ElasticFit: Fit-Aware 3D Object Insertion via VLM Reasoning and Generative Adaptation

Tzu-Hsin Hsieh
Delft University of Technology
Delft, Netherlands
thhsieh@tudelft.nl

Ricardo Marroquim
Delft University of Technology
Delft, Netherlands
R.Marroquim@tudelft.nl

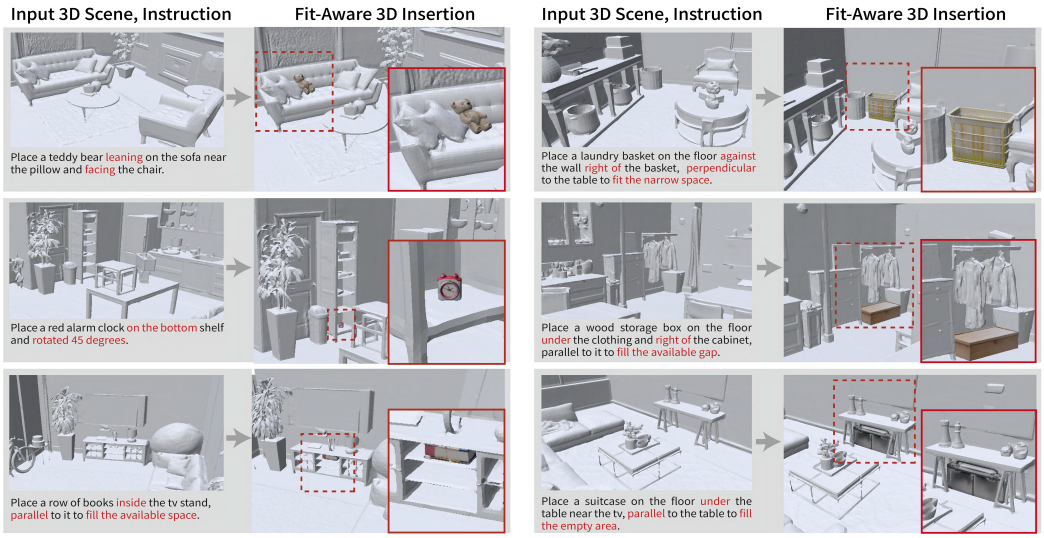

Figure 1: **ElasticFit** inserts generated 3D objects into pre-existing scenes from natural-language instructions. It combines VLM reasoning, generative adaptation, and geometry- and physics-aware refinement. Red text highlights key constraints; results are shown in color.

Abstract

Inserting objects into existing 3D scenes requires more than selecting a plausible location: the inserted object must also fit local geometry while preserving semantic intent and physical plausibility. Although recent Vision-Language Models (VLMs) and generative models enable semantic reasoning and visual content creation, they offer limited 3D grounding and geometric control when an inserted object must fit into constrained local spaces. We introduce **ElasticFit**, a VLM-guided framework for fit-aware object insertion centered on a novel scene-grounded representation. Given a language instruction and rendered scene observations, ElasticFit infers structured fitting cues that specify where the object should be grounded, what volume it should occupy, how it should be oriented, and its adaptation mode (rigid placement, uniform scaling, or elastic fitting). These cues convert high-level VLM reasoning into explicit 3D constraints that condition object generation and guide downstream geometric fitting. ElasticFit then generates a scene-conditioned object prior, reconstructs it in 3D, and refines the mesh through mode-specific fitting while enforcing collision avoidance, contact consistency, and physical grounding. In fixed-asset baseline comparisons, ElasticFit improves spatial relation success from 50.8% to 69.7% and support success from 48.3% to 91.7% over the strongest baseline, while providing novel support for generative "make-it-fit" insertions in complex scenarios. Project page: https://celine-hsieh.github.io/elasticfit/.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Inserting objects into existing 3D scenes is a fundamental capability for augmented and virtual reality, automated interior design, and embodied AI [1, 2, 3, 4]. Unlike isolated object generation or holistic scene synthesis, this task requires reasoning about an already occupied environment where a newly inserted object must satisfy multiple constraints. A successful insertion must not only pinpoint a location but also align with the user's language intent and fit the local geometry [5]. Furthermore, it must account for available free space, support surfaces, contact, and collision constraints. This makes object insertion a joint reasoning problem over semantic intent, scene context, and physical feasibility.

Recent progress in 3D placement and layout generation has demonstrated the potential of Vision-Language Models (VLMs) [6, 7] for high-level scene understanding and commonsense reasoning [8, 9, 10, 4]. These methods can often infer plausible object categories and coarse placement regions from scene context and user intent. However, most existing approaches primarily focus on selecting, retrieving, or positioning a given object, and remain limited when successful insertion requires the object itself to adapt to the local geometry of the target scene. In such cases, the challenge lies not only in deciding *where* to place an object, but also in determining *how* its shape, scale, and pose should adapt to fit constrained space while remaining semantically appropriate and physically feasible. As a result, existing methods are typically restricted to pose adjustment or global scaling, rather than adapting object geometry to local free space [11, 12]. This limitation becomes especially critical in cluttered or spatially constrained scenes.

These limitations suggest three key capabilities required for fit-aware insertion. **Geometry-grounded scene understanding.** Successful insertion requires reasoning over actual support surfaces and surrounding free space, rather than relying only on coarse abstractions such as bounding boxes [13, 14, 15, 3, 4]. ElasticFit grounds insertion in local scene geometry through scene-grounded target regions and fitting constraints. **Make-it-fit adaptation.** Most existing methods assume that a suitable object is already available, either retrieved from an asset repository [11, 4, 16] or provided for placement optimization [5, 2]. This assumption fails when the target region requires the object itself to adapt to surrounding spatial constraints. ElasticFit treats insertion as constrained fitting, enabling rigid alignment, uniform scaling, or elastic adaptation. **Physics-aware refinement.** Although prior methods often enforce geometric constraints, their final refinement is typically not designed to settle inserted objects under physical constraints [5, 4, 10, 3]. As a result, placements may still suffer from intersections, weak support, floating artifacts, or overhangs. ElasticFit refines the generated object with geometry- and physics-aware fitting objectives to improve contact, support, and collision compatibility.

We propose **ElasticFit**, a method for fit-aware 3D object insertion that converts language instructions and rendered scene observations into scene-grounded fitting constraints. Rather than directly predicting a final object transform, ElasticFit infers a scene-grounded fitting representation G, including the target region, orientation, support and contact requirements, and adaptation mode. This representation conditions object generation and mode-specific deformation-aware fitting, allowing the inserted object to be rigidly aligned, uniformly scaled, or elastically adapted to local spatial constraints. By coupling semantic grounding with geometry- and physics-aware optimization, ElasticFit produces insertions that are contextually appropriate, spatially aligned, and physically plausible.

Our contributions are summarized as follows:

- **Fit-aware object insertion formulation:** We formulate object insertion into pre-existing 3D scenes as fit-aware adaptation, jointly considering semantic intent, local geometric compatibility, and object-shape adaptation.
- **Scene-grounded fitting and adaptation:** We introduce a structured representation G that converts VLM-inferred cues into grounded constraints for object generation and rigid, uniform, or elastic fitting.
- **Experiments:** We evaluate ElasticFit on two task sets covering common placement relations and fit-critical constrained insertions. In fixed-asset baseline comparisons, ElasticFit improves spatial relation success from 50.8% to 69.7% and support success from 48.3% to 91.7% over the strongest baseline, and further demonstrates fit-aware insertion under pose, contact, and constrained-space requirements.

2 Related Work

Placement and Editing in Existing 3D Scenes. Recent work has explored language-guided object placement and editing in pre-existing 3D scenes, including placement in complex asset-based environments, placement in real scanned scenes, and language-guided scan or layout editing [5, 2, 17, 18]. FirePlace [5] is the closest prior work, as it grounds MLLM reasoning in fine-grained scene geometry and solves geometric constraints for placing a given 3D object in a pre-existing scene. PLACEIT3D [2] formulates language-guided object placement in real 3D scenes and introduces a benchmark for this task. ScanEdit [17] and EditRoom [18] address instruction-driven editing of existing 3D scans or room layouts through hierarchical reasoning or layout-level editing. Despite their differences, these methods primarily optimize where an object should be placed or how an existing scene should be rearranged, while the inserted asset is typically treated as fixed during placement or editing. ElasticFit studies a complementary make-it-fit setting: rather than only estimating the transformation of a fixed target asset, it generates and adapts the inserted object to a scene-grounded fitting region. This distinction is important in constrained local spaces, such as narrow gaps, shelves, or regions under furniture, where the object geometry itself may need to change to satisfy local spatial affordances. We provide a detailed design-level comparison with FirePlace in Appendix B.

Foundation-Model-Guided 3D Layout Planning. A broad line of work studies 3D layout generation and scene synthesis using learned priors, structured scene representations, and optimization over object arrangements [3, 19, 11, 20, 21, 22, 23]. Recent foundation-model-guided methods further leverage language or vision-language reasoning to improve semantic planning and controllability [24, 25, 16, 26, 15, 27, 28]. Representative examples include LayoutGPT [9], which predicts layouts from textual conditions; Holodeck [4], which generates spatial constraints and solves them through search-based construction; and LayoutVLM [10], which performs visually guided layout reasoning and optimization from multi-view inputs. These methods are closely related to ElasticFit's semantic-spatial reasoning and form the main baseline family. However, they mainly operate on layout abstractions such as object poses, bounding boxes, or symbolic constraints, which are insufficient for fine-grained support, contact, and make-it-fit adaptation in complex pre-existing scenes. In contrast, ElasticFit couples semantic-spatial reasoning with geometry-aware adaptation, enabling inserted objects to conform to local scene structure.

Generative Adaptation and Asset Creation. Recent advances in generative models have led to highly capable text-to-image models [29, 30, 31], which in turn have enabled strong 2D-to-3D and text-to-3D asset creation pipelines [32, 33, 34, 35]. Related work has also begun to explore generative object insertion in existing 3D representations, including neural radiance fields and 3D Gaussian scenes [36, 37, 38, 39, 40]. These methods demonstrate that generation can be used not only to create assets in isolation, but also to introduce new objects into scene context. However, they mainly formulate the problem as generative insertion or editing, rather than as scene-grounded adaptation for satisfying local geometric constraints. ElasticFit instead uses generation as part of a fit-aware insertion process, where the inserted asset is shaped jointly by semantic intent, local geometry, and contextual appearance.

3 Method

3.1 Overview

Given an existing 3D scene S and a natural-language placement prompt L, ElasticFit inserts a new object so that the result is semantically appropriate, geometrically compatible, and physically plausible. Rather than treating placement as rigid pose prediction, ElasticFit formulates object insertion as fitting an object to local scene geometry under semantic and physical constraints. As illustrated in Figure 2, ElasticFit realizes this formulation through three stages: **(1) Semantic-Spatial Grounding**, **(2) Scene-Conditioned Object Prior**, and **(3) Geometry- and Physics-Aware Fitting**.

The semantic-spatial grounding stage analyzes the scene and language input to infer a structured grounding representation G, formally defined in Sec. 3.2. This representation encodes the geometric placement constraints, including the support point, ghost-box center and dimensions, and object orientation, as well as a deformation mode represented by an adaptation scale vector $\mathbf{s} \in \mathbb{R}^3_+$ and a semantic object description. Conditioned on G, the scene-conditioned object prior stage generates an initial object hypothesis O_0 by producing a scene-conditioned 2D object image and lifting it

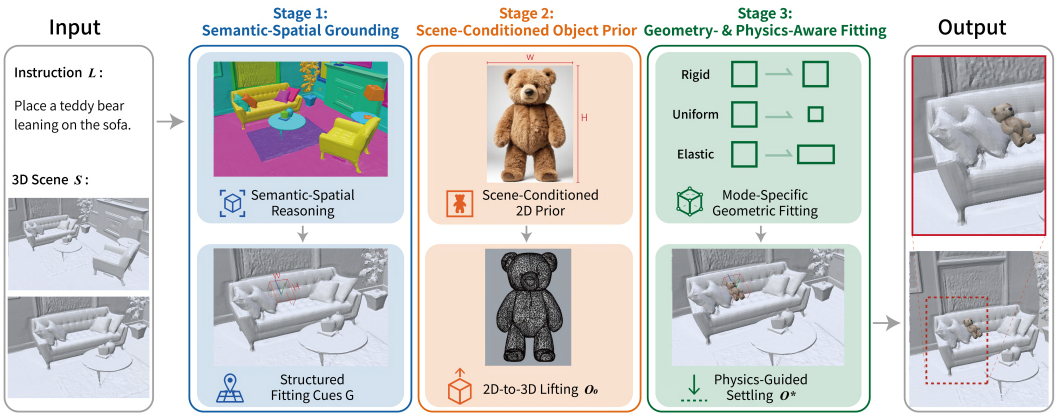

Figure 2: Overview of ElasticFit. Given a user instruction L and an existing 3D scene S, ElasticFit infers structured fitting cues G that encode the target region, orientation, constraints, and adaptation behavior. **Stage 1** performs semantic-spatial grounding, **Stage 2** generates a scene-conditioned object prior O_0. **Stage 3** refines it through geometry- and physics-aware fitting with rigid, uniform, or elastic adaptation. The result is a stable final insertion O^*.

into a 3D mesh. The geometry- and physics-aware fitting stage then refines O_0 according to the geometric constraints and deformation behavior encoded in G, followed by physics-based settling to obtain a stable final insertion O^*. This design separates high-level semantic-spatial reasoning from downstream geometric realization, while connecting them through the unified grounding representation G.

3.2 Semantic-Spatial Grounding [Stage 1]

Given a scene S and instruction L, the grounding stage maps VLM predictions into structured fitting cues G that specify object position, orientation, support, and adaptation.

Semantic-Spatial Reasoning. We construct a view-aligned scene representation for VLM-based reasoning by rendering the scene from the same viewpoint shown to users. Using this view, scene geometry, and dataset-provided semantic-instance annotations, we build two geometric buffers: an XYZ buffer storing per-pixel 3D coordinates and an ID buffer storing the visible semantic instance ID at each pixel. Based on the ID buffer, we generate a Set-of-Mark (SoM) [41] image that overlays visible object IDs on the current view and summarizes candidate anchor objects. We then query a VLM with the unannotated rendered scene view, the SoM image, and the instruction to predict a structured JSON containing (i) a reference anchor object `anchor_object_id`, (ii) a target placement pixel `target_uv` and support object `support_object_id`, (iii) optional orientation cues including `look_at_uv` and `top_uv`, (iv) snapping behavior, and (v) when adaptive generation is enabled, object-generation fields, canonical object scale, adaptation tags, and additional fitting controls.

Structured Fitting Cues. We transform the VLM prediction from image-space cues into explicit 3D fitting conditions. We first validate the predicted `target_uv` against the semantic ID buffer using `support_object_id`; if the predicted pixel does not lie on the intended support object, we replace it with the nearest visible pixel on that object. The corrected pixel is mapped through the XYZ buffer to obtain the 3D support point, which serves as the target surface point for grounding. Orientation is estimated from optional `look_at_uv` and `top_uv` cues, with pose-specific priors, yielding yaw and pitch angles in world coordinates. Using the support point, predicted object dimensions, and inferred orientation, we construct a 3D ghost box whose bottom-face center is placed at the support point. The ghost box is not a tight object bounding box, but an estimated maximum fitting volume that guides downstream object generation and adaptation. Stage 1 outputs a structured grounding representation

$$G = \{p_{\text{surf}},\, c_{\text{box}},\, \mathbf{b}_{\text{box}},\, R(\theta_{\text{yaw}}, \theta_{\text{pitch}}),\, m_{\text{def}},\, \mathbf{s},\, t_{\text{obj}}\}, \tag{1}$$

where $p_{\text{surf}} \in \mathbb{R}^3$ is the grounded support point, $c_{\text{box}} \in \mathbb{R}^3$ and $\mathbf{b}_{\text{box}} \in \mathbb{R}^3_+$ are the ghost-box center and dimensions, $R(\theta_{\text{yaw}}, \theta_{\text{pitch}}) \in SO(3)$ is the inferred object orientation, constructed in a fixed world coordinate frame with roll fixed to zero under an upright-object assumption, m_{def} selects

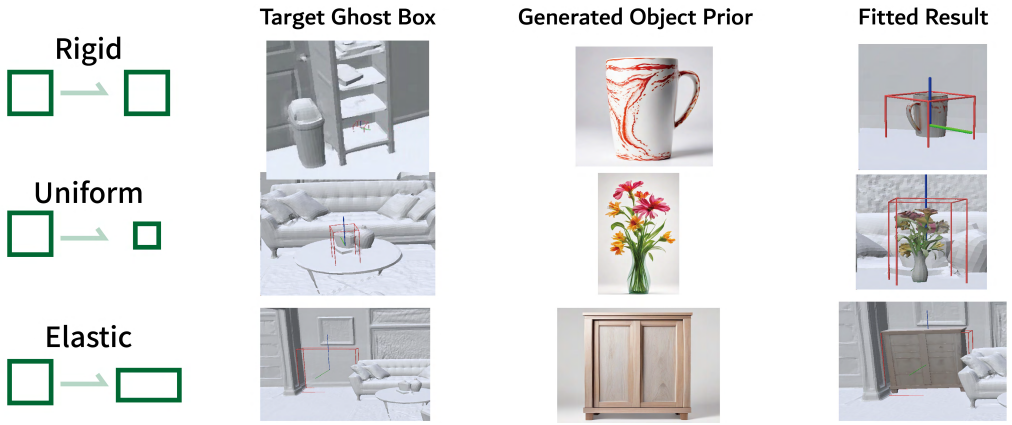

Figure 3: Visualization of the three adaptation modes in Stage 3. Each row shows the target ghost box, generated initial object, and fitted result. Rigid mode preserves geometry, uniform mode applies isotropic scaling, and elastic mode applies anisotropic deformation to fit the target region.

the downstream fitting strategy, $\mathbf{s} \in \mathbb{R}^3_+$ is the corresponding adaptation scale vector, and t_{obj} is the semantic object description used for generation. The deformation mode constrains $\mathbf{s}$ as rigid $\mathbf{s} = (1, 1, 1)$, uniform $\mathbf{s} = (\alpha, \alpha, \alpha)$, or elastic $\mathbf{s} = (s_x, s_y, s_z)$. Thus, G jointly encodes placement geometry, fitting strategy, and semantic conditioning for downstream generation and adaptation.

3.3 Scene-Conditioned Object Prior [Stage 2]

As shown in Figure 2, given the structured fitting cues G, the object-prior stage produces an initial object hypothesis O_0. It first synthesizes a scene-conditioned 2D object prior and then lifts it into 3D. We adopt a feedforward generation-and-lifting design so that O_0 can be produced efficiently from scene-derived conditioning, rather than through optimization-based 3D synthesis.

Scene-Conditioned 2D Prior. Given the grounding representation G and object prompt t_{obj} from Stage 1, we construct an initial visual hypothesis for the target object. Unlike prior placement methods that assume a fixed object or asset representation before placement [5, 9], ElasticFit first infers the target fitting region from the scene and uses the ghost box to define a scene-conditioned generation canvas. We employ an SDXL-style text-to-image generator [29] with a scene-conditioned canvas derived from the ghost box. The ghost-box dimensions are defined in the box-local frame as $\mathbf{b}_{\text{box}} = (b_x, b_y, b_z)$, corresponding to width, depth, and height. For front-view object generation, we set the image canvas aspect ratio according to the projected width-height proportion of the fitting region, i.e., $b_x : b_z$. This encourages the generated object to reflect the expected width-height proportions of the target region before 3D reconstruction. The object prompt t_{obj} specifies the desired object appearance, while the scene-conditioned canvas provides coarse structural guidance. As a result, the generated 2D object prior is biased toward the inferred fitting context rather than synthesized in isolation.

2D-to-3D Lifting. The generated 2D object image is lifted into an initial 3D object hypothesis O_0 using an image-to-3D model. In our implementation, we use TRELLIS [32] through a wrapper that reconstructs a 3D mesh from the generated image. This stage provides a scene-conditioned 3D prior that serves as a close-fit initialization. By preconditioning the object, we significantly reduce the burden of the subsequent physics-guided adaptation.

3.4 Geometry- and Physics-Aware Fitting [Stage 3]

While Stage 2 provides an initial prior O_0, the lifted mesh still requires precise alignment with the scene. Stage 3 refines the prior through a geometry- and physics-aware fitting procedure that enforces support, avoids collisions, and maximizes occupancy. Guided by the suggested deformation behavior m_{def}, we start by employing one of the three modes described below to scale the object within the bounding box.

Mode-Specific Geometric Fitting. **Rigid** mode is used for objects whose manufactured or mechanical proportions should remain fixed. We preserve the object geometry and solve for a grounded, collision-aware pose using a broad-search and narrow-refine procedure: coarse proxy-box collision filtering is followed by mesh-level collision and support refinement.

Uniform mode is used for objects whose recognizable proportions should be preserved. We optimize a single global scale factor together with translation, balancing collision penalties, box constraints, support/gravity terms, and scale maximization so that the object fits the target region without changing its aspect ratio.

Elastic mode is used for objects that can be non-uniformly adjusted to fit constrained or gap-like regions. The object is anisotropically scaled along the box-local axes toward the ghost-box dimensions, while nearby scene samples are used to penalize collisions and maintain support on the target surface. This enables non-uniform adaptation to the available space without assuming that the entire ghost box is collision-free.

Appendix C.3 further provides implementation details, and Algorithm 1 summarizes the overall fitting procedure.

Physics-Guided Settling. After mode-specific fitting, we apply a lightweight rigid-body simulation in PyBullet to allow the adapted object to settle naturally in the existing scene. This step improves physical realism by refining the already fitted object pose under gravity and contact dynamics. We model the scene geometry as a static concave triangle mesh and the adapted object as a dynamic rigid body, simulating gravity, collision/contact response, friction, and damping. To avoid unstable contact initialization, we start the object from a slightly lifted, non-penetrating pose near the fitted placement before running the short settling simulation. We then perform a final refinement based on nearby upward-facing support points, applying only a conservative correction to eliminate small numerical gaps or residual contact errors. The final output is a fitted insertion O^* that respects the semantic intent of G while resting naturally on the supporting scene geometry.

4 Experiments

4.1 Experimental Setup

Dataset and Tasks. We evaluate ElasticFit on single-object insertion tasks, where a target object must be inserted into an existing 3D scene according to a natural-language semantic-spatial instruction while satisfying geometric and physical plausibility. The evaluation is designed to measure both standard placement quality and fit-aware adaptation under local scene constraints. We use both ReplicaCAD [42] and Replica [43] for cross-dataset validation: ReplicaCAD provides clean CAD-like scenes for testing geometric soundness, while Replica provides noisy reconstructed scans for testing robustness. Cross-dataset results are reported in Sec. 4.2.

Task Construction. We design two task sets for evaluating single-object insertion. The **Common Placement Set** (S001–S060) contains 60 standard insertion tasks covering common spatial relations, including `near/far`, `on_top_of`, `left/right/front/behind`, `face_to`, `aligned_with`, and `against_wall`. This set evaluates semantic-spatial grounding and placement quality under common relations, and is used for cross-dataset validation and baseline comparison. For experiments on this set, all methods are evaluated using the same prompts, scenes, and target object categories.

The **Fit-Critical Set** (F001–F060) contains 60 additional tasks for evaluating fit-aware insertion under challenging placement and fitting conditions. These tasks use the same spatial-relation vocabulary as the Common Placement Set, but add pose, contact, and local geometric adaptation requirements. We divide this set into two subsets: **pose-and-contact tasks**, which combine common placement relations with user-specified rotations, leaning, or lying configurations, and **gap-fitting tasks**, which require insertion into gaps, corners, wall-side spaces, or under/inside furniture.

Baselines. We compare against three representative language- or vision-language-guided layout methods: **LayoutGPT**[9], **Holodeck**[4], and **LayoutVLM**[10]. We adapt them to our single-object insertion setting by keeping the input scene unchanged and allowing each method to predict or optimize only the inserted target object. In the baseline comparison, all methods receive the same scene, natural-language prompt, and fixed target object asset for each task. For clarity, we refer to these adapted implementations by their original method names in the remaining experiments.

Table 1: Cross-dataset validation of ElasticFit on matched placement tasks.

Dataset	SRS ↑	CFR ↑	SupR ↑	CPS ↑
ReplicaCAD	**70.4%**	91.4%	**92.7%**	**4.58**
Replica	69.7%	**91.7%**	91.7%	4.18

Table 2: Baseline comparison on the Common Placement Set.

Method	SRS ↑	CFR ↑	SupR ↑	CPS ↑
LayoutGPT	50.8%	73.3%	48.3%	3.72
Holodeck	41.5%	**95.0%**	41.7%	3.63
LayoutVLM	49.0%	93.0%	43.9%	3.82
ElasticFit	**69.7%**	91.7%	**91.7%**	**4.18**

Note. SRS = Spatial Relation Success Rate, CFR = Collision-Free Rate, SupR = Support Success Rate, and CPS = Contextual Plausibility Score. Higher is better for all metrics. Scores are reported in percent except CPS.

LayoutGPT predicts the target object placement from the prompt and scene context. Holodeck uses a constraint-based search procedure to satisfy spatial relations between the target object and existing scene objects. LayoutVLM combines vision-language reasoning with differentiable constraint optimization to predict and refine the object placement.

We evaluate the adapted baselines on the **Common Placement Set**, where all methods use fixed target assets for fair comparison. For baseline comparison, **ElasticFit** is evaluated in a placement-only setting: scene-conditioned object prior generation and deformation-aware adaptation are disabled, while semantic-spatial grounding and geometry- and physics-aware refinement are retained. This isolates placement and grounding quality from object generation and shape adaptation. The complete **ElasticFit** further includes scene-conditioned object prior generation, 3D lifting, and deformation-aware fitting, and is evaluated separately on the **Fit-Critical Set** in Sec. 4.4. All LLM/VLM-based reasoning in ElasticFit and the adapted baselines uses the same backbone, Gemini-3.1 Pro Preview.

Evaluation Metrics. We evaluate each method using four metrics. The first three are automatic geometry-based percentages adapted from common evaluation dimensions in 3D layout generation and object placement: spatial constraint satisfaction, collision avoidance, and physical support [5, 2, 10]. *Spatial Relation Success Rate (SRS)* measures the percentage of ground-truth spatial predicates satisfied by the final insertion using checks on pose, size, orientation, and reference geometry. *Collision-Free Rate (CFR)* measures the percentage of insertions that avoid unacceptable interpenetration, estimated by counting sampled scene points against the fitted object under a fixed tolerance. *Support Success Rate (SupR)* measures the percentage of insertions supported by the annotated surface, based on object-bottom gap and footprint support coverage. *Contextual Plausibility Score (CPS)* follows recent model-assisted evaluation protocols [10, 5], scoring anonymized user-view screenshots and the original prompt with GPT-5.4-mini on a 1–5 scale.

4.2 Cross-Dataset Validation

Protocol. We evaluate the robustness of ElasticFit across clean and reconstructed indoor scenes by comparing ReplicaCAD and Replica on a matched subset of the Common Placement Set. Each task uses the same prompt type, target object category, and evaluator across the two datasets. To isolate the effect of scene geometry, we use the same placement-only variant of ElasticFit as in the baseline comparison described in Sec. 4.1.

Results. As shown in Table 1, ElasticFit performs comparably across the two datasets, effectively bridging the gap between synthetic environments and reconstructed scans. ReplicaCAD is slightly better in support success and contextual plausibility due to cleaner CAD-like geometry, while Replica remains close despite reconstruction artifacts and inherent noise from scanned environments. This indicates robust grounding and refinement across both clean and reconstructed scenes. We conduct our experiments on Replica as it presents typical challenges of real-world AR/VR applications.

4.3 Baseline Comparison

Quantitative Results. We evaluate the adapted baselines and ElasticFit on the Common Placement Set using matched prompts, scenes, target object categories, and the same fixed target assets as the baselines. As shown in Table 2, ElasticFit achieves the best spatial relation success, support success, and contextual plausibility. It improves SRS from the strongest baseline's 50.8% to 69.7%, substantially increases SupR from the best baseline's 48.3% to 91.7%, and obtains the highest CPS of 4.18 compared with the best baseline score of 3.82.

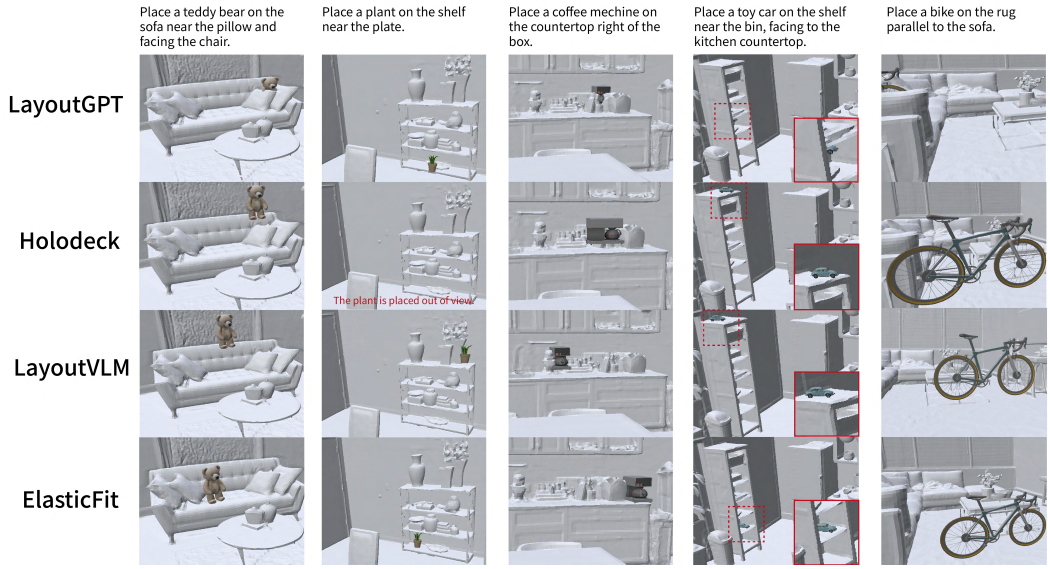

Figure 4: Qualitative comparison on representative Common Placement Set tasks. Baselines often struggle with physical refinement, ambiguous references, viewpoint-dependent relations, or irregular support geometry. ElasticFit produces more spatially and physically plausible placements.

While Holodeck and LayoutVLM obtain marginally higher CFR, this reflects a conservative strategy that places objects far from the cluttered scene geometry. Consequently, they fail to satisfy proximity and contact constraints, as evidenced by their significantly lower SRS and SupR. ElasticFit grounds objects closer to the intended scene regions and support surfaces, improving SRS and SupR while occasionally introducing minor residual intersections on irregular geometry or tightly constrained placements. Overall, ElasticFit maintains a high CFR while producing insertions that better satisfy spatial relations, support contact, and contextual plausibility.

Qualitative Results. Figure 4 shows representative examples from the Common Placement Set. LayoutGPT often predicts plausible coarse locations, but lacks physical refinement and can produce penetrating or unsupported placements. Holodeck improves collision avoidance, but can select incorrect references in cluttered scenes. LayoutVLM reduces ambiguity with additional views and facing cues, but still suffers from imprecise grounding and high runtime cost; runtime comparisons are provided in Appendix F. Holodeck and LayoutVLM rely on bounding-box-based refinement, which can produce floating artifacts on irregular support geometry.

A common issue across baselines is handling viewpoint-dependent spatial relations, such as *in front of*, *behind*, *left of*, and *right of*. Since users may rotate the scene during placement, these relations should follow the user's view rather than the reference object's intrinsic orientation or a fixed world frame. For example, in the coffee-machine example, a baseline may place it on the box's object-centric right side, which appears on the left from the user's current viewpoint. ElasticFit aligns VLM reasoning with the user's current scene view and applies geometry- and physics-aware refinement, producing placements that better satisfy spatial relations, support contact, and collision avoidance.

4.4 Fit-Aware Insertion Analysis

Quantitative Results. Table 3 summarizes the performance of the complete **ElasticFit** method on the Fit-Critical Set and its two subsets. Overall, ElasticFit achieves 60.8% SRS, 71.7% CFR, and 83.3% SupR. The higher performance on pose-and-contact tasks shows that ElasticFit is effective when the main challenge is to satisfy explicit pose,

Table 3: Performance on the Fit-Critical Set.

Subset	SRS ↑	CFR ↑	SupR ↑	CPS ↑
All tasks	60.8%	71.7%	83.3%	4.57
Pose-and-contact tasks	73.3%	76.7%	80.0%	4.50
Gap-fitting tasks	48.3%	66.7%	86.7%	4.63

orientation, support, or contact constraints. In this subset, ElasticFit achieves 73.3% SRS and 76.7% CFR, suggesting that semantic-spatial grounding and physics-guided refinement can reliably produce physically plausible insertions.

In contrast, gap-fitting tasks are more challenging, with SRS decreasing to 48.3% and CFR to 66.7%. This is expected, as insertions in highly constrained spaces are sensitive to minor geometric misalignments. Nevertheless, SupR remains high at 86.7%, and CPS reaches 4.63, indicating that many results are still well-supported and contextually plausible. This suggests that strict geometric-relation scores on constrained-space tasks should be interpreted together with contextual plausibility; we provide a complementary analysis in Appendix A.1.

Qualitative Results. As shown in Figure 1, ElasticFit produces fit-aware insertions that conform to local scene geometry beyond rigid placement.

For pose-and-contact tasks, ElasticFit supports pitch-angle adjustments for *leaning* and *lying* configurations, enabling objects such as a teddy bear to tilt or lie naturally on a sofa. It also supports user-defined rotations, allowing users to specify desired object orientations more explicitly.

For gap-fitting tasks, ElasticFit adapts object geometry by elongating, compressing, or scaling generated objects to occupy narrow regions beside furniture, inside shelves, or under existing objects. These examples illustrate the key motivation of ElasticFit: rather than only searching for a valid pose for a fixed asset, ElasticFit adapts the inserted object to better match local spatial affordances.

4.5 Ablation Studies

Table 4 evaluates the contribution of geometry- and physics-aware fitting on gap-fitting tasks. It uses a separate controlled run with a different input view from Table 3, while sharing the same view across all variants. The second row removes only final settling, while the third removes mode-specific fitting and uses the initial ghost-box fit. We also report penetrating vertex ratio, the fraction of object vertices estimated to penetrate the scene; lower is better. Removing only the final physics-based settling leads to a decrease in SupR from 83.3% to 80.0%, and an increase in penetrating vertex ratio from 1.08% to 1.45%. This confirms that while geometric fitting identifies the correct region, the dynamic simulation is necessary to resolve small-scale issues.

Table 4: Ablation study on gap-fitting tasks.

Setting	SRS ↑	CFR ↑	SupR ↑	CPS ↑	Pen. ↓
ElasticFit	**47.8%**	**76.7%**	**83.3%**	**4.63**	**1.08%**
w/o physics settle	46.7%	**76.7%**	80.0%	4.60	1.45%
w/o geom.-phys. fitting	**47.8%**	36.7%	**83.3%**	4.23	6.09%

In contrast, removing geometry- and physics-aware fitting in Stage 3 substantially degrades geometric feasibility. Although SRS remains similar and SupR is unchanged, CFR drops from 76.7% to 36.7%, and penetrating vertex ratio increases from 1.08% to 6.09%. This indicates that the initial ghost-box fit can often identify a semantically reasonable target region and supported placement, but is insufficient for collision-free insertion in constrained spaces. The lower CPS score further suggests that severe geometric misfit can also reduce the perceived plausibility of the final result.

5 Conclusion and Discussion

ElasticFit formulates 3D object insertion as fit-aware adaptation, grounding generated objects in pre-existing scenes while adapting them to local geometric constraints. Rather than directly predicting rigid placements, ElasticFit infers scene-grounded fitting cues that connect semantic intent, object generation, and deformation-aware fitting. By combining view-aligned VLM grounding, scene-conditioned object priors, and geometry- and physics-aware fitting, ElasticFit improves spatial relation and support success over language- and vision-language-guided baselines, while enabling constrained insertions beyond placement-only methods. These results highlight generative adaptation as a bridge between high-level semantic instructions and physically grounded 3D scene realization.

Despite these gains, our current elastic mode relies on anisotropic scaling. While effective for many categories of objects, true non-rigid deformation (e.g., cloth draping over a chair or pillow compressed into a corner) remains an open challenge. Moreover, because of proximity constraints, sometimes the support constraint is satisfied, but objects are left partially hanging over a surface edge. Incorporating explicit containment constraints could ensure that objects are fully within the supporting surface.

Future work could also explore multi-object insertion, where reasoning should account for multiple simultaneous object placements.

References

[1] Carl-Philipp Hellmuth, Miroslav Bachinski, and Jörg Müller. Interaction techniques for 3d-positioning objects in mobile augmented reality. In *Proceedings of the 2021 International Conference on Multimodal Interaction*, pages 604–612, 2021.

[2] Ahmed Abdelreheem, Filippo Aleotti, Jamie Watson, Zawar Qureshi, Abdelrahman Eldesokey, Peter Wonka, Gabriel Brostow, Sara Vicente, and Guillermo Garcia-Hernando. Placeit3d: Language-guided object placement in real 3d scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6645–6655, 2025.

[3] 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.

[4] 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*, pages 16227–16237, 2024.

[5] Ian Huang, Yanan Bao, Karen Truong, Howard Zhou, Cordelia Schmid, Leonidas Guibas, and Alireza Fathi. Fireplace: Geometric refinements of llm common sense reasoning for 3d object placement. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13466–13476, 2025.

[6] OpenAI. Gpt-4v(ision) system card. `https://cdn.openai.com/papers/GPTV_System_Card.pdf`, 2023. Accessed: 2026-04-21.

[7] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[8] Rao Fu, Jingyu Liu, Xilun Chen, Yixin Nie, and Wenhan Xiong. Scene-llm: Extending language model for 3d visual reasoning. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2195–2206. IEEE, 2025.

[9] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, 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.

[10] 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*, pages 29469–29478, 2025.

[11] 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*, pages 20507–20518, 2024.

[12] Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942, 2021.

[13] Ian Huang, Guandao Yang, and Leonidas Guibas. Blenderalchemy: Editing 3d graphics with vision-language models. In *European Conference on Computer Vision*, pages 297–314. Springer, 2024.

[14] Rio Aguina-Kang, Maxim Gumin, Do Heon Han, Stewart Morris, Seung Jean Yoo, Aditya Ganeshan, R Kenny Jones, Qiuhong Anna Wei, Kailiang Fu, and Daniel Ritchie. Open-universe indoor scene generation using llm program synthesis and uncurated object databases. *arXiv preprint arXiv:2403.09675*, 2024.

[15] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *Forty-first International Conference on Machine Learning*, 2024.

[16] Başak Melis Öcal, Maxim Tatarchenko, Sezer Karaoğlu, and Theo Gevers. Sceneteller: Language-to-3d scene generation. In *European Conference on Computer Vision*, pages 362–378. Springer, 2024.

[17] Mohamed El Amine Boudjoghra, Ivan Laptev, and Angela Dai. Scanedit: Hierarchically-guided functional 3d scan editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 27105–27115, 2025.

[18] Kaizhi Zheng, Xiaotong Chen, Xuehai He, Jing Gu, Linjie Li, Zhengyuan Yang, Kevin Lin, Jianfeng Wang, Lijuan Wang, and Xin Eric Wang. Editroom: Llm-parameterized graph diffusion for composable 3d room layout editing. *arXiv preprint arXiv:2410.12836*, 2024.

[19] Xinpeng Wang, Chandan Yeshwanth, and Matthias Nießner. Sceneformer: Indoor scene generation with transformers. In *2021 International conference on 3D vision (3DV)*, pages 106–115. IEEE, 2021.

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

[21] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019.

[22] Guangyao Zhai, Evin Pınar Örnek, Shun-Cheng Wu, Yan Di, Federico Tombari, Nassir Navab, and Benjamin Busam. Commonscenes: Generating commonsense 3d indoor scenes with scene graph diffusion. *Advances in Neural Information Processing Systems*, 36:30026–30038, 2023.

[23] 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*, pages 16262–16272, 2024.

[24] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3d-gpt: Procedural 3d modeling with large language models. In *2025 International Conference on 3D Vision (3DV)*, pages 1253–1263. IEEE, 2025.

[25] Mengqi Zhou, Yuxi Wang, Jun Hou, Shougao Zhang, Yiwei Li, Chuanchen Luo, Junran Peng, and Zhaoxiang Zhang. Scenex: Procedural controllable large-scale scene generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 10806–10814, 2025.

[26] Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. I-design: Personalized llm interior designer. In *European Conference on Computer Vision*, pages 217–234. Springer, 2024.

[27] Hou In Ivan Tam, Hou In Derek Pun, Austin T Wang, Angel X Chang, and Manolis Savva. Scenemotifcoder: Example-driven visual program learning for generating 3d object arrangements. *arXiv preprint arXiv:2408.02211*, 2024.

[28] Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Zhaoshuo Li. Scenethesis: A language and vision agentic framework for 3d scene generation. *arXiv preprint arXiv:2505.02836*, 2025.

[29] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[30] Youngwan Lee, Kwanyong Park, Yoorhim Cho, Yong-Ju Lee, and Sung Ju Hwang. Koala: Empirical lessons toward memory-efficient and fast diffusion models for text-to-image synthesis. *Advances in Neural Information Processing Systems*, 37:51597–51633, 2024.

[31] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[32] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 21469–21480, 2025.

[33] Jiale Xu, Weihao Cheng, Yiming Gao, Xintao Wang, Shenghua Gao, and Ying Shan. Instantmesh: Efficient 3d mesh generation from a single image with sparse-view large reconstruction models. *arXiv preprint arXiv:2404.07191*, 2024.

[34] Zibo Zhao, Zeqiang Lai, Qiaozhe Lin, Yifei Feng, Xianghui Yang, Huiwen Shi, Junta Wu, Zebin He, Xinzhou Wang, et al. Hunyuan3d 2.0: Scaling diffusion models for high resolution textured 3d assets generation, 2025.

[35] Dmitry Tochilkin, David Pankratz, Zexiang Liu, Zixuan Huang, Adam Letts, Yangguang Li, Ding Liang, Christian Laforte, Varun Jampani, and Yan-Pei Cao. Triposr: Fast 3d object reconstruction from a single image. *arXiv preprint arXiv:2403.02151*, 2024.

[36] Mohamad Shahbazi, Liesbeth Claessens, Michael Niemeyer, Edo Collins, Alessio Tonioni, Luc Van Gool, and Federico Tombari. Inserf: Text-driven generative object insertion in neural 3d scenes. *arXiv preprint arXiv:2401.05335*, 2024.

[37] Chenxi Li, Weijie Wang, Qiang Li, Nicu Sebe, Bruno Lepri, and Weizhi Nie. Freeinsert: Disentangled text-guided object insertion in 3d gaussian scene without spatial priors. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 10915–10924, 2025.

[38] Vivek Madhavaram, Shivangana Rawat, Chaitanya Devaguptapu, Charu Sharma, and Manohar Kaul. Towards a training free approach for 3d scene editing. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2890–2899. IEEE, 2025.

[39] Edward Bartrum, Thu Nguyen-Phuoc, Chris Xie, Zhengqin Li, Numair Khan, Armen Avetisyan, Douglas Lanman, and Lei Xiao. Replaceanything3d: Text-guided object replacement in 3d scenes with compositional scene representations. *Advances in Neural Information Processing Systems*, 37:48568–48598, 2024.

[40] Hanyuan Xiao, Yingshu Chen, Huajian Huang, Haolin Xiong, Jing Yang, Pratusha Prasad, and Yajie Zhao. Localized gaussian splatting editing with contextual awareness. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 5207–5217. IEEE, 2025.

[41] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.

[42] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in neural information processing systems*, 34:251–266, 2021.

[43] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[44] Sketchfab: The leading platform for 3d and ar on the web. `https://sketchfab.com/`. Accessed: 2026-04-30.

Appendix Overview

This appendix provides additional details, analyses, and qualitative results for ElasticFit.

In Appendix A, we discuss system limitations and representative failure cases, including directional errors, deformation artifacts, and concave support ambiguities.

In Appendix B, we provide a design-level comparison with closely related object placement and layout methods, with a particular focus on FirePlace as the closest prior work to our setting.

In Appendix C, we provide implementation details for the three stages of ElasticFit: semantic-spatial grounding, scene-conditioned object prior generation, and geometry- and physics-aware fitting. We also include the structured VLM output schema, prompt organization, fitting objective, and the overall Stage 3 fitting algorithm.

In Appendix D, we describe the task sets and prompt construction used in our evaluation, including the Common Placement Set and the Fit-Critical Set.

In Appendix E, we define the evaluation metrics used in the main paper, including Spatial Relation Success Rate, Collision-Free Rate, Support Success Rate, and Contextual Plausibility Score.

In Appendix F, we describe how the baseline methods are adapted to the single-object insertion setting and provide additional runtime and implementation details for comparison.

In Appendix G, we show additional qualitative examples for baseline comparisons and fit-aware insertion results.

In Appendix H, we document the third-party 3D assets used in our experiments and their licenses.

A System Limitations

Evaluation Scope. Our task suite was developed alongside the method, so the reported results should be interpreted within this controlled setting. We do not quantify repeated-run variability. Our experiments use dataset-provided semantic instance annotations, and robustness to errors in predicted annotations remains untested.

A.1 Complementary Interpretation of SRS and CPS on Gap-Fitting Tasks

The gap-fitting subset contains 30 tasks and has a lower SRS of 48.3%. This result should be interpreted together with CPS, since the two metrics measure different aspects of insertion quality. SRS evaluates whether specified geometric predicates are strictly satisfied, while CPS evaluates the overall contextual plausibility of the rendered result. Among the 25 gap-fitting cases with $\mathrm{SRS} < 1$, 24 cases still receive a high CPS score ($\mathrm{CPS} \geq 4$). This suggests that many SRS-unsatisfied cases correspond to strict predicate-level deviations, especially orientation-sensitive checks in constrained spaces, rather than clearly implausible insertions.

Table 5: Diagnostic breakdown of SRS-unsatisfied cases on the gap-fitting subset. The gap-fitting subset contains 30 tasks; this table analyzes the 25 cases with $\mathrm{SRS} < 1$. Categories are diagnostic and not mutually exclusive.

Diagnostic category	**Count / 25**	**Explanation**
SRS not satisfied, but visually plausible ($\mathrm{CPS} \geq 4$)	24 (96.0%)	These cases do not fully satisfy strict geometric predicates, but their rendered placements remain contextually plausible according to CPS.
`parallel_to` check not satisfied	19 (76.0%)	These cases contain an unsatisfied `parallel_to` relation, indicating that orientation constraints are a common source of strict SRS misses in constrained regions.

A.2 Representative Failure Cases

Figure 5 shows representative failure cases of ElasticFit, highlighting limitations in directional fitting, deformation-aware adaptation, and concave support reasoning.

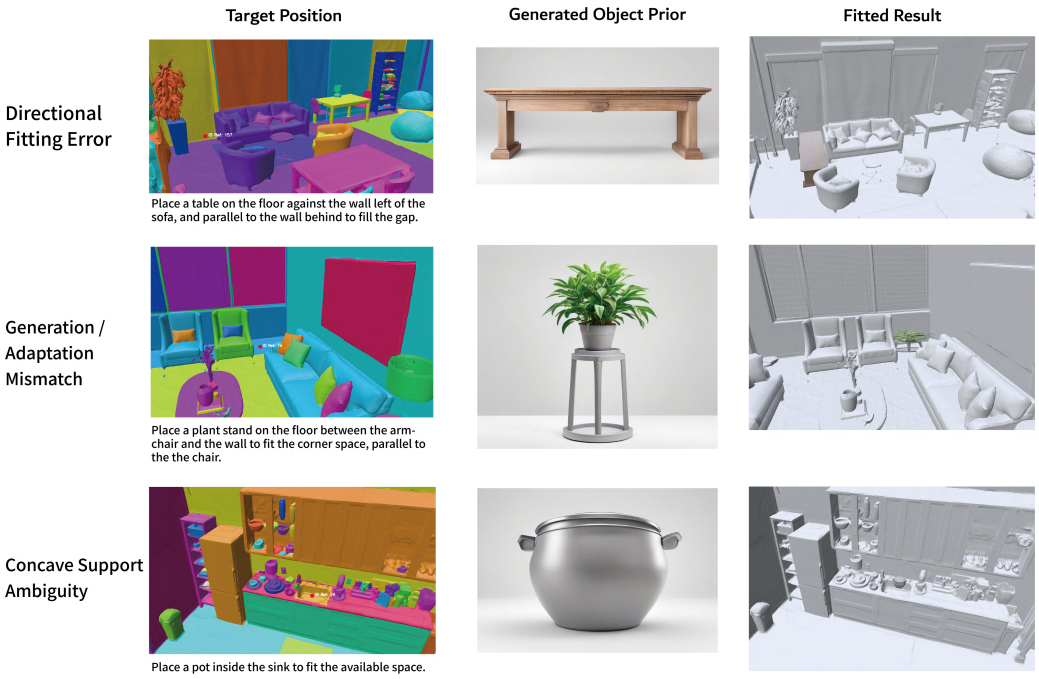

Figure 5: Representative failure cases of ElasticFit. Top: directional fitting errors arise from ambiguous reference geometry or fill directions. Middle: generation–adaptation mismatch occurs when the object prior and subsequent deformation do not match the target fitting region. Bottom: concave support ambiguity occurs in sink-like geometry, where rim or surface support can be mistaken for the intended interior placement.

Table 6: Failure attribution on the Fit-Critical Set. Each failed task is assigned to its dominant failure source. Percentages are computed over all 60 Fit-Critical tasks.

Failure source	**Count / 60**	**Description**
Directional fitting error	2 (3.3%)	Incorrect yaw, pitch, alignment, or gap-fill direction.
Generation / adaptation mismatch	6 (10.0%)	Object prior or deformation mismatches the target fitting region.
Concave support ambiguity	1 (1.7%)	Sink-like concave geometry can confuse inside placement with rim or surface support.

Table 6 summarizes the dominant visible failure sources observed in the Fit-Critical Set. We discuss representative cases below.

First, directional fitting errors can occur even when the target support point is semantically corrected. ElasticFit verifies whether the predicted target point lies on the intended support object using the semantic ID buffer and, when necessary, corrects it to a valid point on the support object. However, this correction only ensures that the point belongs to the intended semantic object; it does not guarantee that the point lies on the intended support surface, such as the top of a shelf rather than a nearby side surface. This makes the fitting direction sensitive to viewpoint and surface visibility, and may lead to floating or offset placements when the visible image evidence is ambiguous. Moreover, even when the support point is corrected back to the intended support surface, the correction does not fully resolve the orientation or fill direction of the target ghost box. For example, in the first row, the

instruction asks for a table to fill the gap along the wall. Although the support point is corrected back to the floor, the current orientation heuristic estimates parallel or perpendicular alignment from the long and short axes of the reference object. When the reference object has similar horizontal extents or ambiguous dominant axes, the inferred fill direction can be incorrect. As a result, the fitted object is placed in a semantically related region but fails to occupy the intended wall-side gap.

Second, generation–adaptation mismatch may arise from the combination of soft object-prior control and elastic fitting. ElasticFit uses the target ghost-box canvas and coarse topology cues, such as *wide horizontal* or *tall vertical*, to encourage the generated object prior to match the intended aspect ratio. However, these cues do not impose hard geometric constraints on the generative model, and the generated prior may still underfill the target canvas or follow shape biases from the model's training distribution. When this prior is subsequently adapted through axis-dependent scaling in the local frame of the target ghost box, elastic fitting can amplify the mismatch, producing visually implausible shapes for objects with strict semantic or structural proportions. As shown in the second row, the generated plant prior does not fully match the intended elongated fitting region, and the fitted result exhibits an unnatural deformation. This suggests the need for stronger spatial conditioning, such as ControlNet-style or mask/box-conditioned generation, together with a deformation-tolerance prior that determines which object categories can safely undergo elastic adaptation.

Third, concave support ambiguity can occur in sink-like geometry, where the intended placement is *inside* a basin rather than *on* a surface. In the sink example in the third row, support and collision refinement can treat the rim or nearby upward-facing geometry as the effective support instead of the basin interior. This may lift the object away from the intended interior region, causing the final placement to appear floating or insufficiently contained.

These cases suggest several directions for future work, including stronger spatial conditioning for object-prior generation, more robust orientation reasoning for ambiguous reference geometry, category-aware deformation constraints, and iterative verification of the fitted result against the original instruction and local scene geometry.

A.3 Broader Impacts

ElasticFit may support AR/VR authoring, interior design, simulation, and embodied-AI scene construction by reducing manual effort for placing and adapting objects in existing 3D scenes. Potential risks include generating misleading or unrealistic scene edits, producing physically unsafe suggestions if used in real-world design workflows without verification, and inheriting biases or artifacts from external VLMs and generative models. The system is intended for research use and should not be used for safety-critical physical planning without additional verification.

B Design-Level Comparison with Closely Related Work

This section provides a design-level comparison between ElasticFit and closely related object placement methods, with a particular focus on FirePlace [5], which is the closest prior work to our setting. FirePlace grounds multimodal language-model reasoning in fine-grained scene geometry and refines the placement of a given 3D object using geometric constraints. However, it assumes that the inserted object is a pre-existing fixed asset. In contrast, ElasticFit studies a complementary fit-aware insertion setting, where the system must reason not only about where the object should be placed, but also about how the object should be generated or adapted to satisfy local scene constraints.

Table 7 summarizes the key design differences. FirePlace is closer to ElasticFit than layout-level methods such as LayoutGPT, Holodeck, and LayoutVLM because it grounds multimodal reasoning in fine-grained scene geometry and refines object placement using geometric constraints. However, FirePlace assumes a pre-existing fixed asset and optimizes its placement, while ElasticFit targets a generative "make-it-fit" setting in which the object may also need to be generated or adapted to satisfy local scene constraints. Since our Fit-Critical Set is specifically designed for cases where fixed assets may not satisfy the available space, a direct numerical comparison on this set would evaluate FirePlace outside its intended setting. We therefore treat FirePlace as a design reference rather than a direct full-pipeline baseline.

ElasticFit extends this line of work by introducing a scene-grounded fitting representation that connects VLM reasoning to both generative object creation and downstream geometric fitting.

Table 7: Design-level comparison between FirePlace and ElasticFit.

Aspect	**FirePlace**	**ElasticFit**
Target object	Pre-existing fixed asset	Generated or fixed object
Main objective	Place a given object	Generative "make-it-fit" insertion
Adaptation	Pose-level placement	Rigid, uniform, or elastic fitting
Constrained spaces	Limited by fixed-asset geometry	Designed for gaps, corners, wall-side, under/inside spaces
Grounding	Fine-grained geometric placement constraints	Grounded fitting representation G with support, ghost box, orientation, mode, scale, and prompt
Physical refinement	Geometry-aware collision/support refinement	Geometry-aware fitting plus physics-guided settling

Table 8: Design-level comparison of object insertion and layout methods.

Method	**Target Object**	**Placement Strategy**	**Adaptation**	**Constrained-Space Insertion**
LayoutGPT	Fixed / retrieved	Layout pose direct prediction	Rigid only	No
Holodeck	Fixed / retrieved	Layout constraints / box solver	Rigid only	No
LayoutVLM	Fixed / retrieved	Visual layout / box solver	Rigid only	No
FirePlace	Fixed asset	Fine-grained geom. constraints	Rigid only	Fixed-asset only
ElasticFit	**Generated / fixed**	**Grounded rep. G + geom.-phys. fitting**	**Rigid / Uniform / Elastic**	**Yes**

Instead of first selecting a fixed mesh and optimizing only its pose, ElasticFit grounds the intended insertion region through a support point, ghost box, orientation, deformation mode, and object prompt. This representation allows the object prior to be generated or selected according to the target scene context, and then fitted using rigid placement, uniform scaling, or elastic adaptation. This design enables insertion in cases where the object must fit into a narrow gap, occupy a wall-side region, or adapt to a constrained space under or inside existing furniture.

Task compatibility with FirePlace. Although the Fit-Critical Set is beyond FirePlace's fixed-asset setting, parts of the Common Placement Set overlap with its intended scope. In particular, the distance and alignment subsets, comprising 24 of the 60 Common Placement tasks, involve standard fixed-asset placement on support surfaces and do not require object generation or shape adaptation. These tasks represent the closest overlap between FirePlace and ElasticFit.

Under our placement-only variant, ElasticFit disables scene-conditioned object generation and deformation-aware adaptation, while retaining view-aligned VLM grounding and geometry- and physics-aware refinement. This setting serves as a proxy for evaluating placement quality in the fixed-asset paradigm. On the full Common Placement Set, this variant achieves 69.7% SRS and 91.7% SupR, indicating strong geometric grounding and support accuracy in a fixed-asset comparison setting.

The remaining Common Placement subsets involve viewpoint-dependent spatial relations, explicit orientation reasoning, or contact-heavy placement near surrounding geometry. We therefore use the Common Placement Set for baseline-compatible quantitative comparison, and the Fit-Critical Set to evaluate ElasticFit's additional generative and adaptive capabilities.

Comparison with representative methods. For broader context, Table 8 summarizes the design-level differences between ElasticFit and representative object insertion or layout methods. The comparison focuses on whether the target object is fixed, retrieved, or generated; whether placement is predicted directly or optimized through geometric fitting; whether the method adapts object

geometry; and whether it is designed for constrained-space insertion where a fixed asset may not satisfy the available space.

LayoutGPT, Holodeck, and LayoutVLM primarily operate in fixed-asset or retrieved-asset layout settings. They can reason about object placement at the scene or bounding-box level, but they do not modify the target object's geometry to satisfy local spatial constraints. FirePlace is closer to ElasticFit because it uses fine-grained geometric reasoning to place a given object in an existing scene, but it still assumes a pre-existing fixed asset and does not support object generation or elastic adaptation. ElasticFit extends this direction by first grounding a target fitting region and then generating or adapting the object through rigid, uniform, or elastic fitting.

C Implementation Details

This appendix details the implementation of ElasticFit's three stages: VLM grounding, scene-conditioned object prior generation, and mode-specific fitting and physics-based settling.

C.1 Stage 1: Semantic-Spatial Grounding

We provide additional details for the semantic-spatial grounding stage described in Sec. 3.2, focusing on the structured VLM output, semantic correction, ghost-box construction, orientation resolution, and prompt organization.

VLM Output Schema. Stage 1 queries the VLM with the clean rendered scene image, the Set-of-Mark image, the visible object list, and the user instruction. The VLM is constrained to return a single JSON object following the schema in Table 9. This schema separates spatial grounding, orientation reasoning, adaptation-mode selection, contact handling, gap filling, and object-generation conditioning.

Table 9: Structured JSON output predicted by Stage 1.

Field	Type	Description
`anchor_object_id`	int	Main reference object for spatial reasoning.
`support_object_id`	int	Object expected to support the inserted object.
`target_uv`	$[u, v]$	Image-space target point for the object's bottom center.
`look_at_uv`	$[u, v]$	Optional point that the object front should face.
`top_uv`	$[u, v]$	Optional top contact point for leaning or tilted poses.
`scale`	$[w, d, h]$	Object size in the default upright pose.
`deformation_tag`	int	Adaptation mode: rigid, elastic, or uniform.
`pose_tag`	int	Pose mode: upright, lying flat, or upside down.
`orientation_relation_tag`	int	Default, parallel/aligned, or perpendicular relation.
`orientation_reference_object_id`	int	Reference object for orientation alignment.
`yaw_offset_degrees`	float	Explicit user-specified yaw offset.
`pitch_offset_degrees`	float	Explicit user-specified pitch offset.
`is_snapped`	bool	Whether the object should be flush against another object.
`snap_object_id`	int	Object used for horizontal snapping.
`enable_gap_filling`	bool	Whether the instruction requests filling available space.
`fill_front/back/left/right`	bool	Local directions for gap expansion.
`place_object_prompt`	string	Prompt for isolated object generation.
`place_object_name`	string	Short object name for logging and visualization.

Semantic Correction. The VLM-predicted target point may fall slightly outside the intended support surface. Let $\hat{q} = (\hat{u}, \hat{v})$ be the predicted target pixel and $\hat{s}$ the predicted support object ID. If the ID buffer does not match the expected support object, we replace $\hat{q}$ with the nearest visible pixel on the intended support object:

$$q^\star = \arg\min_{q \in \Omega(\hat{q})} \|q - \hat{q}\|_2 \quad \text{s.t.} \quad B_{\mathrm{id}}(q) = \hat{s}. \tag{2}$$

The grounded support point is then obtained from the XYZ buffer:

$$p_{\mathrm{surf}} = B_{\mathrm{xyz}}(q^\star). \tag{3}$$

Ghost-Box Construction. The predicted object scale is interpreted in the default upright pose, $b_{\text{box}} = (w, d, h)$. Given the grounded support point p_{surf} and orientation $R(\theta_{\text{yaw}}, \theta_{\text{pitch}})$, we place the ghost box by anchoring the center of its bottom face at p_{surf}:

$$c_{\text{box}} = p_{\text{surf}} - R(\theta_{\text{yaw}}, \theta_{\text{pitch}}) \begin{bmatrix} 0 \\ 0 \\ -h/2 \end{bmatrix}. \tag{4}$$

This convention keeps the object grounded when height or pitch changes.

Orientation Resolution. ElasticFit resolves object orientation using a priority order. Explicit yaw instructions in the user prompt override inferred facing cues. Otherwise, yaw is estimated from `look_at_uv` when available, or from the oriented bounding box of a reference object for relations such as "parallel to", "aligned with", or "perpendicular to". For leaning configurations, pitch is estimated from the vector between the support point and the predicted `top_uv`; for lying configurations, the base pitch is set to $-\pi/2$.

Prompt Structure. Table 10 summarizes the Stage 1 prompt organization. The prompt specifies the input view, coordinate convention, visible object list, placement rules, adaptation-mode definitions, snapping and gap-filling logic, and the required JSON output schema. The full prompt template will be released with the code for reproducibility.

Table 10: Prompt structure for Stage 1 VLM reasoning.

Prompt block	**Purpose**
Role and task	Defines the VLM as a 3D spatial reasoning stage for placing a new object in an existing 3D scene.
Scene context and inputs	Provides image resolution, normalized image coordinates, visible object IDs, object metadata, and the user instruction.
Reference and scale rules	Selects a valid anchor object and predicts the target object dimensions in its default upright pose.
Coordinates and placement rules	Predicts the target support point, optional facing point, and optional top-contact point for leaning or lying poses.
Physical attributes and orientation	Predicts adaptation mode, pose tag, orientation relation, yaw offset, pitch offset, and orientation reference object.
Snapping and gap filling	Handles contact-heavy instructions such as *against* or *flush with*, and fill-space instructions such as *fill*, *occupy*, or *fit*.
Image-generation prompt	Produces an isolated-object prompt and a short object name for scene-conditioned generation.
Output schema	Enforces a strict JSON object with predefined keys and no additional text.

C.2 Stage 2: Scene-Conditioned Object Prior

This section provides additional implementation details for Stage 2 beyond the high-level description in Sec. 3.3. We focus on prompt construction, topology-aware canvas sizing, and generation controls used to produce stable object-centric images for 3D lifting.

Prompt Construction. The object prompt predicted by Stage 1 is converted into an object-centric image-generation prompt by appending a canonical description:

```
[object_description], front view, full object visible, centered,
upright, isolated single object, clean light gray studio background,
realistic product photo, filling the entire frame from edge to edge,
perfectly fitted, extremely large relative scale within canvas,
vivid colors, high contrast against the background
```

These attributes reduce background clutter, cropping, and multi-object generations before image-to-3D lifting.

Topology-Aware Canvas and Prompt Tags. Given the ghost-box dimensions $\mathbf{b}_{\text{box}} = (b_x, b_y, b_z)$ predicted by Stage 1, we set the front-view canvas aspect ratio using the width-height proportion of the fitting region:

$$r = \frac{b_x}{b_z}. \tag{5}$$

Starting from a target area of 768^2 pixels, we compute a canvas size whose aspect ratio matches r, then round the width and height to the nearest multiple of 64 and clamp both dimensions to the supported range $[512, 1024]$. For elastic objects, we also prepend a coarse topology tag to the prompt: `wide horizontal` if $r > 1.2$, and `tall vertical` if $r < 0.8$. No topology tag is added for rigid or uniform modes.

Negative Prompt. We use a negative prompt to suppress multiple-object layouts, cropping, indoor backgrounds, and low-quality artifacts, with terms such as:

```
multiple objects, cropped, partial object, scene, room, floor,
wall, table, sofa, bed, shelf, background clutter, blurry, deformed,
duplicate, text, watermark.
```

This helps keep the generated image object-centric for subsequent 3D reconstruction.

C.3 Stage 3: Geometry- and Physics-Aware Fitting

This section provides implementation details for the fitting and physics-guided settling stage described in Sec. 3.4. All fitting procedures operate in the ghost-box local frame defined by the predicted center, dimensions, yaw, and pitch from Stage 1. The scene is represented by dense surface samples with normals, cropped around the ghost box to reduce computation while preserving local support and collision geometry.

Table 11: Summary of the three adaptation modes in Stage 3.

Mode	Scale	Main variable	Solver
Rigid	$(1, 1, 1)$	Translation near ghost box	Heuristic search
Uniform	(α, α, α)	Translation and global scale	Gradient optimization
Elastic	(s_x, s_y, s_z)	Box-local extents	Box-aligned expansion

Mode-Specific Fitting. Rigid fitting uses a broad-to-narrow search: candidate anchors are first filtered by proxy-box collision and support checks, then shortlisted candidates are evaluated using mesh-level penetration statistics. Uniform fitting optimizes translation and a single isotropic scale factor while retaining the best safe checkpoint. Elastic fitting searches for a maximum empty box in the ghost-box local frame by expanding dense XY seeds along local axes under collision and support checks, then maps the generated mesh into the selected box. Algorithm 1 summarizes the overall fitting procedure before the detailed objective and checks are described below.

Optimization Objective. For optimization-based fitting, the objective combines collision, box-bound, gravity, centering, and scale terms:

$$\mathcal{L} = \lambda_{\text{col}}\mathcal{L}_{\text{col}} + \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{grav}}\mathcal{L}_{\text{grav}} + \lambda_{\text{center}}\mathcal{L}_{\text{center}} - \lambda_{\text{scale}}\alpha. \tag{6}$$

Here, $\mathcal{L}_{\text{col}}$ penalizes point-to-plane penetration against nearby scene samples, $\mathcal{L}_{\text{box}}$ penalizes violations of the ghost-box bounds, $\mathcal{L}_{\text{grav}}$ encourages lower physically grounded placements, and $\mathcal{L}_{\text{center}}$ keeps the object near the predicted target region. For uniform mode, α is the optimized isotropic scale factor. Rigid mode uses heuristic search rather than this objective, while elastic mode uses box-local expansion instead of gradient optimization.

Collision and Support Checks. Collision is estimated using both coarse proxy-box tests and mesh-level point-to-plane checks. For mesh-level checks, each object vertex queries nearby scene samples and computes a local signed distance using scene normals. Penetration statistics such as maximum penetration, mean penetration, and penetration ratio are used to select or reject candidates. Support reasoning uses upward-facing scene samples under the object footprint and robust height estimates from top-ranked support points, reducing sensitivity to noisy scan geometry.

Algorithm 1: Geometry- and Physics-Aware Fitting

Input: Initial mesh O_0, grounding representation G, scene samples P
Output: Final inserted object O^*
Transform O_0 and local scene samples into the ghost-box frame defined by G;
Crop nearby support and collision geometry from P;
if $m_{\mathrm{def}} = \mathrm{rigid}$ **then**
  Search for a grounded pose while preserving object scale;
else if $m_{\mathrm{def}} = \mathrm{uniform}$ **then**
  Optimize translation and isotropic scale under geometric and support terms;
else if $m_{\mathrm{def}} = \mathrm{elastic}$ **then**
  Find a collision-aware box-local region and apply axis-dependent scaling;
Select the best fitted candidate using penetration and support statistics;
Run a short rigid-body settling simulation from a lifted initialization;
Apply a conservative support-aware grounding correction;
return O^*

Physics-Guided Settling and Final Support Clamp. After mode-specific fitting, ElasticFit applies a short PyBullet rigid-body settling step. The scene is loaded as a static concave triangle mesh cropped around the ghost box, while the adapted object is represented as a dynamic rigid body initialized slightly above the fitted pose to avoid initial penetration. A short simulation with gravity, contact response, friction, and damping refines the final pose. We then apply a conservative grounding correction and a support-object-aware clamp to remove small residual gaps and prevent the object bottom from falling below the intended support surface. The clamp uses the predicted support object ID when available, avoiding correction to nearby but semantically incorrect geometry.

D Task Sets and Prompt Construction

This section describes the task sets used in our evaluation and how prompts are constructed. We design two groups of tasks: a Common Placement Set for baseline-compatible placement evaluation, and a Fit-Critical Set for evaluating the complete ElasticFit under pose, contact, and local fitting constraints.

Table 12: Task taxonomy used in the evaluation. All tasks include a base support condition specifying the intended placement surface; the listed types indicate the primary additional spatial or fit-critical constraint.

Task set	Subset / category	Primary constraint types
Common Placement	Distance	near, far
	Directional	left of, right of, in front of, behind
	Facing	face to
	Alignment	aligned with, parallel to
	Against wall	against wall
Fit-Critical	Pose-and-contact	user-defined rotation, leaning, lying
	Gap-fitting	between objects, wall-side gap, under or inside space

D.1 Common Placement Set

The Common Placement Set contains 60 tasks, denoted as S001–S060. It is used for cross-dataset validation and baseline comparison. All methods receive the same prompts, target object categories, and scene conditions, and all methods use fixed target assets so that the comparison focuses on placement quality rather than object generation or shape adaptation.

The set covers five primary relation categories, with 12 tasks assigned to each category: *distance*, *directional*, *facing*, *alignment*, and *against-wall* relations. These categories indicate the main relation being tested in each task, although individual prompts may include additional contextual constraints. Distance tasks include relations such as *near* and *far*. Directional tasks include *left of*, *right of*, *in*

front of, and *behind*. Facing and alignment tasks evaluate object orientation with respect to reference objects, while against-wall tasks evaluate whether the object is placed near a wall-side region with a plausible orientation.

Prompt construction. Common placement prompts follow templates such as:

> Place a [object] on the [support object] near [reference object].
> Place a [object] on the [support object], to the left of [reference object].
> Place a [object] on the [support object], facing [reference object].
> Place a [object] on the [support object], aligned with [reference object].
> Place a [object] on the [support object] against the wall.

These tasks test whether a method can identify the intended placement region, ground the object in the scene, and satisfy semantic-spatial constraints without relying on object-shape adaptation.

D.2 Fit-Critical Set

The Fit-Critical Set contains 60 tasks, denoted as F001–F060. It is used to evaluate complete ElasticFit, including scene-conditioned object prior generation, 2D-to-3D lifting, and deformation-aware fitting. The set builds on the spatial-relation structure of the Common Placement Set, but adds pose, contact, and constrained-space requirements that require fit-aware insertion rather than rigid placement alone.

The Fit-Critical Set is divided into two subsets: pose-and-contact tasks and gap-fitting tasks.

Pose-and-contact tasks. The first 30 tasks, F001–F030, focus on explicit pose, orientation, and contact constraints. They extend common placement prompts with three additional relation types: *user-defined rotation*, *leaning*, and *lying*. These tasks test whether a method can interpret pose-level instructions and produce physically plausible contact with support surfaces or nearby objects.

Prompt construction. Pose-and-contact prompts follow templates such as:

> Place a [object] on the [support object] rotated [angle] degrees.
> Place a [object] on the [support object], leaning against [reference object].
> Place a [object] lying on the [support object].

Gap-fitting tasks. The remaining 30 tasks, F031–F060, focus on constrained local spaces. They extend common placement prompts with three fit-critical relation types: *between objects*, *wall-side gap*, and *under or inside space*. These tasks test whether a method can adapt the inserted object to available scene geometry, such as narrow gaps, side regions beside furniture, corners, and under- or inside-furniture spaces.

Prompt construction. Gap-fitting prompts follow templates such as:

> Place a [object] between [object A] and [object B] to fill the gap.
> Place a [object] on the floor against the wall beside [reference object] to fill the side gap.
> Place a [object] under the [furniture] to fill the available space.
> Place a [object] inside the [furniture], parallel to it, to fill the available space.

E Evaluation Metrics

This section defines the evaluation metrics used in Sec. 4. Unless otherwise stated, automatic metrics are computed from the final inserted object, annotated task relations, semantic instance annotations, and sampled scene surface points.

E.1 Spatial Relation Success Rate

For each task, the evaluator checks the annotated spatial relations between the inserted object and the referenced scene objects. For a prediction with relation set $\mathcal{R}$, the per-task spatial relation success is

computed as

$$\mathrm{SRS}_{\mathrm{task}} = \frac{1}{|\mathcal{R}_{\mathrm{eval}}|} \sum_{r \in \mathcal{R}_{\mathrm{eval}}} \mathbf{1}[r \text{ is satisfied}], \tag{7}$$

where $\mathcal{R}_{\mathrm{eval}}$ denotes the subset of annotated relations that can be automatically evaluated from the available 3D geometry and metadata.

For distance relations, *near* and *far* are evaluated using the horizontal distance between the inserted object and the reference object, normalized by the reference object's horizontal bounding-box diagonal. For orientation relations, *face to* is evaluated by comparing the predicted object forward direction with the direction toward the reference object, and *parallel to*/*aligned with* is evaluated by the angular difference between the object and reference directions. For support relations such as *on top of*, the spatial check reuses the support success test described below.

View-dependent relations such as *left of*, *right of*, *in front of*, and *behind* can depend on the user-view frame, the reference object's intrinsic facing direction, or a dataset-specific coordinate convention. To avoid imposing an ambiguous frame convention in the automatic metric, SRS focuses on relations with stable 3D geometric definitions. Directional predicates are instead assessed through contextual plausibility scoring and qualitative analysis, where the rendered user view and original instruction are jointly available.

E.2 Collision-Free Rate

Collision-Free Rate measures whether the inserted object avoids unacceptable interpenetration with the existing scene. The scene mesh is sampled into a point set $\mathcal{P}$ with semantic object IDs. For each prediction, sampled scene points belonging to the annotated support object and wall-contact object are ignored when appropriate, since contact with these objects is expected.

Each remaining scene point is expressed in a yaw-aligned object frame for the horizontal coordinates, while its vertical coordinate is measured relative to the predicted object center. We count a scene sample as inside the predicted object box if it lies within the box extents after applying small horizontal and vertical boundary margins, which avoids treating boundary contact as collision. Let N_{inside} be the number of such inside scene samples. An insertion is considered collision-free if

$$\mathrm{CFR}_{\mathrm{task}} = \mathbf{1}\left[N_{\mathrm{inside}} \leq \tau_{\mathrm{col}}\right], \tag{8}$$

where τ_{col} is a fixed tolerance for sparse sampling noise. The final Collision-Free Rate is the percentage of evaluated tasks whose collision-free indicator is 1.

E.3 Support Success Rate

Support Success Rate measures whether the inserted object is physically supported by the annotated support surface. For each prediction, the evaluator collects scene points belonging to the annotated support object and lying within the predicted object footprint. The support height z_{sup} is estimated as the 95th percentile of these footprint points, making the measurement robust to sparse or noisy samples. The object bottom height is computed from the predicted center and height as $z_{\mathrm{bottom}} = c_z - s_z/2$.

A placement is considered supported when the object bottom is sufficiently close to the estimated support height and the footprint has enough support coverage:

$$\mathrm{SupR}_{\mathrm{task}} = \mathbf{1}\left[|z_{\mathrm{bottom}} - z_{\mathrm{sup}}| \leq \tau_{\mathrm{gap}} \ \wedge \ C_{\mathrm{sup}} \geq \tau_{\mathrm{cover}}\right], \tag{9}$$

where C_{sup} is the fraction of footprint support points lying within a small height band around z_{sup}. In our implementation, we use $\tau_{\mathrm{gap}} = 0.02\,\mathrm{m}$, a $0.02\,\mathrm{m}$ height band for coverage estimation, and $\tau_{\mathrm{cover}} = 0.08$.

For CAD-style scenes or thin support surfaces, a bounding-box support variant can be used, where the object bottom is compared to the reference object's top bounding-box face and the XY footprint overlap is used as support coverage. The final Support Success Rate is the percentage of evaluated tasks whose support indicator is 1.

```
You are an impartial evaluator for a 3D object placement task.

You will see multiple anonymized candidate placements for the SAME instruction.
The candidates are labeled only as Candidate A, Candidate B, Candidate C, and
Candidate D.
The method names are hidden.

Evaluate ONLY the inserted target object with respect to the instruction and
the visible scene context.
Ignore rendering quality aesthetics unless they make placement impossible to
judge.

Instruction:  {prompt}
Target object hint:  {target_object_hint}

Task relations, if provided:
{task_relations_text}

Important rule:
Score each candidate using the 1-5 rubric.
You may use the other candidates only to calibrate obvious placement quality
differences, but do NOT rank them directly.
Do NOT force all candidates to have different scores.
If multiple candidates are similarly plausible, they may receive the same
score.

Do NOT penalize object identity/category mismatch, shape quality, or generation
quality, unless the inserted object is missing, unrecognizable, or impossible
to judge.
Evaluate only placement plausibility.
If a relation is visually ambiguous or hard to verify from the camera view, do
NOT treat it as an issue.
Only penalize a relation when it is clearly violated.
When uncertain, give the benefit of the doubt.

Give each candidate a Contextual Plausibility Score from 1 to 5:
5 = Clearly satisfies the instruction and looks semantically natural.
4 = Satisfies the instruction overall; only minor placement awkwardness, but no
clear violation.
3 = Main support/place is correct, but one important relation is clearly wrong
or the placement is noticeably unnatural.
2 = Object is visible, but the support surface or multiple important relations
are clearly wrong.
1 = Object is missing, unrecognizable, or placed in a clearly implausible/wrong
region.
```

Figure 6: Prompt template used for Contextual Plausibility Score evaluation.

E.4 Contextual Plausibility Score

Contextual Plausibility Score (CPS) is computed using a grouped model-assisted evaluator. For each task, anonymized screenshots of candidate placements are grouped and labeled only as Candidate A, B, C, and D; method names are hidden. The evaluator receives the original instruction, a target-object hint, optional task relations, and the anonymized candidate images. Candidates from the same task are scored together to calibrate placement quality differences, but the evaluator assigns an individual 1–5 score to each candidate and is not forced to rank them or give different scores. The evaluator is instructed to evaluate only placement plausibility and to give the candidate the benefit of the doubt when a relation is visually ambiguous. Figure 6 shows the prompt template and scoring rubric. CPS has not been calibrated against human judgments and is used only as a complementary measure of placement plausibility, not of shape fidelity or deformation quality.

E.5 Penetrating Vertex Ratio

Penetrating Vertex Ratio (PVR) is used in the ablation analysis as a fine-grained diagnostic of object-scene interpenetration. For each fitted object, every mesh vertex queries its nearest scene sample, and a local signed distance is estimated using the corresponding scene normal. A vertex is counted as penetrating when the estimated signed distance is negative:

$$\mathrm{PVR} = \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \mathbf{1}[d_i < 0], \tag{10}$$

where $\mathcal{V}$ is the set of object vertices and d_i is the estimated signed distance for vertex v_i. Lower values indicate that a smaller fraction of the object mesh penetrates the scene. Unlike the binary Collision-Free Rate, PVR captures finer differences between ablation variants.

F Baseline Adaptation Details

We adapt LayoutGPT, Holodeck, and LayoutVLM to the single-object insertion setting used for the Common Placement Set. For all methods, the input scene remains fixed and only the target object is inserted. Each method receives the same scene, target object category, fixed target asset, and natural-language instruction. This controlled setting isolates semantic-spatial placement quality from differences in object generation or asset selection.

Table 13: Summary of adapted baselines for the single-object insertion setting.

Method	Spatial semantics	Physical plausibility
LayoutGPT	Directly predicts target position and yaw from a CSS-style scene description.	No explicit collision or support refinement; placements are only clamped to scene bounds.
Holodeck	Generates Holodeck-style spatial constraints and searches over discrete placement candidates.	Uses AABB-level collision, support, wall, and relation checks during candidate scoring.
LayoutVLM	Initializes object pose from vision-language reasoning and semantic constraints.	Uses bounding-box-level differentiable optimization for feasible placement refinement.
ElasticFit	Uses view-aligned structured grounding with support validation and local scene cues.	Uses geometry-aware local mesh grounding and physics-guided placement refinement.

LayoutGPT. LayoutGPT is adapted as a text-only layout prediction baseline for single-object insertion. Following the original LayoutGPT formulation, the scene is represented as a CSS-style structured layout, where each existing object is described by its semantic name, caption, 3D bounding-box size, center position, and yaw orientation. For each task, the input prompt provides the fixed scene bounds, a list of selected context objects, the target object category and fixed asset size, and the user instruction. The model is asked to output exactly one CSS-style block for the inserted target object:

$$o_T = (\ell_T, w_T, h_T, x_T, y_T, z_T, \theta_T), \tag{11}$$

where $d_T = (\ell_T, w_T, h_T)$ denotes the fixed asset dimensions, $p_T = (x_T, y_T, z_T)$ is the predicted object center, and θ_T is the predicted yaw orientation. The existing scene objects are kept unchanged, and no collision, support, scaling, deformation, pitch, or roll refinement is applied after prediction. Predicted positions are only clamped to the scene bounds before evaluation.

Holodeck. Holodeck is adapted as a constraint-generation and search-based placement baseline while preserving the main design of the original method. The original Holodeck uses an LLM to generate spatial relational constraints between objects, rather than directly predicting absolute bounding-box values, and then searches for layouts that satisfy these constraints while enforcing hard constraints such as collision avoidance and room-boundary validity [4]. In the adapted single-object insertion setting, the existing scene is kept fixed and the same principle is applied only to the inserted

Table 14: Prompt structure for the adapted LayoutGPT baseline.

Prompt block	Purpose
Task instruction	Specifies single-object insertion into a fixed existing 3D scene.
Coordinate system	Defines `left`, `depth`, `top`, and yaw orientation.
Scene bounds	Provides the valid 3D range of the scene in centimeters.
Output template	Enforces one CSS-style block for the target object.
Rules	Predicts only position and yaw while keeping the target asset fixed.
Scene objects	Lists context objects with captions, size, position, and orientation.
Target object size	Provides fixed length, width, and height of the inserted asset.
User instruction	Provides the natural-language placement request.

target object. This preserves Holodeck's core design choice: language-based relation generation followed by solver-based placement search.

First, a language model receives the user instruction, target object description, target asset size, and a list of pre-existing scene objects described by ID, semantic label, center, size, yaw, and coarse location. It outputs a Holodeck-style constraint line for `object-target`, using relations such as *near*, *left of*, *on top of*, *face to*, *align*, and *against wall*. The output is parsed into relation-anchor pairs, and invalid relations or anchors are discarded.

Second, an AABB-based solver generates discrete candidate placements around the referenced anchors. Directional relations sample candidates around the anchor according to its yaw, *on top of* uses dense sampling on the anchor's top AABB surface, and *against wall* samples candidates along semantic wall AABBs. Candidates are ranked using relation scores, floor-bound checks, AABB-level collision checks, and support validation. The target asset remains fixed, and no object generation, deformation, pitch/roll estimation, local mesh grounding, or physics-based settling is applied.

LayoutVLM. We adapt LayoutVLM as a vision-language-guided placement baseline for single-object insertion. All existing scene objects are kept fixed, and the method predicts and refines only the inserted target object. For each task, the VLM receives three visual inputs: a top-down sliced scene plan with grid lines and object IDs, a perspective rendering of the current scene, and a canonical rendering of the target asset. It also receives a structured scene program describing the fixed scene objects with semantic labels, bounding-box sizes, centers, and yaw orientations.

The VLM is prompted to output a LayoutVLM-style Python program containing an initial target pose and optional solver-style spatial relations, such as distance constraints, `on_top_of`, `point_towards`, `align_with`, and `against_wall`. We parse the predicted target position, yaw, and relations from this program. When an explicit relation in the instruction is omitted, such as *near*, *far*, *on top of*, or *facing*, we recover the corresponding prompt-required constraint from the instruction. We then apply a self-consistency filter that removes VLM-predicted relations incompatible with the predicted initial pose, while preserving prompt-required relations.

Given the VLM-predicted initial pose (p_0, θ_0) and the parsed constraint set $\mathcal{C}$, the adapted LayoutVLM solver optimizes only the target center p and yaw θ:

$$\min_{p,\theta} \lambda_{\mathrm{rel}} L_{\mathrm{rel}}(p, \theta; \mathcal{C}) + \lambda_{\mathrm{sup}} L_{\mathrm{sup}}(p, \theta) + \lambda_{\mathrm{col}} L_{\mathrm{AABB}}(p, \theta) + \lambda_{\mathrm{bd}} L_{\mathrm{bd}}(p, \theta) + \lambda_{\mathrm{init}} \|p - p_0\|_2.$$

Here, L_{rel} encodes the parsed spatial relations, L_{sup} enforces support containment for `on_top_of` relations, L_{AABB} penalizes bounding-box intersections with fixed scene objects, L_{bd} keeps the target inside the scene bounds, and the initialization term preserves the VLM-predicted pose.

The target asset size is kept fixed throughout this process. Physical feasibility is approximated using bounding-box-level collision, support, and boundary checks. No object generation, mesh-level local grounding, elastic deformation, pitch/roll adaptation, or physics-based settling is applied. Thus, this baseline evaluates vision-language-guided pose prediction and bounding-box-level optimization without ElasticFit's scene-conditioned generation, and deformation-aware fitting.

ElasticFit placement-only setting. For the Common Placement Set baseline comparison, ElasticFit is evaluated in a placement-only setting. Scene-conditioned object generation and deformation-aware

fitting are disabled, and a fixed target asset is used for each task. The method retains view-aligned VLM grounding, target validation through the ID buffer, and geometry- and physics-aware placement refinement. This setting isolates ElasticFit's placement and grounding quality from the additional cost of object generation and 3D reconstruction.

Table 15: Average runtime comparison on the Common Placement Set.

Method	Avg. total time (s)	Notes
LayoutGPT	42.97	End-to-end runtime
Holodeck	16.71	End-to-end runtime
LayoutVLM	100.03	End-to-end runtime
ElasticFit	11.53	Placement-only; 10.50 s VLM + 1.03 s fitting

Runtime and Computational Cost. Table 15 reports the average runtime in the adapted single-object insertion setup. In the placement-only setting, ElasticFit takes 11.53 seconds per task on average, including 10.50 seconds for VLM reasoning and 1.03 seconds for local fitting and physics refinement. ElasticFit is faster than all three adapted baselines in this setting, with most runtime spent on VLM reasoning. Since runtime depends on implementation details, hardware, API latency, and retry behavior, these numbers should be interpreted as measurements for our experimental setup rather than universal speed comparisons.

Compute Resources. All local experiments were run on a workstation with an NVIDIA GeForce RTX 4070 Ti SUPER GPU with 16 GB dedicated GPU memory and approximately 64 GB system RAM. CPU preprocessing, scene rendering, geometric fitting, physics simulation, and evaluation scripts were executed on the same local workstation. VLM reasoning and model-assisted evaluation were performed through remote API calls, while image generation and 2D-to-3D reconstruction were executed locally when required. No model training or fine-tuning was performed.

The complete ElasticFit method additionally requires scene-conditioned image generation and 2D-to-3D reconstruction, so its runtime is higher and depends on generation, reconstruction, API latency, and retry behavior. The reported experiments include the 60-task Common Placement Set, the 60-task Fit-Critical Set, and the 30-task gap-fitting ablation subset. Total compute is estimated from logged per-task runtimes rather than from a cluster-level accounting system, and excludes preliminary debugging runs, failed API calls, and exploratory parameter tuning.

G Additional Qualitative Examples

This section provides additional qualitative examples for the Common Placement Set and the Fit-Critical Set.

G.1 Additional Common Placement Results

Figure 7 shows additional qualitative results on the Common Placement Set. These examples cover diverse spatial relations and illustrate that ElasticFit produces placements that are spatially grounded, supported by the intended scene geometry, and visually consistent with the input instruction.

G.2 Additional Fit-Critical Results

Figure 8 shows additional qualitative results on the Fit-Critical Set. These examples illustrate final fitted insertions produced by ElasticFit under pose-and-contact constraints and gap-fitting constraints in constrained local scene regions.

H Asset Licenses

Table 16 lists the third-party 3D assets used in our experiments. All assets were obtained from Sketchfab [44] under the Creative Commons Attribution 4.0 (CC BY 4.0) license. We use these assets in accordance with their license terms and do not redistribute the original third-party asset files.

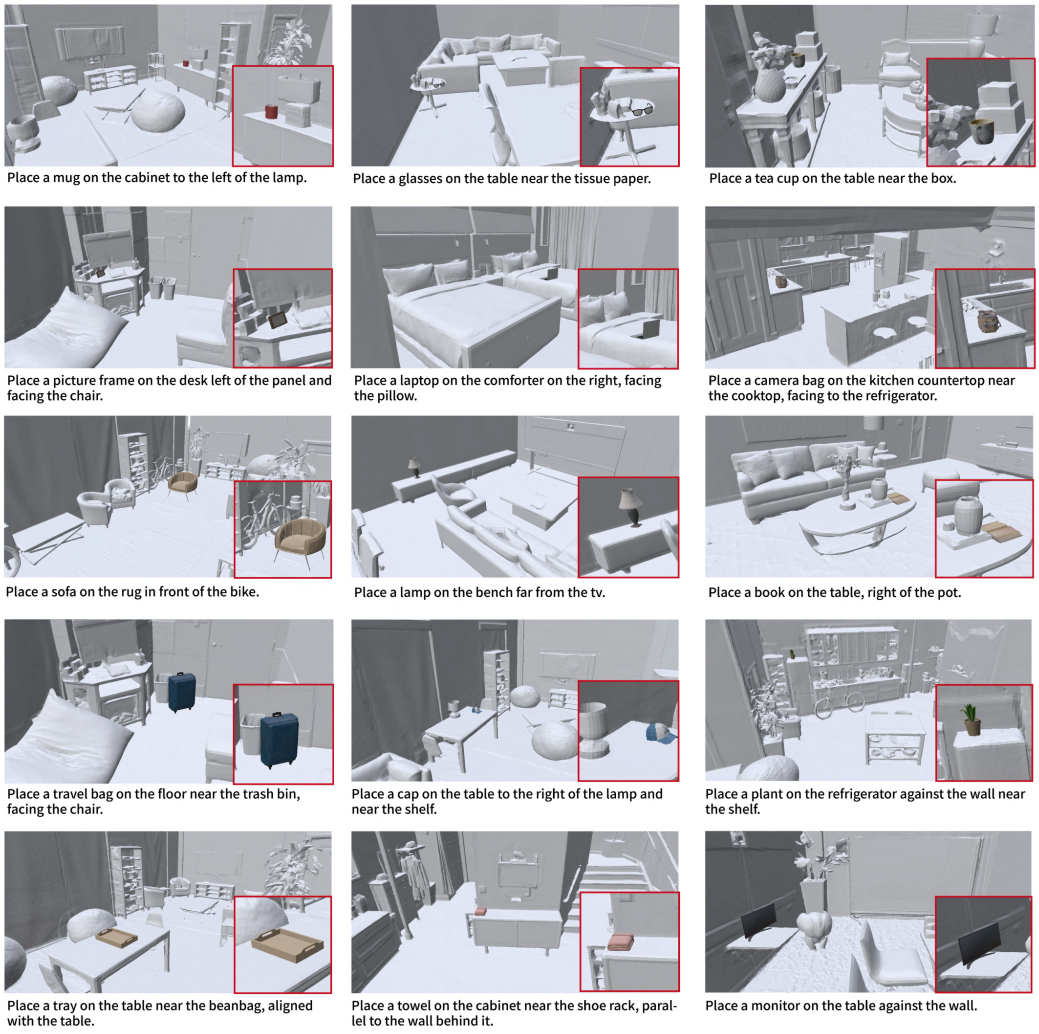

Figure 7: Additional qualitative results on the Common Placement Set. Each example shows the final fitted placement produced by ElasticFit, with red boxes highlighting the inserted object. The examples include diverse spatial relations such as left/right, near/far, facing, alignment, and against-wall placement.

Table 16: Third-party 3D assets used in the experiments.

Assets		
Alarm clock	Book	Chair
Backpack	Bowl	Coffee machine
Bike	Camera bag	Fan
Camera	Cap	Glasses
Lamp	Laptop	Monitor
Mug	Picture frame	Pillow
Plant	Salad plate	Small oven
Sofa	Speaker	Tea cup
Teddy bear	Towel	Toy car
Toy rabbit	Travel bag	Tray
Vase		



Figure 8: Additional qualitative results on the Fit-Critical Set. The examples show final fitted insertions produced by ElasticFit, including pose-and-contact cases and gap-fitting cases in constrained local scene regions.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately state the paper's scope and main contributions: fit-aware single-object insertion in pre-existing 3D scenes using VLM grounding, scene-conditioned object generation, and geometry- and physics-aware fitting. The main quantitative claims are supported by the experimental results in Sec. 4, including the reported improvements in spatial relation success and support success over the strongest baseline.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses main limitations in the last section and in more detail in Appendix A, including directional fitting errors, deformation artifacts from soft object-prior control and elastic fitting, and concave support errors in container-like regions. It also discusses future directions such as stronger spatial conditioning, more robust orientation reasoning, category-aware deformation constraints, and iterative verification.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: This paper does not present theoretical results, formal theorems, lemmas, or proof-based claims. The mathematical expressions are used to define the proposed representation and optimization objectives rather than to state theoretical guarantees.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides information needed to reproduce the main experimental results, including the method design in Sec. 3, task construction in Sec. 4 and Appendix D, evaluation metrics in Sec. 4 and Appendix E, implementation details in Appendix C, and baseline adaptations in Appendix F. Exact numerical reproduction may depend on access to the cited datasets and third-party foundation models or APIs, whose versions or behavior may change over time.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide a supplementary evaluation package containing task definitions, evaluation scripts, metric computation code, saved raw prediction records used for the reported experiments, precomputed contextual plausibility scores, final CSV tables, and reproduction instructions. These materials allow inspection of the evaluation protocol and regeneration of the reported quantitative tables from the released intermediate predictions and scores. Re-running the full end-to-end generation pipeline requires external foundation-model APIs, large third-party generative models, and licensed 3D scene datasets, so the supplementary package focuses on the evaluation artifacts needed to reproduce the reported experimental tables.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the experimental setup in Sec. 4, including datasets, task sets, baseline protocols, evaluation metrics, and ablation settings. Additional details are provided in the appendices, including system limitations, implementation details, task construction, metric definitions, baseline adaptations, contextual plausibility evaluation, runtime analysis, and supplementary qualitative results. Since ElasticFit is not trained

end-to-end, standard training details such as train/test splits for model training and learned optimizer hyperparameters are not applicable.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports aggregate results over fixed evaluation task sets, but does not report error bars, confidence intervals, or statistical significance tests. Repeated runs are costly because the paper depends on external VLM calls, generative object creation, 2D-to-3D lifting, and physics-based fitting. The results are therefore supported through task-set-level aggregate metrics, qualitative examples, and ablation studies rather than statistical significance analysis.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports compute resources in Appendix B, including the local workstation setup, NVIDIA GeForce RTX 4070 Ti SUPER GPU with 16GB VRAM, approximately 64GB system RAM, use of remote API calls, and average per-task runtime for the placement-only setting. No model training or fine-tuning is performed; the main compute costs come from VLM/API calls, image generation, 2D-to-3D lifting, geometric fitting, physics simulation, and evaluation.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and believe the research conforms to it. The work uses existing 3D scene datasets and generated insertions, does not involve human subjects or private user data. Limitations and broader impacts are discussed in the paper.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses potential positive impacts for AR/VR content creation, interior design, and embodied AI, as well as possible negative impacts such as unrealistic or misleading scene edits, unsafe physical suggestions if deployed in real environments, and dependence on generative-model biases. The paper frames ElasticFit as a research prototype rather than a deployed system.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new high-risk pretrained model, image generator, or scraped dataset. The method uses existing external foundation models and public 3D scene datasets, and any released supplementary material is limited to anonymized task definitions, evaluation scripts, example prediction files, and reproduction instructions. Therefore, the safeguard requirements for releasing high-risk models or scraped datasets are not applicable.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the existing datasets, models, software packages, and prior methods used in the experiments. Appendix H lists the 31 third-party 3D assets used in the experiments and states that they were obtained from Sketchfab under the Creative Commons Attribution 4.0 (CC BY 4.0) license. We use these assets in accordance with their license terms and do not redistribute the original third-party asset files.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

Justification: The paper introduces evaluation task definitions for the Common Placement Set and Fit-Critical Set, including prompt templates, task taxonomy, and evaluation metadata. These assets are documented in Appendix D and provided in the supplementary material together with evaluation scripts and reproduction instructions. The assets do not contain personal data or human-subject data, so consent requirements are not applicable.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. No participants were recruited, no human-subject data were collected, and no compensation was provided. The contextual plausibility evaluation is performed using a model-assisted evaluator rather than human annotators.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. No participants were recruited, no human-subject data were collected, and no personal data were used. Therefore, IRB approval or equivalent human-subject review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The paper explicitly describes the use of VLM/LLM-based reasoning as a core component of ElasticFit. In particular, the method uses VLM reasoning to infer structured, scene-grounded fitting cues for object insertion, as described in Sec. 3 and Sec. 3.2. The paper also describes model-assisted contextual plausibility evaluation in Sec. 4 and Appendix E, and discusses baseline adaptation details in Appendix F. These models are used for fitting-cue prediction, baseline reasoning, and semantic plausibility evaluation, not merely for writing, editing, or formatting.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.