. HunyuanImage-2.1. https://huggingface.co/tencent/HunyuanImage-2.1. Accessed: 2026-05-26.

Kavana Venkatesh, Connor Dunlop, and Pinar Yanardag. 2026. Crea: A collaborative multi-agent framework for creative image editing and generation. *Advances in Neural Information Processing Systems*, 38:171332–171392.

Xinran Wang, Songyu Xu, Shan Xiangxuan, Yuxuan Zhang, Muxi Diao, Xueyan Duan, Kongming Liang, Zhanyu Ma, and 1 others. 2026a. Cinetechbench: A benchmark for cinematographic technique understanding and generation. *Advances in Neural Information Processing Systems*, 38.

Zehan Wang, Jiayang Xu, Ziang Zhang, Tianyu Pang, Chao Du, Hengshuang Zhao, and Zhou Zhao. 2026b. Genspace: Benchmarking spatially-aware image generation. *Advances in Neural Information Processing Systems*, 38.

Zengbin Wang, Xuecai Hu, Yong Wang, Feng Xiong, Man Zhang, and Xiangxiang Chu. 2026c. Everything in its place: Benchmarking spatial intelligence of text-to-image models. *arXiv preprint arXiv:2601.20354*.

Zilin Wang and Stella X Yu. 2026. Vision harnessing agent for open ad-hoc segmentation. *arXiv preprint arXiv:2605.19410*.

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. 2026. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *Advances in neural information processing systems*, 38:140319–140351.

Ruihan Zhang, Borou Yu, Jiajian Min, Yetong Xin, Zheng Wei, Juncheng Nemo Shi, Mingzhen Huang, Xianghao Kong, Nix Liu Xin, Shanshan Jiang, and 1 others. 2025a. Generative ai for film creation: A survey of recent advances. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 6267–6279.

Yuyao Zhang, Jinghao Li, and Yu-Wing Tai. 2025b. Layercraft: Enhancing text-to-image generation with cot reasoning and layered object integration. *arXiv preprint arXiv:2504.00010*.

A The Prompt Design of SpatialGuard

This section reports the prompt templates used by SpatialGuard. The prompts are written as executable templates that connect language parsing, layout execution, image realization, validation, and repair. Fields enclosed by braces are runtime placeholders filled by the current prompt, the editable layout state, the generated visual evidence, or the shared knowledge maintained by the Layout Harness.

A.1 Spatial Layout Architect

The Spatial Layout Architect converts the input prompt into an explicit 3D layout state. It identifies objects, decomposes spatial phrases into checkable relations, assigns object geometry and camera parameters, and revises the layout from structured repair feedback. Table 3 gives the corresponding template.

A.2 Visual Realizer

The Visual Realizer grounds the current layout in pixel space. It renders the layout, derives instance masks and token aligned object regions, and uses FLUX.1 dev to synthesize the candidate image while keeping the layout constraints visible to the generator. The full template appears in Table 4.

A.3 Visual Alignment Critic

The Visual Alignment Critic judges the consistency among the prompt, the layout state,

and the generated image. Rather than returning a single preference score, it localizes each violation to a concrete object, relation, visibility condition, or camera field, which makes the next repair step executable. Table 5 lists the prompt used for this stage.

A.4 Layout Harness

The Layout Harness organizes the execution layer around the editable layout state. It keeps rules, tools, shared memory, and feedback in a persistent loop, so that satisfied constraints remain recoverable while failed constraints are converted into field level repair actions. Table 6 presents its execution prompt.

Table 3: The prompt template for the Spatial Layout Architect.

Spatial Layout Architect

[INST]<SYS>
You operate as the Spatial Layout Architect and parse the input prompt into an executable 3D layout state for image synthesis. Identify all required objects, normalize categories and counts, decompose compound spatial phrases into primitive relations, and assign each object a category, 3D position, layout readable size, azimuth, support status, visibility requirement, and relation list. Maintain camera parameters and a record of constraints that must be checked after rendering. If repair feedback is provided, update only the fields named by the feedback and preserve constraints that already pass validation. Your output format is {layout state, object manifest, spatial constraint set, camera parameters, memory update}.
</SYS>

<USER>
The input prompt is `{input prompt}`. The appearance description is `{appearance description}`. The parsed object candidates are `{object candidates}`. The rule bank is `{rule constraints}`. The previous layout state is `{previous layout state}`. The repair feedback is `{repair feedback}`. The shared knowledge is `{shared knowledge}`. Produce the next layout state and make every object, relation, scale, support, occlusion, visibility, and camera constraint explicit.
</USER>[INST]

Table 4: The prompt template for the Visual Realizer.

Visual Realizer

[INST]<SYS>
You are responsible for transforming the current 3D layout into visual conditions for image synthesis. Render the layout, produce instance masks, align prompt tokens with object regions, and compose the generator prompt without altering required spatial constraints. Use the layout rendering as global geometry and the instance masks as object region anchors. Invoke FLUX.1 dev for image synthesis and return the candidate image metadata, condition manifest, and any realization issue that may affect validation. Your output format is {candidate image, rendered layout, instance masks, token region map, condition manifest, memory update}.
</SYS>

<USER>
The input prompt is `{input prompt}`. The appearance description is `{appearance description}`. The current layout state is `{layout state}`. The rendered layout is `{rendered layout}`. The instance masks are `{instance masks}`. The token region map is `{token region map}`. The shared knowledge is `{shared knowledge}`. Generate a candidate image that preserves object identity, boundary regions, spatial relations, visibility, and camera framing from the layout.
</USER>[INST]

Table 5: The prompt template for the Visual Alignment Critic.

Visual Alignment Critic

[INST]<SYS>
Evaluate consistency among the input prompt, the spatial constraint set, the layout state, and the candidate image as the Visual Alignment Critic. Check object presence, object completeness, relation direction, support, scale, occlusion, visibility, camera framing, and appearance consistency. Return a structured validation record rather than a global score. Each failed item must identify the relevant object, relation, or camera field, describe the visual evidence, and provide a repair hint that can be converted into layout operations. Your output format is {validation record, passed constraints, failed constraints, repair feedback, memory update}.
</SYS>

<USER>
The input prompt is `{input prompt}`. The spatial constraint set is `{spatial constraint set}`. The current layout state is `{layout state}`. The candidate image is `{candidate image}`. The rendered layout is `{rendered layout}`. The instance masks are `{instance masks}`. The shared knowledge is `{shared knowledge}`. Produce a structured validation record and repair feedback. If all constraints are satisfied, mark the record as overall pass.
</USER>[INST]

Table 6: The prompt template for the Layout Harness.

Layout Harness
[INST]<SYS> Maintain execution discipline around the editable layout state as the Layout Harness. Convert validation results into rule based checks, select callable tools, update shared knowledge, and decide whether another round is needed. Use deterministic repair actions whenever a failure can be expressed as object translation, object scaling, support correction, occlusion separation, visibility restoration, focal length adjustment, global scale adjustment, or camera elevation adjustment. Preserve constraints that already pass validation when repairing failed constraints. Your output format is {repair plan, tool actions, stopping decision, shared knowledge update}. </SYS>
<USER> The validation record is `{validation record}`. The repair feedback is `{repair feedback}`. The current layout state is `{layout state}`. The rule constraints are `{rule constraints}`. The tool library is `{tool library}`. The previous actions are `{previous actions}`. The shared knowledge is `{shared knowledge}`. Produce an executable repair plan with target fields, action names, parameter values, priority, and a stopping decision for the next round. </USER>[INST]