PRIMLLM: LEARNING TO ASSEMBLE 3D GEOMETRIC PRIMITIVES FROM TEXT VIA ANNOTATION-FREE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Geometric primitive assembly provides a compact and structurally editable 3D representation by composing simple parametric shapes. We study text-conditioned primitive assembly: generating executable primitive programs from text without reference geometry. This task requires jointly reasoning about an object's semantic components and their geometric configurations. Existing approaches often rely on reference geometry or paired supervision over predefined component vocabularies, leaving open-ended text-conditioned assembly underexplored. We propose PrimLLM, a two-stage post-training framework that adapts a 9B open-weight language model to assemble general-purpose primitives with limited annotation. At both training and inference, the model is conditioned on dense captions produced by component planning, specifying component geometry and spatial relations. Structure-Aware Supervised Fine-Tuning (SFT) learns program syntax and mappings from component-level semantics to geometric configurations using dense caption-program pairs. Annotation-Free Reinforcement Learning (RL) then improves assembly quality by combining rule-based rewards for program format validity and physical plausibility with vision-language model feedback on multi-view renderings, without additional program annotations. The generation policy remains text-only, with visual feedback used exclusively to compute RL rewards. In our evaluation, PrimLLM matches or surpasses frontier proprietary models in semantic consistency and structural plausibility, and outperforms all evaluated open-weight baselines, including models with up to two orders of magnitude more parameters. Starting from the SFT model, RL improves assembly quality on both seen and unseen categories using only a small set of text descriptions per category. Code and data will be released.

1 INTRODUCTION

Geometric primitive assembly constructs 3D objects by combining standard primitives, such as cubes, cylinders, and spheres (Tulsiani et al., 2017; Monnier et al., 2023; Ye et al., 2025). This explicit representation is compact and directly editable at the structural level, making it well suited to computer-aided design and game asset creation (Lê et al., 2021; Li et al., 2023; Tian et al., 2025). We study text-conditioned primitive assembly, where the goal is to generate an executable primitive program from a natural-language description alone, without any reference mesh or point cloud. The program specifies a variable number of primitive instances and their spatial transformations to construct the object. This requires the model to infer the object's semantic components, approximate them with suitable primitives, and arrange these primitives into a coherent whole.

Most existing studies formulate primitive assembly as a geometric reconstruction problem. Optimization-based methods fit primitives to target shapes (Liu et al., 2022; 2023), while shape-conditioned autoregressive methods predict primitive assemblies from geometric observations (Ye et al., 2025; Tian et al., 2025). These methods can produce meaningful abstractions, but the structural cues they exploit come from the input geometry rather than from language. In contrast, text-conditioned generation requires inferring such structure from language alone. AssetFormer (Zhu et al., 2026) explores this setting by generating architectural assets from paired training data using a predefined vocabulary of building components. An important question thus remains open: how

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

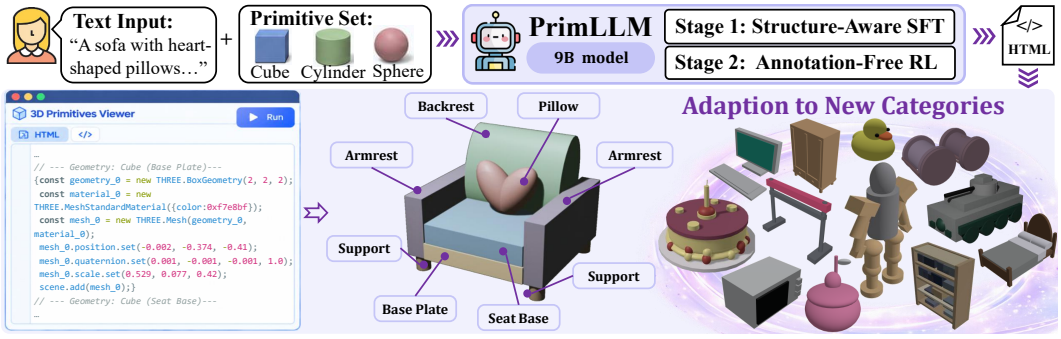


Figure 1: PrimLLM is a lightweight 9B language model for text-conditioned primitive assembly. In a text-to-text formulation, it maps text descriptions to executable programs that assemble standard primitives into 3D objects. Through Structure-Aware SFT and Annotation-Free RL, PrimLLM improves assembly quality and supports adaptation to new categories without additional annotations.

can a model infer object components from natural language, compose general-purpose geometric primitives, and adapt to unseen categories?

Pretrained large language models (LLMs) offer a promising starting point for this task, as they combine semantic knowledge with code-generation capabilities. Recent program-generation systems (Wang et al., 2026a;b) based on closed-source LLMs show the potential of converting natural language into 3D content. However, it remains unclear whether lightweight, open-weight LLMs can achieve semantic-to-primitive assembly under limited supervision.

An intuitive idea is to perform supervised fine-tuning (SFT) on text–primitive program pairs. SFT enables LLMs to adapt to geometry serialized as programs (Wang et al., 2024; Fang et al., 2025). A valid program, however, does not necessarily yield a reasonable object, as primitive assembly requires reasoning about component-level semantics and their geometric configurations. Moreover, the token-level imitation objective of SFT does not optimize the quality of the final 3D assembly, while limited supervision restricts the model's ability to explore diverse and reasonable assemblies for the same description. Recent reasoning models, such as DeepSeek-R1 (Guo et al., 2025), have demonstrated that reinforcement learning (RL) provides a complementary path by optimizing outcomes. The challenge is to translate outcome feedback into effective rewards that guide self-exploration toward higher-quality primitive assemblies without requiring additional annotations.

To address these challenges, we propose PrimLLM, a two-stage post-training framework that adapts an open-weight LLM for text-conditioned primitive assembly with limited supervision. First, in the Structure-Aware SFT stage, we introduce component planning, a coarse-to-fine procedure that infers object components and their organization from brief descriptions to produce dense captions. By pairing these captions with primitive programs, the model learns program syntax and establishes implicit associations between component-level semantics and programs. Second, in the Annotation-Free RL stage, we leverage multi-dimensional rewards covering program validity, physical plausibility, and rendered visual quality to guide the model toward better primitive compositions and geometric configurations without additional annotated data. In both stages, the policy takes only text as input, and visual information is used solely for reward computation. We conduct extensive comparative and ablation studies with both quantitative and qualitative evaluations. Results show that our 9B model matches or surpasses frontier proprietary models, as well as open-weight models with substantially larger parameter scales, in semantic consistency and structural plausibility. Moreover, RL training with a small amount of annotation-free data enables PrimLLM to adapt to categories unseen during SFT. These results demonstrate an annotation-efficient path for extending the primitive generation capabilities of lightweight, open-weight LLMs beyond supervised categories, as shown in Figure 1.

2 Related Work

Geometric Primitive Assembly. This task represents 3D objects by composing primitives, aiming to capture object structure. Optimization-based methods formulate assembly as geometric fitting, optimizing primitive parameters against point clouds and meshes (Liu et al., 2022; 2023) or multi-view renderings (Monnier et al., 2023; Kluger et al., 2024; Gao et al., 2025). However, prioritizing geometric accuracy can yield over-segmented, fragmented assemblies. Learning-based methods

(Ye et al., 2025; Li et al., 2026) use autoregressive models to learn primitive compositions from geometric supervision. While these methods produce meaningful abstractions, they rely on input geometry to provide structural information. In contrast, our method infers component structure from text alone, learns correspondences between semantics and geometric primitives, and employs RL with annotation-free rewards to guide self-exploration and support adaptation to novel categories.

Large Language Models for 3D Generation. LLMs have increasingly been applied to 3D content generation. Several works (Wang et al., 2024; Fang et al., 2025) explore mesh generation by serializing 3D geometry into tokens or text sequences that LLMs can process. These methods generally rely on high-quality supervised data, while mesh representations do not inherently expose semantic components, limiting flexible editing and composition. Structured and procedural 3D modeling methods (Avetisyan et al., 2025; Lu et al., 2025; Sun et al., 2025) instead represent 3D content through structured descriptions or executable programs, improving interpretability and editability. Several recent systems (Lin et al., 2026; Wang et al., 2026a;b) adopt LLM-based agent frameworks that organize planning, code generation, and tool use around pretrained LLMs. Rather than relying solely on inference-time orchestration, we employ a two-stage post-training framework to improve an open-weight LLM's structural understanding and primitive-program generation capabilities.

Reinforcement Learning for Reasoning. RL is widely used for post-training LLMs through policy optimization algorithms such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024). Reasoning models such as DeepSeek-R1 (Guo et al., 2025) and Kimi k1.5 (Team et al., 2025) demonstrate that RL with verifiable rewards (Lambert et al., 2024) can improve LLMs' reasoning capabilities. Whereas SFT trains models to imitate reference outputs, RL can use outcome-level feedback to guide exploration, improving performance in mathematical reasoning and code generation (Lyu et al., 2025; Guo et al., 2025). Despite these advances in textual and multimodal reasoning, applying RL to structured 3D generation remains relatively underexplored. We apply this paradigm to geometric primitive assembly, combining rule-based checks with feedback from vision-language models (VLMs) to guide exploration of plausible primitive compositions and spatial configurations.

3 PRELIMINARIES

Problem Definition. Given a text description t of an object and a predefined set of primitive types $\mathcal{T}$, our goal is to generate the object as a composition of 3D geometric primitives $\mathcal{P} = \{p_1, \ldots, p_N\}$. Each primitive p_i is defined by its type, spatial and appearance attributes: $p_i = (\tau_i, c_i, \mathbf{t}_i, \mathbf{r}_i, \mathbf{s}_i)$, where $\tau_i \in \mathcal{T}$ denotes the primitive type, including cubes, cylinders, and spheres; c_i denotes its color in hexadecimal format; $\mathbf{t}_i \in \mathbb{R}^3$, $\mathbf{r}_i \in \mathbb{R}^4$, and $\mathbf{s}_i \in \mathbb{R}^3$ denote its 3D translation, quaternion-based rotation, and axis-wise scale. Each primitive is instantiated and transformed according to these attributes, and the resulting primitives are assembled to form the final object: $\mathcal{O}(\mathcal{P}) = \bigcup_{i=1}^{N} p_i$.

Executable Primitive Program. To make primitive assemblies compatible with LLMs, we adopt HTML as the executable representation for three reasons: 1) HTML is widely present in LLM pre-training corpora, giving LLMs strong prior knowledge of its syntax and structure; 2) HTML provides a structured representation in which each primitive is encoded as an independent instance; 3) HTML can be directly executed and rendered in Web-based 3D environments such as Three.js, facilitating visual verification. Specifically, we define an HTML template (see Appendix A.1) for scene setup, primitive creation, and rendering. Each primitive instance is defined by its type and transformation parameters, as shown in Figure 3. A complete object is represented by an HTML program containing multiple primitive instances. Importantly, our method is not tied to HTML and can support other structured representations, such as JSON, with an appropriate renderer.

4 METHOD

This section introduces PrimLLM, a post-training framework that enables lightweight LLMs to assemble geometric primitives into 3D objects from text descriptions. As illustrated in Figure 2, it consists of two stages: Structure-Aware SFT and Annotation-Free RL. The former uses component planning to infer object components, transforming coarse-grained semantics into dense captions, and then performs supervised fine-tuning to establish component-level semantic-to-program mappings. The latter uses annotation-free rewards to guide exploration of better primitive compositions and spatial configurations, improving the structural plausibility and semantic consistency.

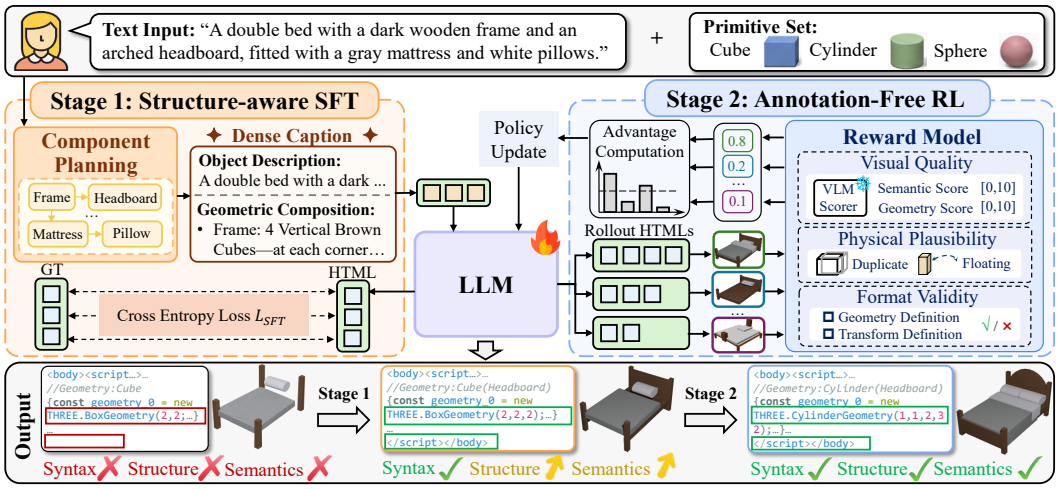


Figure 2: Overview of PrimLLM, a two-stage post-training framework: 1) Structure-Aware SFT uses component planning to generate dense captions describing components and spatial relations, guiding the model to learn correspondences between component-level semantics and programs; 2) Annotation-Free RL refines primitive assemblies through automatically computed format, physical, and visual rewards without additional annotations. The bottom panel illustrates improvements in program validity, structural plausibility, and semantic consistency across the two stages.

4.1 Stage 1: Structure-Aware SFT for Primitive Assembly Initialization

In this stage, we aim to initialize the model's capability to generate executable primitive programs from text descriptions. Direct generation requires the model to infer component composition, structural organization, and geometric configurations. Since a single description may correspond to many reasonable primitive assemblies, establishing reliable semantic-geometric correspondences is challenging under limited supervision. To address this, we introduce component planning to transform brief descriptions into dense captions that detail the object's components and their organization. We construct supervised data from these captions for SFT, enabling the model to learn a mapping from component-level semantics to primitive programs and thus achieve more reliable generation.

Component Planning. Given a text input t, we prompt an external LLM Φ_{planning} to infer the target object's components and generate a structured description, i.e., a dense caption d:

$$d = \{(k_i, g_i, l_i)\}_{i=1}^{N_c} = \Phi_{\text{planning}}(t; P_{\text{prompt}}), \tag{1}$$

where P_{prompt} is the predefined prompt template and N_c is the number of identified components. Each component is represented by a triplet (k_i, g_i, l_i), where k_i denotes the semantic label (e.g., lamp base or arm), g_i describes its geometric shape and approximate dimensions (e.g., a flat cuboid or a tall, slender cylinder), and l_i specifies its spatial relationships with other components. As illustrated in Figure 2, the dense caption explicitly captures the object's component composition, allowing the subsequent SFT and RL stages to focus on learning the mapping from component-level semantics to executable primitive programs, thereby reducing the learning burden.

Supervised Fine-tuning. Following the dense caption format, we construct supervised data (see Section 4.3) for SFT. Given a dense caption d and its corresponding HTML program $h = \{h_1, h_2, \ldots, h_L\}$, where L denotes the token sequence length, the model Φ_{sft} is trained to autoregressively generate h conditioned on d using the standard cross-entropy objective:

$$\mathcal{L}_{\text{SFT}} = -\frac{1}{M} \sum_{n=1}^{M} \sum_{l=1}^{L_n} \log p_{\Phi_{\text{sft}}} \left(h_{n,l} \mid d_n, h_{n,<l} \right), \tag{2}$$

where M denotes the number of training samples and L_n the token length of the n-th target program. Through this supervised warm-up, the model learns primitive types, geometric parameters, and program syntax, providing a reliable initialization for RL. However, SFT primarily imitates reference programs rather than directly optimizing assembly quality. Generated results may therefore retain structural defects, such as floating or intersecting primitives, and semantic inconsistencies. These limitations motivate the subsequent Annotation-Free RL stage, which further improves generation quality through multi-dimensional rewards.

4.2 STAGE 2: ANNOTATION-FREE RL FOR PRIMITIVE ASSEMBLY REFINEMENT

We introduce Annotation-Free RL to further improve primitive assembly quality. Starting from the SFT-trained model as the initial policy, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024). To avoid additional program annotations and to support adaptation to categories unseen during SFT, we design a reward mechanism that automatically evaluates format validity, physical plausibility, and visual quality, guiding self-exploration and refinement.

Annotation-Free Reward Design. We design three rewards using rule-based checks and a pretrained VLM: Format Validity Reward (R_{format}), Physical Plausibility Reward (R_{physics}), and Visual Quality Reward (R_{visual}). Each reward is normalized to $[-1, 1]$, and the overall reward is

$$R = \lambda_{\text{format}} R_{\text{format}} + \lambda_{\text{physics}} R_{\text{physics}} + \lambda_{\text{visual}} R_{\text{visual}}, \tag{3}$$

where λ_{format}, λ_{physics}, and λ_{visual} are the corresponding weighting coefficients.

• *Format Validity Reward* measures whether the generated HTML program is valid and executable. We first check HTML syntax, including tag closure, structural completeness, and the absence of illegal characters. We then verify the implementation of primitives, including geometry and material definitions, calls to `position.set()`, `quaternion.set()`, and `scale.set()`, as well as the required rendering code. The reward is computed as

$$R_{\text{format}} = 2 \cdot (\lambda_{\text{syntax}} S_{\text{syntax}} + \lambda_{\text{prim}} S_{\text{prim}}) - 1, \tag{4}$$

where $S_{\text{syntax}}, S_{\text{prim}} \in [0, 1]$ denote the scores from standard HTML syntax checks and primitive-specific checks, respectively. λ_{syntax} and λ_{prim} denote the corresponding weights.

• *Physical Plausibility Reward* evaluates whether the generated primitive assembly satisfies basic geometric constraints. We consider two types of violations: overlap and floating. For overlap, given two primitives p_i and p_j with their occupied voxel sets $\mathcal{V}_i$ and $\mathcal{V}_j$, an overlap violation is detected when the intersection-over-minimum ratio exceeds a predefined threshold δ_o:

$$\frac{|\mathcal{V}_i \cap \mathcal{V}_j|}{\min(|\mathcal{V}_i|, |\mathcal{V}_j|)} > \delta_o, \tag{5}$$

where $|\mathcal{V}_i|$ represents the number of occupied voxels of primitive p_i. For floating, a primitive p_i is considered floating if it is simultaneously far from all other primitives and from the ground plane:

$$\min_{j \neq i} \text{Dist}(p_i, p_j) > \delta_d \quad \text{and} \quad \text{Dist}(p_i, \mathcal{G}) > \delta_g, \tag{6}$$

where $\mathcal{G}$ denotes the ground plane, $\text{Dist}(p_i, p_j)$ and $\text{Dist}(p_i, \mathcal{G})$ denote the minimum inter-primitive and primitive-to-ground distances, respectively, and δ_d and δ_g are the corresponding distance thresholds. Further, let $r_o, r_f \in [0, 1]$ denote the proportions of primitives violating the overlap and floating constraints, with larger values indicating more severe violations. The reward is defined as

$$R_{\text{physics}} = 2\left[1 - \tanh\left(\gamma\left(\lambda_o r_o + \lambda_f r_f\right)\right)\right] - 1, \tag{7}$$

where λ_o and λ_f denote the weights, and γ controls the sensitivity of the reward to violation severity.

• *Visual Quality Reward* measures the quality of the 3D object rendered from the generated HTML program. A pretrained VLM jointly interprets the target description and multi-view renderings, providing a fine-grained visual reward signal. For each generated result, we render four views and concatenate them into a single multi-view image, and candidates sampled for the same dense caption are scored jointly within one request to keep within-group comparisons consistent. The VLM evaluates the rendered object from two aspects and assigns scores in $[0, 10]$: (1) Semantic Score S_{sem}, which measures the consistency of the generated object and its key characteristics with the target description; and (2) Geometric Score S_{geo}, which evaluates the plausibility of the 3D structure, correctness of component connections, and overall proportions. Reward validity is assessed in Appendix A.3. The reward is computed as:

$$R_{\text{visual}} = (S_{\text{sem}} + S_{\text{geo}})/10 - 1. \tag{8}$$

Group Relative Policy Optimization. Building upon the above rewards, we optimize the SFT model using GRPO. Given a dense caption d, the current policy $\pi_{\theta_{\text{old}}}$ independently samples

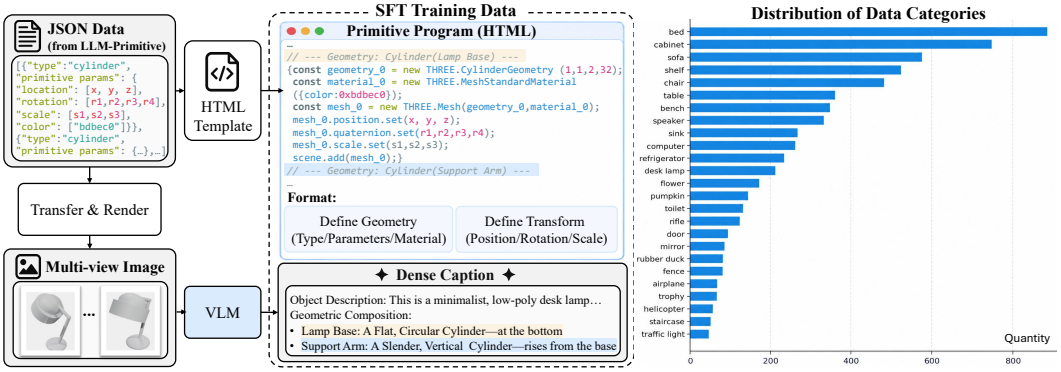


Figure 3: SFT data construction and category distribution. Left: Source primitive assemblies are converted from JSON into executable HTML programs using a predefined template and rendered from multiple viewpoints. A VLM generates dense captions from these renderings, which are paired with the corresponding programs for SFT. Right: The dataset exhibits an imbalanced distribution between frequent head categories and infrequent tail categories.

G candidate outputs $\{h^{(1)}, h^{(2)}, \ldots, h^{(G)}\}$, and the reward function evaluates them to obtain $\{r^{(1)}, r^{(2)}, \ldots, r^{(G)}\}$. The advantage of each sample is estimated by normalization within the group:

$$\hat{A}^{(i)} = \frac{r^{(i)} - \text{mean}(\{r^{(k)}\}_{k=1}^{G})}{\text{std}(\{r^{(k)}\}_{k=1}^{G})}. \tag{9}$$

The policy is then optimized by maximizing the following GRPO objective:

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{d\sim\mathcal{D},\, \{h^{(i)}\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}} \Bigg[\frac{1}{G}\sum_{i=1}^{G} \min \Bigg(\frac{\pi_\theta(h^{(i)} \mid d)}{\pi_{\theta_{\text{old}}}(h^{(i)} \mid d)} \hat{A}^{(i)}, \\ \text{clip}\left(\frac{\pi_\theta(h^{(i)} \mid d)}{\pi_{\theta_{\text{old}}}(h^{(i)} \mid d)}, 1-\epsilon, 1+\epsilon \right) \hat{A}^{(i)} \Bigg) - \beta D_{\text{KL}}\left(\pi_\theta \,\|\, \pi_{\text{ref}}\right) \Bigg], \end{aligned} \tag{10}$$

where π_θ denotes the updated policy, ϵ is the clipping coefficient that limits the policy update, and β controls the strength of the KL divergence penalty from a reference policy π_{ref}. By maximizing the above objective, the model reinforces high-quality generation behaviors while suppressing low-quality ones, enabling continuous self-improvement.

4.3 Data Construction

For SFT, we process the training data from LLM-Primitives (Tian et al., 2025). It contains 12,000 samples built from three geometric primitive types, i.e., cube, cylinder, and sphere, and covers approximately 25 object categories, with the distribution shown in Figure 3. Each sample stores the type, translation, rotation, scale, and color of its constituent primitives in JSON format. We convert these samples into executable HTML programs using a predefined template and render the corresponding 3D objects from multiple viewpoints. A VLM then generates dense captions from these images. Finally, we pair each dense caption with its corresponding HTML program to form the SFT dataset. For RL, we construct training data to support subsequent studies on data scaling and category adaptation. First, we select 10 categories from the SFT dataset, covering both head and tail categories, as in-domain categories. We use existing dense captions as in-context examples to prompt an LLM to generate additional dense captions. Second, we select 10 categories unseen during SFT as out-of-domain categories. Given the category names and representative examples, an LLM generates dense captions, followed by manual filtering for structural and semantic validity. In total, we obtain 8,000 training samples, with 400 samples per category in each subset.

5 Experiments

5.1 Experimental Settings

Dataset. We construct the training dataset following Section 4.3. For evaluation, we use the 20 object categories included in RL training: 10 in-domain categories seen during SFT and 10 out-of-

Table 1: Quantitative comparison of PrimLLM with proprietary and open-weight LLMs. *The parameter counts of proprietary models are not publicly disclosed.

Type	Method	Params. (B)	HTMLValid(%) ↑	CLIPScore↑	VQAScore↑	StructureScore↑	WinRate vs.Ours (%) ↑
Proprietary	GPT-5.5	*	88.00	27.22	0.7721	3.710	37.20
	Claude Sonnet 5	*	97.50	26.32	0.7730	3.954	38.49
	Gemini 3.8 Flash	*	99.50	27.42	0.7944	4.021	44.37
Open-weight	GLM-5.2	753	71.50	26.37	0.7459	3.870	36.25
	DeepSeek-V4-Pro	1600	87.00	26.44	0.7569	3.910	39.23
	Qwen3.5-27B	27	74.50	25.39	0.7158	3.296	9.32
	Qwen3.5-9B	9	71.50	25.12	0.7050	3.010	9.23
	PrimLLM(Ours)	9	**100.00**	**27.45**	**0.7951**	**4.109**	N/A

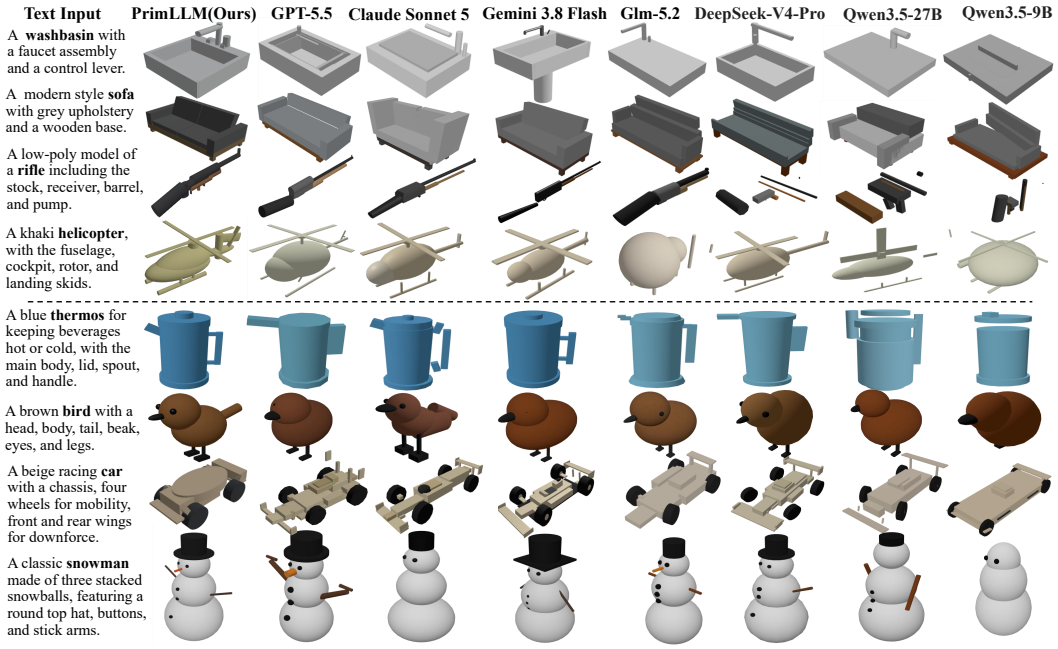


Figure 4: Qualitative comparison with proprietary and open-weight baselines on in-domain (top) and out-of-domain (bottom) categories. Some open-weight baselines struggle with complex assemblies, producing scattered components (e.g., rifles in the third row). Proprietary baselines capture finer details but sometimes lack coherent connections. In contrast, PrimLLM generates semantically consistent assemblies with more complete components and coherent spatial arrangements.

domain categories unseen during SFT. We independently generate 10 new textual descriptions per category, yielding 200 test samples evenly split between the two subsets for subsequent experiments.

Metrics. We evaluate generated assemblies along four dimensions: 1) program validity, quantified by HTML Valid (%), the percentage of programs that load and render correctly; 2) semantic alignment, assessed using CLIP and VQA Scores between input descriptions and multi-view renderings; 3) structural plausibility, assessed using a GPT-assigned Structure Score from 1 to 5; and 4) pairwise preference, reported as Win Rate vs. Ours (%), the percentage of comparisons in which GPT prefers a baseline's output to PrimLLM's. Win rates below 50% favor PrimLLM. CLIP, VQA, and Structure Scores are computed only for successfully rendered outputs. Details are provided in Appendix C.2.

5.2 Comparison

We compare PrimLLM with proprietary and open-weight LLMs. Proprietary baselines include GPT-5.5, Claude Sonnet 5, and Gemini 3.8 Flash, while the open-weight baselines include GLM-5.2, DeepSeek-V4-Pro, Qwen3.5-27B, and our base model, Qwen3.5-9B. All models are evaluated on the same test set, with baselines using default reasoning settings. For fairness, all models are instructed to generate executable programs using only cubes, cylinders, and spheres. Unsupported primitives are removed from all outputs before unified rendering and evaluation.

Quantitative Evaluation. Table 1 shows that PrimLLM performs best overall among the evaluated models. It improves CLIP, VQA, and Structure Scores over its Qwen3.5-9B base model by 9.28%, 12.78%, and 36.51%, respectively. Compared with Gemini 3.8 Flash, it achieves comparable CLIP

Table 2: Quantitative ablation of our training strategy: SFT establishes basic program-generation capabilities; Component Planning (C.P.) reduces ambiguity in the semantic-to-program mapping; RL refines semantic consistency and structural plausibility on in-domain and out-of-domain categories.

SFT	C.P.	RL	In-domain			Out-of-domain		
			HTMLValid (%) ↑	VQAScore ↑	StructureScore ↑	HTMLValid (%) ↑	VQAScore ↑	StructureScore ↑
×	×	×	72.00	0.7382	3.100	71.00	0.6718	2.920
✓	×	×	95.00	0.7821	3.864	94.00	0.6851	3.350
✓	✓	×	99.00	0.7985	4.021	95.00	0.6951	3.559
×	✓	✓	72.00	0.7369	3.756	78.00	0.7002	3.635
✓	✓	✓	**100.00**	**0.8248**	**4.290**	**100.00**	**0.7653**	**3.928**

Table 3: Ablation of reward: R_{format} promotes program validity; R_{physics} penalizes physically implausible primitives; R_{visual} improves both semantic consistency and structural plausibility.

Reward	HTML Valid (%) ↑	VQA. ↑	Structure. ↑
w/o R_{format}	98.50	0.7827	4.020
w/o R_{physics}	100.00	0.7873	3.894
w/o R_{visual}	100.00	0.7163	3.513
All	**100.00**	**0.7951**	**4.109**

Table 4: Effect of RL data scale and joint training on category adaptation. Single RL trains one model per category; Joint RL trains one model across $K = 20$ categories. n denotes training captions per model.

Setting	n	In-domain		Out-of-domain	
		VQA. ↑	Structure. ↑	VQA. ↑	Structure. ↑
Single RL	100	0.7940	3.907	0.6984	3.577
	200	0.7974	3.975	0.7100	3.585
	400	0.8042	4.155	0.7327	3.920
Joint RL	$K \times 400$	**0.8248**	**4.290**	**0.7653**	**3.928**

and VQA Scores but a higher Structure Score. All baselines have win rates below 50% against PrimLLM. These results suggest that Structure-Aware SFT and Annotation-Free RL yield a 9B model with competitive semantic alignment and improved program validity and structural plausibility.

Qualitative Evaluation. Figure 4 presents qualitative results for in-domain categories (top) and out-of-domain categories unseen during SFT (bottom). For in-domain objects, PrimLLM generates coherent assemblies that capture the specified components and their spatial arrangement, such as the washbasin with its faucet and control lever. For out-of-domain objects, it preserves details, including the bird's tail and the snowman's hat, buttons, and stick arms. In contrast, some baselines omit or misplace requested components. Qwen3.5-9B generates a thermos lid detached from the body and a snowman without a hat or arms. While proprietary models remain competitive on examples such as the racing car, PrimLLM shows advantages in component completeness and spatial coherence. More qualitative results are provided in Appendix C.3.

5.3 Ablation Study

We conduct ablation studies to evaluate the contributions of the training stages, component planning, and reward design in PrimLLM. The results are presented in Tables 2 and 3, and in Figure 5.

Effect of the Training Strategy. Table 2 compares SFT, Component Planning (C.P.), and RL configurations on in-domain and out-of-domain categories. The first three rows show that SFT establishes basic primitive-program generation capabilities, substantially improving HTML validity, while C.P. provides further gains. The last two rows highlight the importance of SFT initialization for RL. Without SFT, low initial output validity and unstable rewards hinder GRPO optimization. Comparing the third and last rows shows that RL further improves semantic consistency and structural plausibility. Notably, the model without RL performs well on in-domain categories but struggles with out-of-domain categories. RL improves performance on both subsets, with larger gains on out-of-domain categories, supporting adaptation beyond SFT categories.

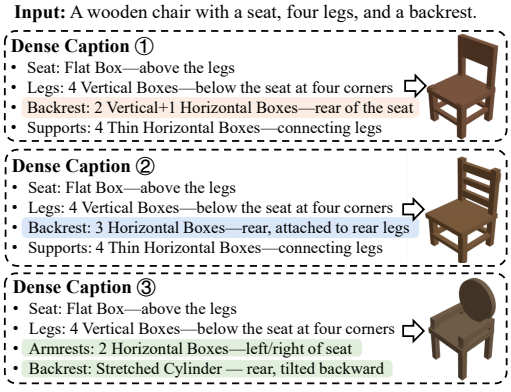


Figure 5: Different dense captions yield different primitive assemblies for the same text input.

Effect of Component Planning. Brief text descriptions often underspecify fine-grained object structure. Component planning infers object components and their spatial organization, providing explicit structural guidance for semantic-to-program mapping. Comparing the second and third rows of Table 2 shows that C.P. improves performance, suggesting that dense captions help reduce

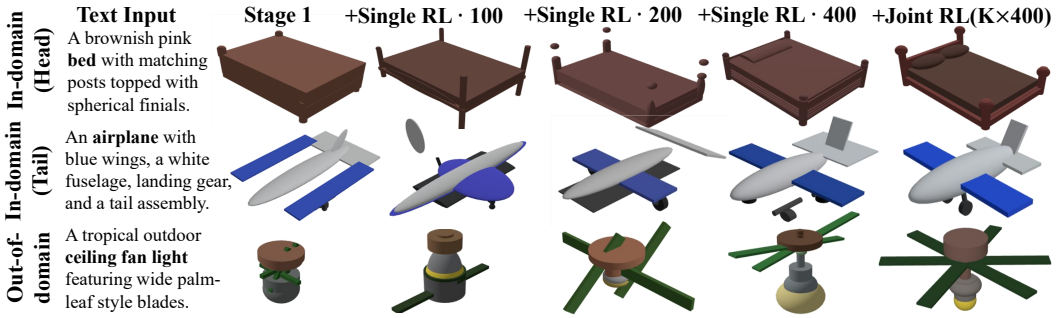


Figure 6: Qualitative comparison of category adaptation with varying RL data scales and training settings. Rows show in-domain head, in-domain tail, and out-of-domain categories, respectively. All RL variants start from the same Stage 1 model, which performs better on head categories than on tail and out-of-domain categories. More RL data generally improves assembly quality, especially for the latter two groups, while joint training further improves structural coherence in all groups.

ambiguity and establish more reliable mappings. Figure 5 shows that different dense captions for the same text input lead to corresponding changes in the generated geometry, illustrating the model's ability to follow component-level semantics for fine-grained control over primitive assembly.

Effect of Reward. Table 3 highlights complementary reward contributions. Removing R_{format} reduces HTML validity, while removing R_{physics}, which penalizes excessive overlap and floating, lowers the Structure Score. Removing R_{visual} decreases VQA and Structure Scores, suggesting that VLM feedback provides complementary guidance on semantic consistency and overall structure.

5.4 Category Adaptation Analysis

We study how RL data scale and joint training affect category adaptation to categories unseen during SFT. Experiments cover the same 10 in-domain and 10 out-of-domain categories, with all RL runs initialized from the same Stage 1 model and evaluated on the same test set.

Effect of RL Data Scale. We train a separate RL model for each category using 100, 200, or 400 dense captions and evaluate it on the test samples of that category. Table 4 reports results averaged separately over the in-domain and out-of-domain subsets, showing improved performance as the number of training captions increases. Figure 6 illustrates improvements in component completeness and spatial arrangement, especially for tail and out-of-domain categories. With 400 captions per category, the assemblies recover the main object structure in most examples. These results indicate that RL supports category adaptation with limited text data, without requiring additional annotations.

Single-Category vs. Joint Training. With 400 captions per category, we compare joint training across all 20 categories with single-category training. Table 4 and Figure 6 show that joint training performs better on both in-domain and out-of-domain categories. This reflect the benefit of learning shared assembly patterns from diverse components and spatial relations. The joint model is our final PrimLLM, used in the main comparisons and as the full configuration in the ablation studies.

6 Conclusion and Limitations

We presented PrimLLM, a two-stage framework that adapts a 9B language model for text-conditioned geometric primitive assembly with limited annotation. Structure-Aware SFT uses component planning to establish a semantic-to-program mapping, while Annotation-Free RL refines generated assemblies through format, physical, and visual rewards. Our experiments suggest that this combination improves semantic consistency and structural plausibility and supports adaptation to categories unseen during SFT without additional paired program annotations.

PrimLLM has several limitations. First, using only three primitive types makes certain fine geometric details and complex surfaces difficult to represent compactly. Second, our study focuses on external shape and visible components, without systematically evaluating the plausibility of internal functional structures, such as storage spaces and shelf layouts inside cabinets. Future work will explore richer primitive types and adaptive assembly granularity, as well as generation and evaluation methods that account for both internal and external structures and their functional requirements.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript drafting, translation, and language refinement, improving readability and clarity throughout the paper. As described in the paper, LLMs and VLMs were also used for component planning, visual reward computation, and evaluation. Details are provided in the Method, Experiment, and Appendix sections. Generative AI was used as an assistive tool rather than as an autonomous researcher. All AI-assisted content and outputs were reviewed by the authors. For example, AI-generated text was manually checked and revised to ensure accuracy and consistency. AI-assisted component planning and dense caption generation were also inspected. In addition, the AI-based visual reward was validated by measuring its agreement with human evaluations. The authors retain responsibility for all scientific decisions and conclusions, as well as for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide detailed descriptions of the proposed framework, data construction, training procedures, and experimental settings in the main paper. The appendix provides additional details on training configurations and hyperparameters, reward design, dataset structure, and evaluation protocols. We also conduct ablation studies to assess the contributions of individual components and the robustness of the training strategy. Upon acceptance, we will release the source code, model checkpoints, and constructed datasets to support transparent verification and further research on text-conditioned primitive assembly.

REFERENCES

Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, Jakob Engel, Edward Miller, Richard Newcombe, and Vasileios Balntas. Scenescript: Reconstructing scenes with an autoregressive structured language model. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 247–263, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73030-6.

Shuangkang Fang, I-Chao Shen, Yufeng Wang, Yi-Hsuan Tsai, Yi Yang, Shuchang Zhou, Wenrui Ding, Takeo Igarashi, and Ming-Hsuan Yang. Meshllm: Empowering large language models to progressively understand and generate 3d mesh. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14061–14072, October 2025.

Zhirui Gao, Renjiao Yi, Yuhang Huang, Wei Chen, Chenyang Zhu, and Kai Xu. Self-supervised learning of hybrid part-aware 3d representations of 2d gaussians and superquadrics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9649–9659, October 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Florian Kluger, Eric Brachmann, Michael Ying Yang, and Bodo Rosenhahn. Robust shape fitting for 3d scene abstraction. *IEEE transactions on pattern analysis and machine intelligence*, 46(9): 6306–6325, 2024.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *CoRR*, abs/2411.15124, 2024. doi: 10.48550/ARXIV.2411.15124. URL https://doi.org/10.48550/arXiv.2411.15124.

Songlin Li, Despoina Paschalidou, and Leonidas Guibas. Pasta: Controllable part-aware shape generation with autoregressive transformers. In *2026 International Conference on 3D Vision (3DV)*, pp. 1522–1533. IEEE, 2026.

Yuanqi Li, Shun Liu, Xinran Yang, Jianwei Guo, Jie Guo, and Yanwen Guo. Surface and edge detection for primitive fitting of point clouds. In *ACM SIGGRAPH 2023 conference proceedings*, pp. 1–10, 2023.

Youtian Lin, Yikang Yang, Zhanpeng Hu, Mengqi Zhou, Feihu Zhang, Xun Cao, Jiaheng Liu, and Yao Yao. Procedura: Agentic 3d modeling with procedural control, 2026. URL https://arxiv.org/abs/2608.26238.

Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In Ales Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part IX*, volume 15067 of *Lecture Notes in Computer Science*, pp. 366–384. Springer, 2024. doi: 10.1007/978-3-031-72673-6_20. URL https://doi.org/10.1007/978-3-031-72673-6_20.

Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S Chirikjian. Robust and accurate superquadric recovery: A probabilistic approach. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2666–2675. IEEE, 2022.

Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S Chirikjian. Marching-primitives: Shape abstraction from signed distance function. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8771–8780. IEEE, 2023.

Sining Lu, Guan Chen, Nam Anh Dinh, Itai Lang, Ari Holtzman, and Rana Hanocka. Ll3m: Large language 3d modelers, 2025. URL https://arxiv.org/abs/2508.08228.

Chengqi Lyu, Songyang Gao, Yuzhe Gu, Wenwei Zhang, Jianfei Gao, Kuikun Liu, Ziyi Wang, Shuaibin Li, Qian Zhao, Haian Huang, Weihan Cao, Jiangning Liu, Hongwei Liu, Junnan Liu, Songyang Zhang, Dahua Lin, and Kai Chen. Exploring the limit of outcome reward for learning mathematical reasoning. *ArXiv*, abs/2502.06781, 2025. URL https://api.semanticscholar.org/CorpusID:276250407.

Eric-Tuan Lê, Minhyuk Sung, Duygu Ceylan, Radomir Mech, Tamy Boubekeur, and Niloy J. Mitra. Cpfn: Cascaded primitive fitting networks for high-resolution point clouds. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7438–7446, 2021. doi: 10.1109/ICCV48922.2021.00736.

Tom Monnier, Jake Austin, Angjoo Kanazawa, Alexei Efros, and Mathieu Aubry. Differentiable blocks world: Qualitative 3d decomposition by rendering primitives. *Advances in Neural Information Processing Systems*, 36:5791–5807, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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)*, pp. 1253–1263, 2025.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

Kuan Tian, Zhihao Hu, Yonghang Guan, and Jun Zhang. Llm-primitives: Large language model for 3d reconstruction with primitives. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, pp. 1–12, 2025.

Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1466–1474. IEEE, 2017.

Puyi Wang, Yuhao Wang, Linjie Li, Zhengyuan Yang, Kevin Qinghong Lin, Yangguang Li, and Yu Cheng. Scenecode: Executable world programs for editable indoor scenes with articulated objects. *arXiv preprint arXiv:2605.19587*, 2026a.

Rui-Huan Wang, Si-Tong Wei, Jia-Qi He, Heng-Yi Wei, Baoquan Chen, and Peng-Shuai Wang. adsl: Agentic 3d creation via joint agent-program design. *arXiv preprint arXiv:2608.17975*, 2026b.

Zhengyi Wang, Jonathan Lorraine, Yikai Wang, Hang Su, Jun Zhu, Sanja Fidler, and Xiaohui Zeng. Llama-mesh: Unifying 3d mesh generation with language models. *arXiv preprint arXiv:2411.09595*, 2024.

Jingwen Ye, Yuze He, Yanning Zhou, Yiqin Zhu, Kaiwen Xiao, Yong-Jin Liu, Wei Yang, and Xiao Han. Primitiveanything: Human-crafted 3d primitive assembly generation with auto-regressive transformer. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–12, 2025.

Lingting Zhu, Shengju Qian, Haidi Fan, Jiayu Dong, Zhenchao Jin, Siwei Zhou, Gen Dong, Xin Wang, and Lequan Yu. Assetformer: Modular 3d assets generation with autoregressive transformer. *CoRR*, abs/2602.12100, 2026. doi: 10.48550/ARXIV.2602.12100. URL https://doi.org/10.48550/arXiv.2602.12100.

A Details of Our Framework

A.1 Primitive Program Representation

We represent primitive assemblies as executable HTML programs built on Three.js. HTML provides the runtime scaffold, while JavaScript defines each primitive's geometry, appearance, and spatial transformation, placing the primitives in a shared scene to form a complete object. This representation preserves individual primitive attributes, facilitating program generation, rendering, and subsequent checks.

Primitive Instance Definition. As defined in the main paper, each primitive is represented as $p_i = (\tau_i, c_i, \mathbf{t}_i, \mathbf{r}_i, \mathbf{s}_i)$, specifying its type, color, translation, rotation, and scale. We use three primitive types: cube, defined by `THREE.BoxGeometry(2, 2, 2)`; cylinder, by `THREE.CylinderGeometry(1, 1, 2, 32)`; and sphere, by `THREE.SphereGeometry(1, 32, 32)`. We then specify material properties and color through `THREE.MeshStandardMaterial` and combine the geometry and material using `THREE.Mesh` to construct each instance. We set its translation and quaternion rotation and adjust its axis-wise scale to approximate different shapes before adding it to the scene through `scene.add`. Final dimensions depend on both the base geometry and axis-wise scaling. Note that semantic components and geometric primitives are not in one-to-one correspondence: a single component may be represented by several primitives.

Program Template. Figure 7 shows the HTML program template. The blue region contains environment initialization code provided as a fixed prefix and excluded from generation. The middle orange region defines primitive instances, while the gray region contains the rendering loop and closing code. By integrating primitive construction with scene initialization, camera, lighting, and rendering code, the program can run directly and be viewed interactively in a browser with the Three.js dependencies configured.

A.2 Component Planning Process

Given a brief text description, the component planning module prompts an external LLM to identify the object's components and describe their geometric shapes, approximate dimensions, and spatial relations. The resulting dense caption provides explicit component-level guidance for primitive-program generation. Figure 9 presents the full prompt used for component planning.

A.3 Reward Configuration and Validity Analysis

Reward Configuration. Our reward function combines format validity (R_{format}), physical plausibility (R_{physics}), and VLM-based visual quality (R_{visual}), with $\lambda_{\text{format}} = 0.3$, $\lambda_{\text{physics}} = 0.1$, and $\lambda_{\text{visual}} = 0.6$ during RL training. In reward ablations, we remove the selected component and renormalize the remaining weights to sum to one while preserving their relative proportions. In the default format-reward configuration, $\lambda_{\text{syntax}} = 0.2$ and $\lambda_{\text{prim}} = 0.8$ are the weights for HTML syntax and primitive-specific checks, respectively. In the default physical-plausibility configuration, $\lambda_o = 0.6$ and $\lambda_f = 0.4$ correspond to overlap and floating penalties, respectively, and $\gamma = 4.0$ controls the sensitivity of the tanh mapping.

Details of Visual Quality Reward. The VLM evaluates each assembly's semantic consistency and geometric plausibility, assigning a Semantic Score S_{sem} and a Geometric Score S_{geo}, both in $[0, 10]$. For each dense caption, we combine the four rendered views of each candidate into a multi-view image and present all candidate images together with the caption in a single VLM request. The VLM assigns both scores to every candidate jointly rather than evaluating candidates in isolation. This shared comparison context is intended to promote more consistent scoring within the group. Figure 11 provides the full evaluation prompt.

Reward Trends across Checkpoints. We track the average format, physical, and visual rewards of assemblies generated by model checkpoints at different training steps. As shown in Figure 8, the three rewards generally increase and then level off as training progresses, indicating improvements under the corresponding criteria and diminishing gains at later steps.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body { margin: 0; overflow: hidden; background-color: #f0f0f0; }
        canvas { display: block; }
    </style>
    <script type="importmap">
        {"imports": {"three":"/html_tool/libs/three.module.js","three/addons/":"/html_tool/libs/"}}
    </script>
</head>
<body>
    <script type="module">
        import * as THREE from 'three';
        import { OrbitControls } from 'three/addons/controls/OrbitControls.js';

        // ============ 1. Scene Init ============
        const scene = new THREE.Scene();
        scene.background = new THREE.Color(0xf0f0f0);
        const camera = new THREE.PerspectiveCamera(45,
                       window.innerWidth / window.innerHeight, 0.01, 200);
        camera.position.set(3.000, 3.000, 3.000);
        const renderer = new THREE.WebGLRenderer({ antialias: true });
        renderer.setSize(window.innerWidth, window.innerHeight);
        renderer.setPixelRatio(window.devicePixelRatio);
        document.body.appendChild(renderer.domElement);

        // ============ 2. Controls ============
        const controls = new OrbitControls(camera, renderer.domElement);
        controls.target.set(0.000, 0.000, 0.000);
        controls.update();

        // ============ 3. Lighting ============
        const ambientLight = new THREE.AmbientLight(0xffffff, 0.6);
        scene.add(ambientLight);
        const dirLight = new THREE.DirectionalLight(0xffffff, 1.5);
        dirLight.position.set(5, 10, 7);
        scene.add(dirLight);

        // ============ 4. Build Primitives ============
        // --- Geometry: cylinder (Lamp Base) ---
        {
           const geometry_0 = new THREE.CylinderGeometry(1, 1, 2, 32);
           const material_0 = new THREE.MeshStandardMaterial({color: 0xbdbec0});
           const mesh_0 = new THREE.Mesh(geometry_0, material_0);
           mesh_0.position.set(-0.092, 0.363, -0.0);
           mesh_0.quaternion.set(-0.046, 0.037, -0.294, 0.954);
           mesh_0.scale.set(0.36, 0.14, 0.356);
           scene.add(mesh_0);
        }
        …

        // ============ 5. Render Loop ============
        function animate() {
            requestAnimationFrame(animate);
            controls.update();
            renderer.render(scene, camera);
        }
        animate();
    </script>
</body>
</html>
```

Environment Initialization (Fixed Prefix)

Primitive Instance Construction

Rendering and Closing

Figure 7: HTML program template. The blue region initializes the environment and serves as a fixed prefix. The orange region defines primitive instances, while the gray region contains the rendering loop and closing code. The model generates the orange and gray sections.

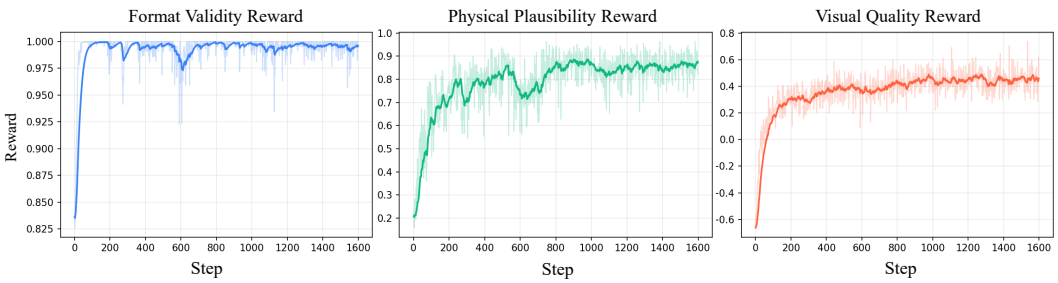


Figure 8: Trends in format, physical, and visual rewards across RL training checkpoints.

Table 5: Within-group agreement between the visual quality reward and human judgments across object complexity levels.

	Visual Quality Reward vs. Human		
Object Complexity	**Spearman $\rho \uparrow$**	**Kendall $\tau \uparrow$**	**Top-1 Accuracy $\uparrow$**
Simple	0.6474	0.5268	0.75
Medium	0.5851	0.4845	0.65
Complex	0.5197	0.4495	0.65
Overall Average	0.5841	0.4869	0.68

Within-Group Reward Validity Analysis. We validate the within-group effectiveness of the visual quality reward R_{visual} using 30 groups of candidate primitive assemblies, with 10 groups each for simple, medium, and complex objects. Complexity levels are assigned coarsely based on the number of semantic components. Each group contains 8 candidates sampled through policy rollouts conditioned on the same dense caption. Three human evaluators jointly rank the candidates within each group. We measure agreement between rankings induced by R_{visual} and the human rankings using Spearman's rank correlation (ρ) and Kendall's rank correlation (τ). We also report Top-1 Accuracy as the proportion of groups with a matching top choice; to account for ties, a hit is counted whenever the two top-ranked sets share at least one candidate. We aggregate results across two evaluation rounds. Table 5 reports overall correlations of $\rho = 0.5841$ and $\tau = 0.4869$, with a Top-1 Accuracy of 0.68. These results indicate that the visual quality reward can identify human-preferred candidates within rollout groups, supporting its use as a relative quality signal for RL.

B Details of Data Construction

For SFT, we follow Section 4.3 to convert JSON primitive assemblies from LLM-Primitives into executable HTML programs. We render each object from multiple viewpoints and provide the images to a VLM to generate a dense caption describing its components, geometric attributes, and spatial relations. Figure 10 presents the full caption-generation prompt.

Each training sample is organized as a sequence of messages: the system message specifies generation instructions, and the user message contains a dense caption and the fixed HTML prefix. SFT samples additionally include an assistant message containing the ground-truth program continuation, from primitive construction through rendering and closing code. Table 6 illustrates the format using an adapted SFT sample.

C Details of Our Experiments

C.1 Implementation

Unless otherwise specified, we adopt Qwen3.5-9B as the base LLM and Qwen3.5-35B-A3B as the VLM-based reward model for evaluating the visual quality reward in the RL stage. Both SFT and GRPO use LoRA (Hu et al., 2022) with a rank of 8 and an alpha of 32. For SFT, we train for 3 epochs with a learning rate of 1×10^{-4}, taking approximately 6 GPU hours. For GRPO, we use a learning rate of 2×10^{-5}, generate 8 candidate responses per input, and set the KL coefficient to 0.005, with the complete RL training taking approximately 72 GPU hours. We additionally employ HY3 as the external LLM for component planning and HY-Vision-2.0-Instruct as the VLM for dense caption generation during data construction.

C.2 Evaluation

We evaluate the generated results from four aspects: program validity, semantic alignment, structural plausibility, and pairwise preference.

Program Validity. HTML Valid (%) measures the percentage of generated HTML programs that load and render correctly. A higher value indicates more reliable program generation.

Semantic Alignment. We use CLIP Score (Radford et al., 2021) and VQA Score (Lin et al., 2024) to assess alignment between input descriptions and multi-view renderings. CLIP Score uses the ViT-B-32 model to measure global semantic similarity, computed as the cosine similarity between text and image embeddings multiplied by 100. VQA Score uses CLIP-FlanT5-XXL to estimate the probability of an affirmative answer to a question asking whether the rendered image matches the description, providing a finer-grained assessment of semantic consistency. Higher scores indicate better text–image alignment.

Structural Plausibility. We use GPT-5.5 to assign a Structure Score from 1 to 5 based on multi-view renderings. The assessment considers geometric plausibility, component connections, spatial arrangement, and overall proportions. Higher scores indicate more coherent and plausible structures. Figure 12 provides the detailed scoring prompt.

Pairwise Preference. We use GPT-5.5 to make a forced choice between PrimLLM and a competing method on pairs where both outputs render successfully, using their multi-view renderings and the corresponding text description to assess semantic consistency, component completeness, and structural coherence. Win Rate vs. Ours (%) is the percentage of evaluated pairs in which the competing method is preferred. A higher value indicates better performance of the competing method, whereas a value below 50% indicates better performance of PrimLLM. Figure 13 presents the prompt used for pairwise evaluation.

C.3 More Qualitative Results

Figure 14 presents additional qualitative comparisons of our method with GPT-5.5, Claude Sonnet 5, Gemini 3.8 Flash, GLM-5.2, DeepSeek-V4-Pro, Qwen3.5-27B, and Qwen3.5-9B. Figure 15 shows PrimLLM's generated results across more object categories.

Table 6: SFT message format illustrated with a desk-lamp sample.

Role	Content
system	You are a senior 3D web development expert and creative coder, highly proficient in Three.js. You specialize in low-poly modeling using basic geometric primitives. **Task Description:** Based on the input [Object Description to be Created], reconstruct the object using only three geometric primitive types (Cube, Sphere, and Cylinder) to produce a preview-ready HTML file. The [Three.js Template Code] has already been set up for you; you are required to generate the primitive building code (starting from `// ============ 4. Build Primitives ============`). **Requirements:** **1. Structural Decomposition:** Break down the object into its simplest geometric components, using ONLY basic primitives. **2. Coordinate System:** Use standard 3D coordinates (Y-axis up). Ensure correct positioning, rotation, and scaling of all parts. **3. Code Structure:** Each primitive must be defined as an independent block with explicit: - `geometry` creation - `material` with color - `mesh` creation - `mesh.position.set(x, y, z)` - `mesh.scale.set(sx, sy, sz)` - `mesh.quaternion.set(x, y, z, w)` - `scene.add(mesh)` *[Output-format instructions omitted.]*
user	**Object Description to be Created:** Object Description: This object is a minimalist, low-poly desk lamp. The main functional units include the lamp base for stability, a vertical support arm, a horizontal arm connecting to the lamp head, a cylindrical lamp shade, and a dome-shaped bulb housing. Geometric Composition: 1. Lamp Base: A flat, circular Cylinder forms the stable base of the lamp. It is positioned at the bottom, centrally aligned and colored solid white. *[Components 2–5 and the final appearance description omitted.]* **Three.js Template Code (Already Provided):** *[Fixed HTML prefix omitted.]*

assistant	`// ============ 4. Build Primitives ============`
	`// --- Geometry: cylinder ---`
	`{`
	`const geometry_0 = new THREE.CylinderGeometry(1, 1, 2, 32);`
	`const material_0 = new THREE.MeshStandardMaterial({`
	`  color: 0xffffff,`
	`});`
	`const mesh_0 = new THREE.Mesh(geometry_0, material_0);`
	`mesh_0.position.set(-0.092, 0.363, -0.0);`
	`mesh_0.quaternion.set(-0.046, 0.037, -0.294, 0.954);`
	`mesh_0.scale.set(0.36, 0.14, 0.356);`
	`scene.add(mesh_0);`
	`}`
	[Remaining four primitive instances omitted.]
	`// ============ 5. Render Loop ============`
	`function animate() {`
	`  requestAnimationFrame(animate);`
	`  controls.update();`
	`  renderer.render(scene, camera);`
	`}`
	`animate();`
	`</script></body></html>`

Prompt for Component Planning

Task Description:
Given a brief text description, infer the object's components and their geometric organization. Produce a structured dense caption to guide a separate model in generating a coherent primitive assembly.

Instructions:
1. Object Description: Summarize the overall form and appearance of the object. Identify its main functional components.
2. Component Decomposition: Identify the components needed to represent the object. Use only Box (cube), Sphere, and Cylinder primitives. A component may comprise multiple primitives; specify their counts and arrangement.
3. Geometry and Appearance: Describe each component's primitive shapes, colors, approximate dimensions or proportions, and orientation. Allow non-uniform scaling, such as an ellipsoid formed from a sphere or a thin plate formed from a box.
4. Spatial Organization: Specify relative positions and explain how components connect, support, or align with one another. Keep dimensions and spatial relationships consistent throughout the assembly.
5. Reasonable Completion: When structural details are unspecified, infer a simple, plausible configuration consistent with the description. Provide one coherent plan without adding unrelated components or unnecessary decoration.

Output Format: Return only the following two sections.

Object Description:
[A concise description of the object's overall form, appearance, and main components.]

Geometric Composition:
1. [Component Name]: [Primitive type(s) and count, color, approximate dimensions or proportions, orientation, and spatial relationships with other components.]
2. [Component Name]: [...]
Continue for all planned components.

Input: {TEXT DESRIPTION}

Figure 9: Prompt for component planning.

Prompt for SFT Dataset Construction

Task Description:
Analyze the object in the provided multi-view images and produce a structured dense caption describing its components and geometric composition. All images show the same object from different viewpoints.

Instructions:
1. Object Description: Briefly describe the object, its overall appearance, and its main functional components.
2. Component Decomposition: Identify each visible component and the primitives needed to approximate it. Use only Box (cube), Sphere, and Cylinder. A component may contain multiple primitives; specify their counts when identifiable.
3. Geometry and Appearance: Describe the primitives' relative positions within each component and how components connect, support, or align with one another. Integrate evidence across views to avoid counting the same part multiple times.
4. Spatial Organization: Describe the primitives' relative positions within each component and how components connect, support, or align with one another. Integrate evidence across views to avoid counting the same part multiple times.
5. Visual Grounding: Base the description on visible evidence. Do not invent hidden components or unsupported structural details. Use approximate proportions rather than exact measurements.

Output Format: Return only the following two sections.

Object Description:
[A concise description of the object's overall form, appearance, and main components.]

Geometric Composition:
1. [Component Name]: [Primitive type(s) and count, color, approximate dimensions or proportions, orientation, and spatial relationships with other components.]
2. [Component Name]: [...]
Continue for all planned components.

Example:[EXAMPLE]

Input: {TEXT DESRIPTION}

Figure 10: Prompt for dense caption generation from multi-view renderings for dataset construction.

Prompt for Visual Quality Reward

You are a professional 3D model evaluator. Evaluate the following 3D models using the criteria below.

Task Description:
- There are {num_models} models to evaluate, numbered from 1 to {num_models}.
- Each model is presented as a composite grid image containing renderings from {num_views} different viewpoints.
- Carefully inspect each grid and consider all views together to evaluate the model as a whole.
- Score every model against the same target object description.

Object Description: {object_description}

Evaluation Criteria
Assign a score from 0 to 10 for each criterion.
1. Semantic Consistency
How accurately does the generated model represent the Object Description?
- 10: Fully matches the description; all key features are correct.
- 7–9: Largely matches the description, with minor deviations.
- 4–6: Partially matches the description but lacks important features.
- 1–3: Substantially deviates from the description.
- 0: Completely unrelated to the description.
2. Geometric Plausibility
Is the model's 3D structure plausible? Are components correctly connected and reasonably proportioned?
- 10: Structurally sound, with well-balanced proportions.
- 7–9: A plausible structure with minor flaws.
- 4–6: A generally plausible structure with noticeable problems.
- 1–3: Severe structural problems.
- 0: No recognizable structure.

Output Requirements
- Provide scores for ALL {num_models} models.
- Include every model ID from "1" to "{num_models}" as a top-level JSON key.
- Do not omit any model, use ellipses or other omission markers, or end the output before all models have been scored.
- For each model, include:
  - "semantic_score": a numeric score from 0 to 10.
  - "geometric_score": a numeric score from 0 to 10.
  - "brief_reason": one sentence explaining the main strengths and weaknesses.
- Return only a valid JSON object, without Markdown fences or additional text.
The following illustrates the structure of one entry. Replace "<model_id>" with the actual model ID and include an entry for every model:
{ "<model_id>": { "semantic_score": <score from 0 to 10>, "geometric_score": <score from 0 to 10>,
"brief_reason": "<One sentence explaining the main strengths and weaknesses.>"}}

Figure 11: Prompt for visual quality reward evaluation.

Prompt for Structure Score Evaluation

You are a professional 3D model evaluator specializing in objects assembled from geometric primitives.

Task Description:
The provided 2×2 image grid shows four views of the SAME 3D model.
Evaluate the model by considering all four views together.

Evaluation Criteria:
Assess component completeness, proportions, spatial relationships, and structural plausibility:
1. Component Completeness
Are the essential components present? Are any clearly necessary components missing?
2. Proportions
Are the relative sizes and dimensions of components reasonable and balanced?
3. Spatial Relationships
Are component positions, orientations, and connections consistent with basic physical and structural expectations?
4. Structural Defects
Check for unsupported floating parts, severe interpenetration, misalignment, unintended asymmetry, incorrect orientations, or disconnected assemblies.
5. Multi-view Consistency
The four views should depict a coherent geometric structure without contradictory evidence across viewpoints.

Scoring Rubric:
Assign one integer score from 1 to 5, where 1 is very poor and 5 is very good.
5 — Very good: Essential components are complete, proportions are balanced, and spatial relationships, support, and alignment are coherent. No apparent floating parts, interpenetration, or misalignment are visible. All four views consistently support a structurally plausible assembly.
4 — Good: The main structure is plausible, with only minor proportional errors, small positional offsets, or slight overlaps. The overall assembly remains coherent and appears stable.
3 — Fair: The main structure is recognizable but contains several noticeable issues, such as one or two disproportionate components, misalignment, minor floating parts, or obvious interpenetration. The intended assembly remains understandable.
2 — Poor: Multiple components have incorrect proportions, positions, or intersections. Several essential parts are missing, or support relationships are incorrect. The assembly appears structurally implausible or incorrectly constructed.
1 — Very poor: The geometry is severely distorted, with inconsistent proportions, extensive interpenetration or floating parts, or largely disconnected components. Also assign 1 to blank images or results without meaningful geometric structure.

Output Format:
Return only a valid JSON object, with no Markdown fences or additional text.
Replace the score placeholder with an integer from 1 to 5.
Provide a one-sentence explanation grounded in visible geometric evidence.
{"scores": {
"geometric_rationality": <integer from 1 to 5>
},
"brief_reason": "<One sentence describing the main geometric strengths and defects, including completeness, proportions, spatial relationships, interpenetration, or floating parts.>"}

Figure 12: Prompt for structure score evaluation from multi-view renderings.

Prompt for Pairwise Preference Evaluation

You are a professional 3D model evaluator conducting a pairwise comparison.

Task Description:
The comparison image contains two side-by-side 2×2 grids, labeled A (left) and B (right).
- Each grid shows four views of the SAME 3D model:
  - Top-left: isometric_bl
  - Top-right: isometric_br
  - Bottom-left: isometric_fl
  - Bottom-right: isometric_fr
- A and B are two 3D models generated by different methods for the SAME dense caption.

Determine which model is better overall by jointly considering:
1. Semantic Consistency
Alignment with the dense caption, including object category, components, and key attributes.
2. Geometric Rationality
Component completeness, reasonable proportions and spatial relationships, and the absence of severe interpenetration or unsupported floating parts.
3. Structural Quality
Overall structural coherence, consistency across views, appropriate symmetry, and whether the object appears complete and well assembled.

Use the following caption as the required semantic reference:
{object_description}

Decision Rules:
- Choose the better model: "A_better" or "B_better".
- Give equal importance to all three criteria. Do not declare an overall winner based solely on a slight advantage in one criterion; weigh the strengths and weaknesses together.
- Do not favor A or B because of its position on the left or right. Judge only the model content.

Output Format:
Return only a valid JSON object, with no Markdown fences or additional text.
The "winner" value must be exactly "A_better", "B_better".
Explain the main reasons in no more than two sentences.
{
  "winner": "<A_better or B_better>",
  "reason": "<Main reasons for the decision, in no more than two sentences.>"
}

Figure 13: Prompt for pairwise preference evaluation between PrimLLM and competing methods.

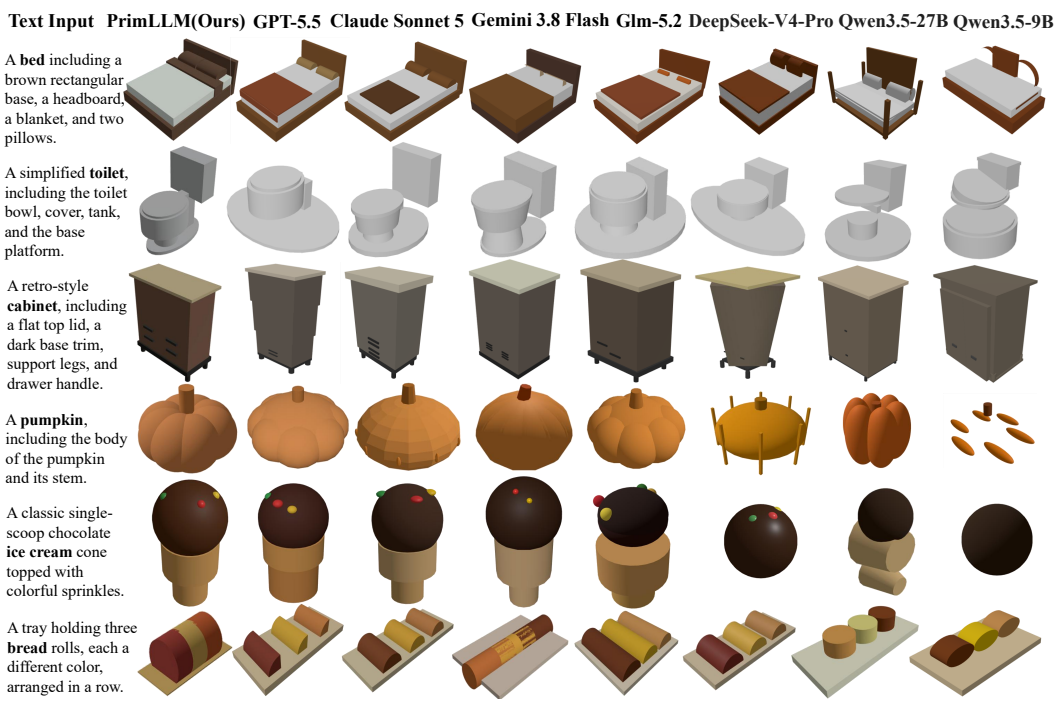


Figure 14: Additional qualitative comparison results generated from different methods.



Figure 15: Additional primitive assemblies generated by PrimLLM across diverse object categories.