BrickAnything: Geometry-Conditioned Buildable Brick Generation with Structure-Aware Tokenization

Zhengyang Ni[1,2,*] **Feng Yan**[1,2,*] **Yu Guo**[1,2] **Fei Wang**[1,2,†]

[1]State Key Laboratory of Human-Machine Hybrid Augmented Intelligence
[2]Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University

Abstract

Generating physically buildable brick structures from 3D shapes requires more than geometric reconstruction: the output must also satisfy discrete part constraints and structural stability. Existing brick generation methods either rely on heuristic optimization, which can break down when the target 3D shape does not admit a feasible structure under predefined constraints, or generate brick sequences without explicitly modeling the underlying 3D geometry and assembly relations. In this work, we present **BrickAnything**, a geometry-conditioned autoregressive framework for generating buildable brick structures from diverse 3D representations. BrickAnything uses point clouds as a unified geometric interface and predicts brick sequences that reconstruct the target shape under assembly constraints. To model structural dependencies among bricks, we introduce a structure-aware tree tokenization, which represents brick structures through local attachment relations. This formulation makes sequence generation more consistent with the physical construction process, and reduces invalid intermediate states. We further introduce preference-based alignment post-training, validity-constrained decoding and adaptive rollback to improve buildability objectives such as stability and geometric fidelity. Extensive experiments demonstrate that BrickAnything produces geometrically faithful and physically realizable brick structures, and that the proposed tokenization effectively reduces rollback and regeneration compared with conventional ordering strategies. Our code is available at `https://github.com/xjtunzy/BrickAnything`.

1 Introduction

Discrete brick-based structures offer a physically grounded and modular representation for constructing 3D objects, bridging digital geometry and real-world assembly. Unlike conventional 3D representations such as mesh[1, 2, 3, 4, 5, 6], point cloud[7, 8, 9, 10, 11, 12], Gaussian Splatting[13, 14, 15, 16], or implicit fields[17, 18, 19, 20], brick assemblies are governed by strict combinatorial and structural constraints. A generated brick structure must not only approximate the target geometry, but also satisfy valid part compatibility, inter-brick connectivity, and structural stability. This combination of geometric fidelity and physical buildability makes brick generation a fundamentally different and more challenging problem than conventional 3D content synthesis.

Existing methods for brick structure generation can be broadly categorized into generative approaches and heuristic or search-based methods. Recent generative models, such as BrickGPT[21], LEGO®-Maker[22] and BrickNet[23], formulate brick construction as a sequential prediction problem, enabling flexible and expressive generation of complex assemblies. However, these methods typically rely on high-level inputs such as text or images, which do not provide explicit 3D geometric constraints. As a result, the generated structures are primarily guided by semantic priors or 2D visual

[*]Equal contribution.
[†]Corresponding author.

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

cues, making it difficult to ensure accurate spatial configurations and faithful reconstruction of target shapes, especially for geometrically complex objects. On the other hand, heuristic or search-based methods, including Image2Lego[24] and Legolization[25], explicitly incorporate geometric and structural constraints by formulating brick assembly as a combinatorial optimization problem. While these methods can produce physically feasible structures, they often rely on fixed target representations and heuristic or search-based optimization strategies. This limits their flexibility when the target geometry is not directly realizable in the discrete brick space, and leads to high computational complexity as the search space grows. This raises a natural question: **Can we design a unified framework for brick generation that explicitly leverages 3D geometry, adapts to diverse input representations, and generates buildable structures in a more flexible and effective manner?**

To address this question, we propose **BrickAnything**, a unified framework for geometry-conditioned buildable brick generation. We adopt point clouds as a modality-agnostic intermediate representation, allowing diverse 3D inputs to be mapped into a common geometric interface. Point clouds provide explicit spatial information while being easily derived from a wide range of 3D modalities, making them a natural interface for geometry-conditioned generation.

Conditioned on this explicit 3D geometry, we formulate brick construction as an autoregressive sequence generation problem. A key component of BrickAnything is a structure-aware tree tokenization scheme. Instead of representing brick assemblies only as spatially ordered sequences, our tokenization organizes bricks according to local attachment relations. This representation better reflects the physical construction process, produces more compact sequences, and makes structural dependencies between bricks explicit during generation.

Building on this tokenization, we further propose a buildability-aware reward that jointly measures structural stability and geometric fidelity. This reward is used to construct preference pairs for Direct Preference Optimization (DPO), enabling the model to optimize buildability-related objectives that are difficult to capture with likelihood-based training alone. During inference, we also introduce a lightweight parent-aware rollback strategy tailored to the proposed tree tokenization. When an unstable brick is detected, the generation process rolls back to its parent brick in the tree sequence and regenerates the subsequent structure. This simple mechanism effectively improves the robustness of autoregressive brick generation.

Through extensive experiments, we demonstrate that BrickAnything improves both geometric fidelity and buildability compared with conventional brick generation baselines. In particular, our structure-aware tokenization explicitly preserves local attachment relations and substantially reduces rollback and regeneration during inference. We summarize our contributions as follows:

- We introduce **BrickAnything**, a unified geometry-conditioned framework for buildable brick generation. By using point clouds as a modality-agnostic intermediate representation, BrickAnything enables explicit 3D-guided generation from diverse input modalities.
- We propose a structure-aware tree tokenization scheme for brick structures. Instead of relying only on global spatial ordering, our representation organizes bricks according to local attachment relations, explicitly modeling parent-child assembly dependencies during autoregressive generation.
- We propose a buildability-aware reward that jointly measures structural stability and geometric fidelity, and use it to construct preference pairs for Direct Preference Optimization (DPO), enabling the model to optimize buildability-related objectives beyond likelihood-based training.
- We design a lightweight parent-aware rollback strategy tailored to the proposed tree tokenization. When an unstable brick is detected, the generation process rolls back to its structurally related parent and regenerates the subsequent substructure, improving inference robustness.

2 Related work

2.1 Brick Structure Generation

Generating brick structures from a reference 3D shape has been widely studied in prior work[25, 26]. Early approaches formulate this task as a combinatorial construction problem guided by handcrafted heuristics or search strategies. These methods typically enforce constraints such as structural connectivity, stability, and efficient brick usage, and often rely on sequential planning or heuristic search to assemble bricks that reconstruct a target geometry. While effective in producing

physically feasible designs, they generally assume that the input shape can be directly realized using discrete bricks. As a result, they fail when there is no feasible assembly solution for the given brick set — i.e., when the target geometry cannot be constructed under the prescribed brick types, connection rules, and stability constraints. Moreover, the reliance on heuristic rules or search-based strategies leads to high computational costs and limited scalability.

Recent learning-based methods have explored generative brick modeling with autoregressive models[21, 23, 22]. BrickGPT[21] generates physically stable and buildable brick structures from text prompts by predicting bricks sequentially, while enforcing validity and physical constraints during inference. BrickNet [23] further extends autoregressive brick generation by introducing a graph-backed connectivity representation and a large-scale human-designed LDraw dataset, enabling the modeling of more diverse part types and connection semantics. LEGO®-Maker[22] further explores image-conditioned LEGO® model generation using an autoregressive framework with multiple brick types, but its experiments mainly focus on a limited set of object categories. Despite these advances, existing learning-based methods are mainly conditioned on text descriptions or 2D visual observations, which do not provide explicit and unambiguous 3D geometric constraints. As a result, they may struggle to faithfully reconstruct complex target geometries or generate structurally valid brick structures that are tightly aligned with an input 3D shape. This motivates a unified geometry-conditioned formulation that directly uses 3D shape information while preserving the buildability constraints of brick assemblies.

2.2 Autoregressive Models for 3D Generation

Autoregressive models have become an effective paradigm for modeling structured data by factorizing generation into sequential token prediction[27]. Following their success in language[28, 29] and image generation[30, 31, 32], this paradigm has also been explored for 3D generation across different representations, including point cloud[33, 34], Gaussian splatting[35, 36], and mesh[37, 38]. Brick-based generation can also be naturally formulated as a sequential prediction problem, since a brick assembly of brick structures consists of discrete elements with spatial, topological, and physical dependencies. Recent methods[21, 22, 23] serialize brick assemblies into token sequences and generate brick structures step by step. In this work, we introduce a geometry-conditioned autoregressive framework that uses point clouds as explicit 3D guidance and organizes brick assemblies according to local attachment relations, enabling more accurate and physically buildable generation.

2.3 RLHF with Direct Preference Optimization

Aligning model outputs with human preferences is a critical step in improving generative models, especially for large language models (LLMs)[39, 40, 41]. Techniques like reinforcement learning from human feedback (RLHF) and direct preference optimization (DPO) have become standard for incorporating preference signals. While early RLHF methods using PPO[42] often suffer from instability and high computational cost, DPO[43] eliminates the need for an explicit reward model and directly optimizes policy likelihood. Recently, preference-based optimization has also been applied to structured generation tasks, including 3D shape generation[4, 44, 45]. In this work, we adapt DPO to brick structure generation by constructing preference pairs with a buildability-aware reward that jointly considers geometric fidelity and structural stability.

3 Method

BrickAnything comprises three components, as illustrated in Figure 1. Following BrickGPT [21], we construct brick structures within a $20 \times 20 \times 20$ voxel grid using the same eight commonly available standard LEGO® bricks: 1×1, 1×2, 1×4, 1×6, 1×8, 2×2, 2×4, and 2×6. Section 3.1 introduces our tokenization scheme and model architecture. Section 3.2 presents a buildability-aware DPO post-training stage, where preference pairs are constructed by jointly considering geometric fidelity and structural stability. Section 3.3 introduces validity-constrained decoding and stability-guided rollback.

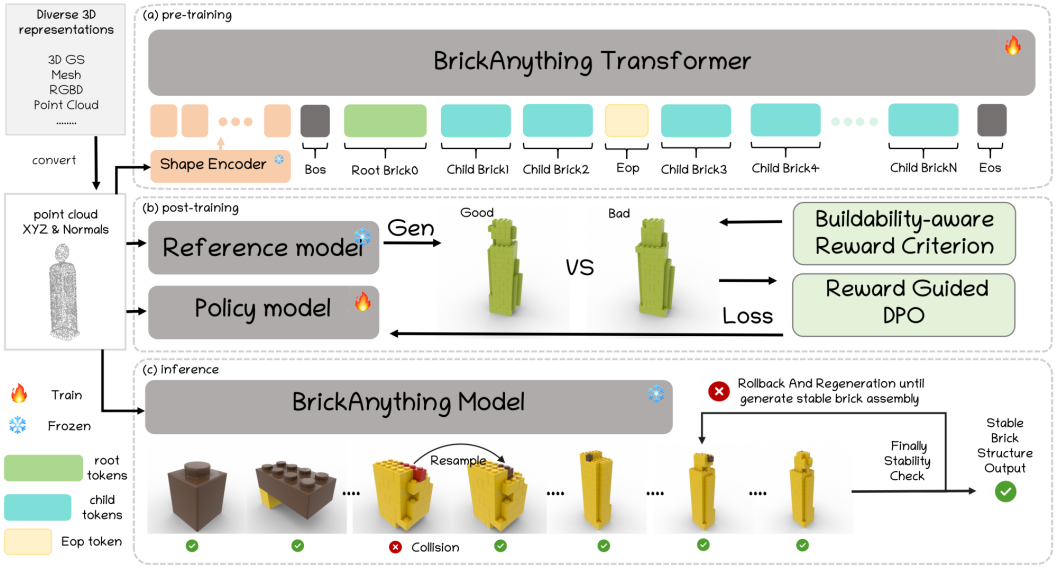

Figure 1: **BrickAnything Framework Overview.** BrickAnything converts diverse 3D inputs into point clouds as a modality-agnostic geometric interface. The framework consists of three stages: 1) **Pre-training**, where a shape encoder and BrickAnything Transformer learn structure-aware autoregressive brick generation; 2) **Post-training**, where buildability-aware rewards are used to construct preference pairs and optimize the policy model with reward-guided DPO; and 3) **Inference**, where validity-constrained decoding and rollback-based regeneration are applied to produce stable brick structures.

3.1 Structure-Aware Tokenization and Geometry-Conditioned Generation

Unlike prior brick generation methods [21, 22] that represent structures as flattened sequences that preserve spatial order, we propose a structure-aware tree tokenization scheme that explicitly encodes local attachment relations between bricks.

Structure-aware tree tokenization. Given a brick structure $\mathcal{B} = \{b_i\}_{i=1}^{N}$, each brick is represented as $b_i = [h_i, w_i, x_i, y_i, z_i]$, where (h_i, w_i) denotes its footprint size along the x- and y-axes, and (x_i, y_i, z_i) is the discrete position of the stud closest to the origin. We first construct a vertical attachment graph $\mathcal{G} = (\mathcal{B}, \mathcal{E})$, where an edge indicates that two bricks are vertically adjacent and overlap in the xy-plane:

$$(b_i, b_j) \in \mathcal{E} \quad \Longleftrightarrow \quad |z_i - z_j| = 1 \text{ and } \Omega_i^{xy} \cap \Omega_j^{xy} \neq \emptyset \tag{1}$$

where Ω_i^{xy} is the footprint of b_i on the xy-plane. For a stable brick structure, $\mathcal{G}$ is connected [25]. We therefore choose the root brick by the lexicographic order of (z, y, x) and perform breadth-first traversal over $\mathcal{G}$. Each unvisited brick is assigned to the first visited neighboring parent, yielding a deterministic BFS spanning tree, as shown in Figure 2.

The root brick is encoded by its absolute attributes:

$$(x_0, y_0, z_0, h_0, w_0) \tag{2}$$

For each non-root brick b_i, we encode it relative to its parent $b_{p(i)}$ as

$$(f_i, h_i, w_i, m_i) \tag{3}$$

where f_i indexes the attachment position on the parent, (h_i, w_i) is the child brick size, and m_i indexes the child-side anchor position, as shown in Figure 2. During BFS traversal, children of the same parent are sorted increasingly by f_i. After all children of the current parent are encoded, an `EOP` token is inserted to mark the end of this local attachment group. The final sequence is

$$\mathbf{s} = [\texttt{BOS}, x_0, y_0, z_0, h_0, w_0, (f, h, w, m), \ldots, \texttt{EOP}, (f, h, w, m), \ldots, \texttt{EOP}, \ldots, \texttt{EOS}] \tag{4}$$

This representation explicitly attaches each newly generated brick to an existing parent, thereby converting brick generation from global coordinate prediction into local attachment prediction. More details of the proposed tokenization scheme are provided in Appendix A.

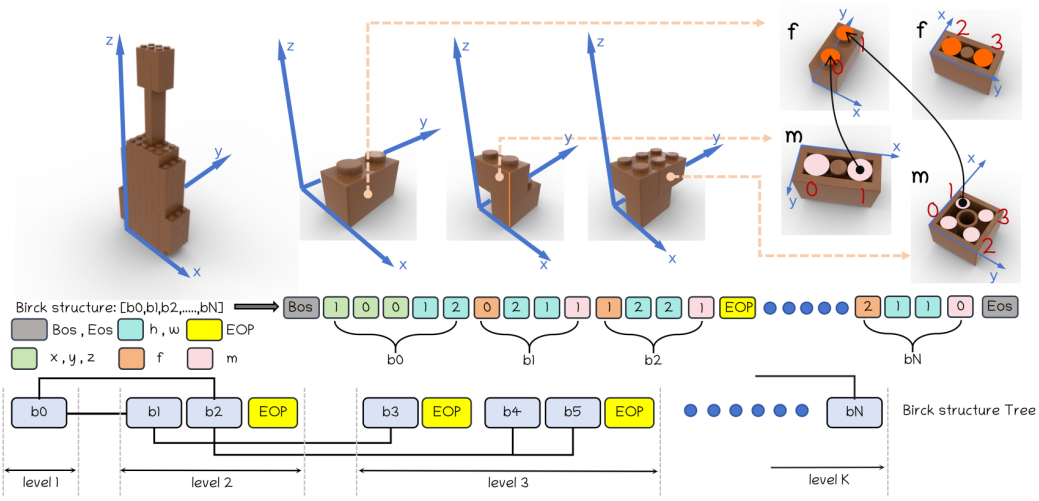

Figure 2: Overview of the proposed structure-aware tree tokenization. The root brick is encoded by absolute attributes (x, y, z, h, w), while each child brick is encoded by relative attachment tokens (f, h, w, m). `EOP` marks the end of each parent's child group in the BFS traversal.

Shape-conditioned brick generation. Given an input 3D shape, we first convert it into a point cloud representation with surface normals, denoted as $\mathcal{P} \in \mathbb{R}^{N\times 3}$ and $\mathcal{N} \in \mathbb{R}^{N\times 3}$, where $N = 8192$. The point cloud and its normals are encoded by a pre-trained Michelangelo[46] encoder $\mathcal{E}_g$, producing a sequence of shape tokens

$$\mathbf{c}_g = \mathcal{E}_g(\mathcal{P}, \mathcal{N}) \tag{5}$$

which serves as the geometric condition for brick structure generation. We choose OPT-350M[47] as our autoregressive transformer architecture. Given the structure-aware tree sequence $\mathbf{s} = (s_1, s_2, \ldots, s_T)$, our transformer decoder with parameters θ generates the sequence conditioned on the shape tokens $\mathbf{c}_g$. The model is trained using the standard next-token prediction objective:

$$\mathcal{L}_{\text{SFT}} = -\sum_{i=1}^{T} \log p_\theta \left(s_i \mid s_{<i}, \mathbf{c}_g\right) \tag{6}$$

During inference, generation starts from the shape condition and the `BOS` token, and the generated sequence is then converted into a brick structure through our tree detokenization algorithm.

3.2 DPO-based Post-Training for Buildable Brick Generation

Next-token prediction optimizes token-level likelihood but cannot directly enforce non-differentiable objectives such as geometric fidelity and physical buildability. To address this, we introduce a reward-guided DPO post-training stage. Specifically, for each input shape, we generate multiple candidate brick structures and rank them according to our proposed buildability-aware reward. The ranked candidates are then used to fine-tune the model with a reward-weighted DPO objective, together with an auxiliary SFT loss.

Buildability-aware reward. Given an input point cloud $\mathcal{P}$ and a generated brick structure $\hat{\mathcal{B}}$, we evaluate generation quality from two aspects: geometric fidelity and physical stability. For geometric fidelity, we define a reward that combines voxel-level occupancy consistency and surface-level geometric alignment.

We first voxelize the input point cloud into an occupancy grid $V_{\mathcal{P}}$, and convert the generated brick structure into another occupancy grid $V_{\hat{\mathcal{B}}}$. The voxel-level consistency is measured by the intersection-over-union:

$$R_{\text{IoU}} = \frac{|V_{\mathcal{P}} \cap V_{\hat{\mathcal{B}}}|}{|V_{\mathcal{P}} \cup V_{\hat{\mathcal{B}}}|} \in [0, 1] \tag{7}$$

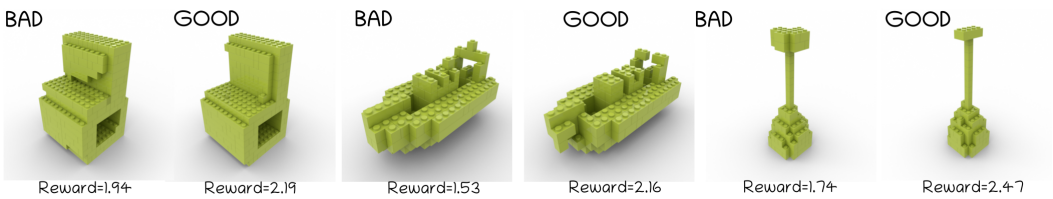

Figure 3: Examples of the collected preference pairs for reward-guided DPO.

Although IoU measures volumetric overlap, it may overlook fine-grained surface discrepancies. Therefore, we introduce a surface-level distance metric to complement the voxel-level score and better capture boundary alignment and local geometric details.

Specifically, we extract a surface mesh $\hat{\mathcal{M}}$ from the generated voxel grid $V_{\hat{\mathcal{B}}}$ using Marching Cubes [48], and uniformly sample a point cloud $\hat{\mathcal{P}}$ from the reconstructed surface. Before computing the distance, both $\hat{\mathcal{P}}$ and $\mathcal{P}$ are centered at their centroids and scaled by their maximum radial distances. We then compute the Chamfer Distance between the normalized point clouds:

$$D_{\mathrm{CD}}(\mathcal{P}, \hat{\mathcal{P}}) = \frac{1}{|\mathcal{P}|} \sum_{\mathbf{x} \in \mathcal{P}} \min_{\hat{\mathbf{x}} \in \hat{\mathcal{P}}} \|\mathbf{x} - \hat{\mathbf{x}}\|_2 + \frac{1}{|\hat{\mathcal{P}}|} \sum_{\hat{\mathbf{x}} \in \hat{\mathcal{P}}} \min_{\mathbf{x} \in \mathcal{P}} \|\hat{\mathbf{x}} - \mathbf{x}\|_2 \tag{8}$$

A smaller Chamfer Distance indicates better surface-level alignment between the generated brick structure and the target geometry. We convert this distance into a bounded reward by

$$R_{\mathrm{CD}} = \max\left(1 - 5D_{\mathrm{CD}}(\mathcal{P}, \hat{\mathcal{P}}), 0\right) \in [0, 1] \tag{9}$$

The final geometric reward is defined as the sum of the voxel-level and surface-level rewards:

$$R_{\mathrm{geo}} = R_{\mathrm{IoU}} + R_{\mathrm{CD}} \in [0, 2] \tag{10}$$

For structural stability, we follow StableLego[49] and estimate a per-brick stability score $s_i \in [0, 1]$ based on static equilibrium under frictional constraints. Since a single unstable brick can compromise the whole assembly, we define the stability reward as the minimum per-brick score:

$$R_{\mathrm{stable}} = \min_{b_i \in \hat{\mathcal{B}}} s_i \in [0, 1] \tag{11}$$

The overall reward is therefore

$$R(\hat{\mathcal{B}}, \mathcal{P}) = R_{\mathrm{geo}} + R_{\mathrm{stable}} \in [0, 3] \tag{12}$$

Preference pair construction. For each input shape, we sample six legal candidate brick structures from the SFT model and compute their rewards. We construct pairwise comparisons and retain only informative pairs whose reward gap is at least 0.2 and whose higher reward is no less than 1. For each retained pair, the higher-reward candidate is used as the preferred sample y_w, and the other as the rejected sample y_l. Figure 3 shows some selection cases of our collected preference pairs.

Reward-guided DPO. Given an input condition x, a preferred sequence y_w, and a rejected sequence y_l, we optimize the policy model π_θ with a reward-weighted DPO objective:

$$\mathcal{L}_{\mathrm{DPO}} = -\mathbb{E}_{(x, y_w, y_l)} \left[\Delta R \log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x) \pi_{\mathrm{ref}}(y_l|x)}{\pi_{\mathrm{ref}}(y_w|x) \pi_\theta(y_l|x)} \right) \right] \tag{13}$$

where π_{ref} is the frozen reference model, β controls the strength of the KL constraint, and

$$\Delta R = R(y_w, x) - R(y_l, x) \tag{14}$$

The reward gap ΔR assigns larger weights to pairs with clearer quality differences, encouraging the model to distinguish high-fidelity and stable assemblies from inferior ones.

To preserve the original data distribution, we further add an auxiliary SFT loss on the ground-truth sequence y_{gt}:

$$\mathcal{L}_{\mathrm{SFT}} = -\sum_{t=1}^{T} \log \pi_\theta(y_{\mathrm{gt},t} \mid y_{\mathrm{gt},<t}, x) \tag{15}$$

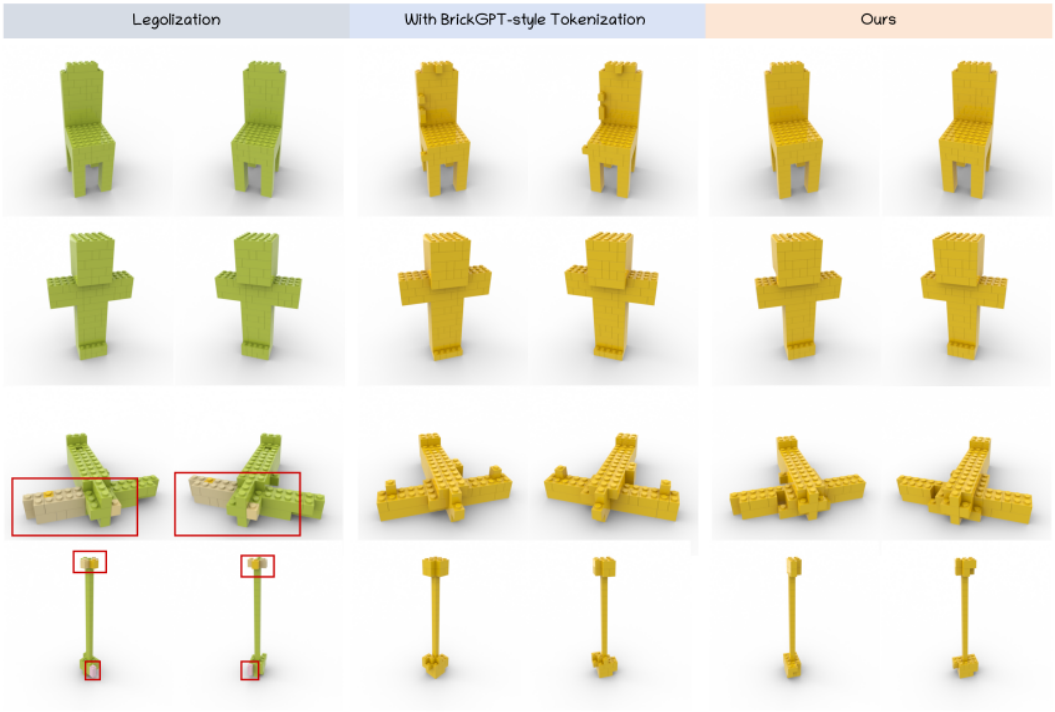

Figure 4: **Qualitative comparison.** BrickAnything produces stable and geometry-faithful structures than the baselines. Red boxes mark Legolization failures.

The final post-training objective is

$$\mathcal{L}_{\text{post}} = \mathcal{L}_{\text{DPO}} + \lambda \mathcal{L}_{\text{SFT}} \tag{16}$$

where λ balances preference alignment and distribution preservation.

3.3 Validity-Constrained Decoding and Stability-Guided Rollback

Validity-constrained decoding. Autoregressive decoding may produce invalid local decisions, including unsupported brick sizes, invalid attachment tokens, and spatial collisions. Instead of accepting each decoded child tuple directly, we perform tuple-level validity checking during inference. For each child brick, the model samples a candidate tuple (f, h, w, m), which is decoded into a concrete brick placement based on the current parent brick and BFS traversal state. The candidate tuple is then verified in two stages. First, we check whether the token combination defines a valid attachment, including a feasible parent connector f, a brick size (h, w) from the predefined brick library, and a valid child-side anchor m. Second, after the candidate is converted into grid occupancy, we check whether it overlaps with any previously generated brick. If either check fails, the candidate tuple is rejected and resampled. The accepted tuple is appended to the sequence only after passing these validity checks.

stability-guided rollback. Although validity-constrained decoding filters invalid local decisions, it does not guarantee the global physical stability of the complete structure. To further improve physical buildability, we perform stability-guided rollback after a complete brick structure is generated. Following StableLego [49], we compute a per-brick stability score $s_i \in [0, 1]$ for each brick in the generated structure, where $s_i = 0$ denotes that brick b_i is unstable. If the generated structure contains unstable bricks, we identify the first unstable brick according to the generation order:

$$k = \min\{i \mid s_i = 0\} \tag{17}$$

We then trace its parent brick $b_{p(k)}$ in the generated tree structure and locate the token position where $b_{p(k)}$ itself is generated in the generated token sequence. The sequence is rolled back to the state before generating $b_{p(k)}$ and the subsequent tokens are regenerated. This process is repeated until a

Table 1: **Quantitative comparison with baselines and ablations.** We compare BrickAnything with representative baselines and ablated variants on the challenging and stable subsets. Lower CD and rollback averages are better, while higher IoU, %Stable, and %Valid indicate better performance. , , and indicate the best, second-best, and third-best results, respectively.

(a) Challenging subset

Method	CD ↓	IoU ↑	Rollback ↓	%Stable ↑	%Valid ↑
Legolization[25]	–	–	–	0.0%	100%
BrickGPT[21]-style tokenization w/o DPO	0.1307	0.544	6.991	74.0%	100%
BrickGPT[21]-style tokenization	0.1309	0.553	6.750	76.0%	100%
Ours w/o Validity-constrained decoding or Rollback or DPO	0.1271	0.602	–	60.6%	68.6%
Ours w/o Rollback or DPO	0.1283	0.597	–	72.0%	100%
Ours w/o DPO	0.1301	0.573	0.444	82.4%	100%
Ours	0.1299	0.586	0.422	83.4%	100%

(b) Stable subset

Method	CD ↓	IoU ↑	Rollback ↓	%Stable ↑	%Valid ↑
Legolization[25]	0.1120	1.000	–	100%	100%
BrickGPT[21]-style tokenization w/o DPO	0.1297	0.721	3.916	100%	100%
BrickGPT[21]-style tokenization	0.1292	0.742	3.620	100%	100%
Ours w/o Validity-constrained decoding or Rollback or DPO	0.1265	0.798	–	63.4%	71.6%
Ours w/o Rollback or DPO	0.1274	0.772	–	91.0%	100%
Ours w/o DPO	0.1290	0.751	0.216	100%	100%
Ours	0.1283	0.788	0.184	100%	100%

stable structure is obtained or the maximum rollback budget is reached. The parent-child attachment relations allow our strategy to locate the source of instability more precisely, regenerate the affected substructure, and preserve stable preceding bricks, leading to more targeted and structurally consistent correction.

4 Experiments

4.1 Implementation Details

Dataset. We construct our training data from a curated collection of approximately 230K high-quality meshes from ShapeNet[50], Objaverse[51], and Objaverse-XL[52], following recent mesh curation practice [53]. Applying Legolization [25] yields around 168K stable mesh–brick pairs, of which 4K are randomly selected for validation and the rest are used for training. For evaluation, we build two test subsets: a stable subset with 500 validation samples successfully converted by Legolization, and a challenging subset with 500 meshes sampled from cases where Legolization fails to produce stable structures. More training details are provided in Appendix C.

4.2 Metrics and Baselines

Metrics We evaluate generated brick structures in terms of geometric fidelity, structural validity, physical stability, and inference correction cost. Geometric fidelity is measured by Voxel IoU and Chamfer Distance, as defined in Sec. 3.2; structural validity is the percentage of outputs satisfying valid brick types, workspace bounds, and collision-free placement; physical stability is measured by the stable generation rate following StableLego [49]; and inference cost is measured by the average number of rollback operations. Unless otherwise specified, Voxel IoU, Chamfer Distance, and rollback average are computed over final generated samples that are physically stable.

Baselines. We compare our method with two representative baselines. First, we implement a baseline using the same point-cloud encoder and transformer architecture as ours, but replacing the proposed structure-aware tree tokenization with a conventional BrickGPT [21]-style tokenization. Specifically, each brick is represented as (h, w, x, y, z), and bricks are sorted into a flat sequence by the lexicographic order of (z, y, x). The baseline is pretrained on the same dataset as our model and

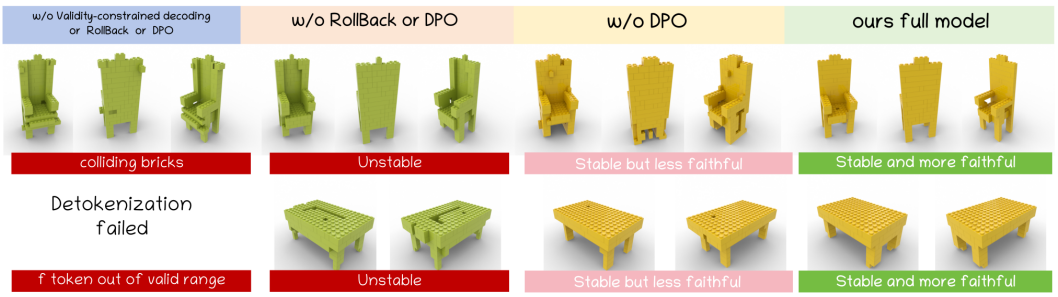

Figure 5: **Qualitative ablation results.** Each component improves buildability or fidelity: validity-constrained decoding removes invalid placements, rollback corrects instability, and DPO enhances shape faithfulness.

equipped with the same brick-by-brick rejection sampling and physics-aware rollback strategies used in BrickGPT [21], isolating the effect of the proposed tokenization scheme. Second, we compare with Legolization [25], a heuristic search-based method for converting 3D shapes into stable brick structures. This comparison evaluates whether our learned model can produce geometrically faithful and physically buildable structures, especially in cases where heuristic search fails.

4.3 Comparison

Table 1 compares BrickAnything with representative baselines on the challenging and stable subsets. On the challenging subset, Legolization [25] obtains 0.0% stable rates, indicating that heuristic search is brittle for geometrically difficult targets. The BrickGPT-style baseline [21] benefits from DPO, improving the stable rate from 74.0% to 76.0%, IoU from 0.544 to 0.553, and reducing the average number of rollbacks from 6.991 to 6.750. In contrast, BrickAnything achieves stronger buildability and fidelity, reaching 83.4% stable rate, 100% valid rate, 0.586 IoU, and a much lower rollback average of 0.422.

On the stable subset, Legolization obtains the best CD and IoU because this subset is selected from its successful cases. Among learned methods, BrickAnything achieves the best overall performance. Compared with the BrickGPT-style baseline, BrickAnything improves IoU from 0.742 to 0.788, reduces CD from 0.1292 to 0.1283, and lowers the rollback average from 3.620 to 0.184, while maintaining 100% stable and valid rates. Together with Figure 4, these results show that structure-aware tree tokenization improves geometric fidelity and sequence coherence, while DPO, validity-constrained decoding, and rollback enhance buildability with fewer correction steps. Additional qualitative results, failure analyses, and image- and text-to-brick extensions are provided in Appendix B.

4.4 Ablation Studies

Table 1 and Figure 5 ablate the main components of BrickAnything. Removing validity-constrained decoding, rollback, and DPO leads to strong geometric scores but substantially weaker buildability, with the stable and valid rates dropping to 60.6% and 68.6% on the challenging subset. Adding validity-constrained decoding restores the valid rate to 100%, but the stable rate remains limited without rollback, indicating that local validity constraints alone cannot guarantee global physical stability. Rollback further improves stability by correcting unstable generations through structurally related resampling. Finally, DPO improves the generation distribution itself: compared with the variant without DPO, the full model improves IoU from 0.573 to 0.586 on the challenging subset and from 0.751 to 0.788 on the stable subset, while reducing the average number of rollbacks from 0.444 to 0.422 and from 0.216 to 0.184, respectively. These results show that validity-constrained decoding, rollback, and DPO address complementary aspects of buildable brick generation. We provide a detailed ablation analysis in Appendix D.

5 Conclusion

We presented **BrickAnything**, a geometry-conditioned framework for buildable brick generation. By combining point-cloud conditioning with structure-aware tree tokenization, BrickAnything models local attachment relations in autoregressive generation. Buildability-aware DPO, validity-constrained decoding, and stability-guided rollback further improve fidelity and stability. Experiments demonstrate superior performance over coordinate-ordered tokenization baselines and robustness when heuristic Legolization fails. We discuss the limitations of our work and potential directions for future research in Appendix E.

References

[1] Zhen Zhou, Jian Liu, Biwen Lei, Jing Xu, Haohan Weng, Yiling Zhu, Zhuo Chen, Junfeng Fan, Yunkai Ma, Dazhao Du, et al. Mesh-pro: Asynchronous advantage-guided ranking preference optimization for artist-style quadrilateral mesh generation. *arXiv preprint arXiv:2603.00526*, 2026.

[2] Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Yikang Yang, Yajie Bao, Jiachen Qian, Siyu Zhu, Xun Cao, Philip Torr, et al. Direct3d-s2: Gigascale 3d generation made easy with spatial sparse attention. *arXiv preprint arXiv:2505.17412*, 2025.

[3] Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Xin Chen, Zhongang Cai, Lei Yang, Gang Yu, et al. Meshanything: Artist-created mesh generation with autoregressive transformers. *arXiv preprint arXiv:2406.10163*, 2024.

[4] Ruowen Zhao, Junliang Ye, Zhengyi Wang, Guangce Liu, Yiwen Chen, Yikai Wang, and Jun Zhu. Deepmesh: Auto-regressive artist-mesh creation with reinforcement learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10612–10623, 2025.

[5] Jiaxiang Tang, Zhaoshuo Li, Zekun Hao, Xian Liu, Gang Zeng, Ming-Yu Liu, and Qinsheng Zhang. Edgerunner: Auto-regressive auto-encoder for artistic mesh generation. *arXiv preprint arXiv:2409.18114*, 2024.

[6] Jian Liu, Jing Xu, Song Guo, Jing Li, Jingfeng Guo, Jiaao Yu, Haohan Weng, Biwen Lei, Xianghui Yang, Zhuo Chen, et al. Mesh-rft: Enhancing mesh generation via fine-grained reinforcement fine-tuning. *arXiv preprint arXiv:2505.16761*, 2025.

[7] Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, Karsten Kreis, et al. Lion: Latent point diffusion models for 3d shape generation. *Advances in neural information processing systems*, 35: 10021–10039, 2022.

[8] Zixuan Huang, Justin Johnson, Shoubhik Debnath, James M Rehg, and Chao-Yuan Wu. Pointinfinity: Resolution-invariant point diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10050–10060, 2024.

[9] Ka-Hei Hui, Chao Liu, Xiaohui Zeng, Chi-Wing Fu, and Arash Vahdat. Not-so-optimal transport flows for 3d point cloud generation. *arXiv preprint arXiv:2502.12456*, 2025.

[10] Zhiyuan Ren, Minchul Kim, Feng Liu, and Xiaoming Liu. Tiger: Time-varying denoising model for 3d point cloud generation with diffusion process. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9462–9471, 2024.

[11] Etai Sella, Noam Atia, Ron Mokady, and Hadar Averbuch-Elor. Blended point cloud diffusion for localized text-guided shape editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19119–19129, 2025.

[12] Chenliang Zhou, Fangcheng Zhong, Param Hanji, Zhilin Guo, Kyle Fogarty, Alejandro Sztrajman, Hongyun Gao, and Cengiz Oztireli. Frepolad: Frequency-rectified point latent diffusion for point cloud generation. In *European Conference on Computer Vision*, pages 434–453. Springer, 2024.

[13] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4):139–1, 2023.

[14] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37:140138–140158, 2024.

[15] David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19457–19467, June 2024.

[16] Lihan Jiang, Yucheng Mao, Linning Xu, Tao Lu, Kerui Ren, Yichen Jin, Xudong Xu, Mulin Yu, Jiangmiao Pang, Feng Zhao, Dahua Lin, and Bo Dai. Anysplat: Feed-forward 3d gaussian splatting from unconstrained views. *ACM Transactions on Graphics (TOG)*, 44(6):1–16, December 2025. doi: 10.1145/3763326.

[17] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65 (1):99–106, 2021.

[18] Paritosh Mittal, Yen-Chi Cheng, Maneesh Singh, and Shubham Tulsiani. Autosdf: Shape priors for 3d completion, reconstruction and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 306–315, June 2022.

[19] Xingguang Yan, Liqiang Lin, Niloy J. Mitra, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Shapeformer: Transformer-based shape completion via sparse representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6239–6249, June 2022.

[20] Yen-Chi Cheng, Hsin-Ying Lee, Sergey Tulyakov, Alexander G. Schwing, and Liang-Yan Gui. Sdfusion: Multimodal 3d shape completion, reconstruction, and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4456–4465, June 2023.

[21] Ava Pun, Kangle Deng, Ruixuan Liu, Deva Ramanan, Changliu Liu, and Jun-Yan Zhu. Generating physically stable and buildable brick structures from text. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14798–14809, 2025.

[22] Jiahao Ge, Mingjun Zhou, Hanyou Zheng, Hao Xu, and Chi-Wing Fu. Lego®-maker: Autoregressive image-conditioned lego® model creation. *ACM Transactions on Graphics (TOG)*, 44(6):1–15, 2025.

[23] Peter Kulits and Cordelia Schmid. Bricknet: Graph-backed generative brick assembly. *arXiv preprint arXiv:2604.22984*, 2026.

[24] Kyle Lennon, Katharina Fransen, Alexander O'Brien, Yumeng Cao, Matthew Beveridge, Yamin Arefeen, Nikhil Singh, and Iddo Drori. Image2lego: Customized lego set generation from images. *arXiv preprint arXiv:2108.08477*, 2021.

[25] Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. Legolization: Optimizing lego designs. *ACM Transactions on Graphics (ToG)*, 34(6):1–12, 2015.

[26] Romain Pierre Testuz, Yuliy Schwartzburg, and Mark Pauly. Automatic generation of constructable brick sculptures. In *EUROGRAPHICS 2013*, 2013.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[29] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[30] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR, 2018.

[31] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020.

[32] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *Advances in neural information processing systems*, 37:84839–84865, 2024.

[33] An-Chieh Cheng, Xueting Li, Sifei Liu, Min Sun, and Ming-Hsuan Yang. Autoregressive 3d shape generation via canonical mapping. In *European Conference on Computer Vision*, pages 89–104. Springer, 2022.

[34] Ziqiao Meng, Qichao Wang, Zhipeng Zhou, Irwin King, and Peilin Zhao. 3d point cloud generation via autoregressive up-sampling. *arXiv e-prints*, pages arXiv–2503, 2025.

[35] Nicolas von Lützow, Barbara Rössle, Katharina Schmid, and Matthias Nießner. Gaussiangpt: Towards autoregressive 3d gaussian scene generation. *arXiv preprint arXiv:2603.26661*, 2026.

[36] Hongyu Liu, Xuan Wang, Yating Wang, Zijian Wu, Ziyu Wan, Yue Ma, Runtao Liu, Boyao Zhou, Yujun Shen, and Qifeng Chen. Avatarpointillist: Autoregressive 4d gaussian avatarization. *arXiv preprint arXiv:2604.04787*, 2026.

[37] Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International conference on machine learning*, pages 7220–7229. PMLR, 2020.

[38] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19615–19625, 2024.

[39] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[40] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[41] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[44] Zhenglin Zhou, Xiaobo Xia, Fan Ma, Hehe Fan, Yi Yang, and Tat-Seng Chua. Dreamdpo: Aligning text-to-3d generation with human preferences via direct preference optimization. *arXiv preprint arXiv:2502.04370*, 2025.

[45] Jingfeng Guo, Jian Liu, Jinnan Chen, Shiwei Mao, Changrong Hu, Puhua Jiang, Junlin Yu, Jing Xu, Qi Liu, Lixin Xu, et al. Auto-connect: Connectivity-preserving rigformer with direct preference optimization. *arXiv preprint arXiv:2506.11430*, 2025.

[46] Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, Bin Fu, Tao Chen, Gang Yu, and Shenghua Gao. Michelangelo: Conditional 3d shape generation based on shape-image-text aligned latent representation. *Advances in neural information processing systems*, 36:73969–73982, 2023.

[47] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[48] LORENSEN WE. Marching cubes: A high resolution 3d surface construction algorithm. *Computer graphics*, 21(1):7–12, 1987.

[49] Ruixuan Liu, Kangle Deng, Ziwei Wang, and Changliu Liu. Stablelego: Stability analysis of block stacking assembly. *IEEE Robotics and Automation Letters*, 9(11):9383–9390, 2024.

[50] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[51] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.

[52] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in neural information processing systems*, 36:35799–35813, 2023.

[53] Yiwen Chen, Yikai Wang, Yihao Luo, Zhengyi Wang, Zilong Chen, Jun Zhu, Chi Zhang, and Guosheng Lin. Meshanything v2: Artist-created mesh generation with adjacent mesh tokenization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13922–13931, 2025.

[54] Zeqiang Lai, Yunfei Zhao, Haolin Liu, Zibo Zhao, Qingxiang Lin, Huiwen Shi, Xianghui Yang, Mingxin Yang, Shuhui Yang, Yifei Feng, et al. Hunyuan3d 2.5: Towards high-fidelity 3d assets generation with ultimate details. *arXiv preprint arXiv:2506.16504*, 2025.

Appendix

A More Details of Structure-Aware Tree Tokenization

A.1 Attachment Token Definition and Detokenization

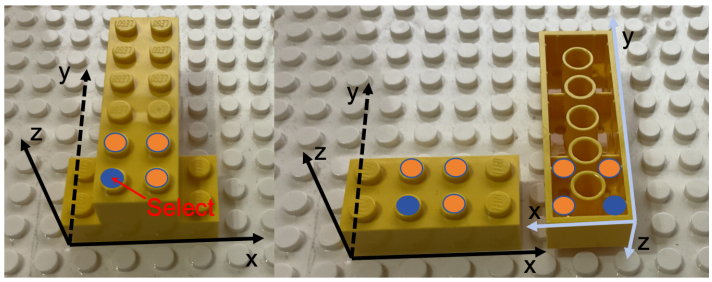

Figure 6: **Example of attachment token definition.** The selected shared stud is expressed in the local coordinate systems of both the parent and child bricks. The parent-side local coordinate determines f_i, while the child-side local coordinate determines m_i.

For each non-root brick b_i, we encode its attachment to the parent brick $b_{p(i)}$ using a deterministic reference stud within their overlapping footprint. Let $\Omega^{xy}_{p(i)}$ and Ω^{xy}_i denote the sets of occupied stud locations of the parent and child bricks in the global xy-plane. Since a valid vertical attachment requires $\Omega^{xy}_{p(i)} \cap \Omega^{xy}_i \neq \emptyset$, we select a canonical attachment location $(q^x_i, q^y_i) = \min\left(\Omega^{xy}_{p(i)} \cap \Omega^{xy}_i\right)$ where $\min$ denotes the lexicographically smallest stud location. This deterministic choice removes ambiguity when multiple studs overlap. We then express this global attachment location in the local coordinate systems of the parent and child bricks:

$$(u^p_i, v^p_i) = (q^x_i - x_{p(i)},\ q^y_i - y_{p(i)}) \tag{18}$$

$$(u^c_i, v^c_i) = (q^x_i - x_i,\ q^y_i - y_i) \tag{19}$$

The parent-side attachment token f_i encodes the vertical direction and the local attachment position on the parent brick, while the child-side anchor token m_i encodes the corresponding local anchor position on the child brick:

$$f_i = s_i h_{p(i)} w_{p(i)} + v^p_i h_{p(i)} + u^p_i \tag{20}$$

$$m_i = v^c_i h_i + u^c_i \tag{21}$$

where $s_i \in \{0, 1\}$ indicates the vertical attachment direction, with $s_i = 0$ meaning that the child is attached above the parent and $s_i = 1$ meaning that it is attached below the parent.

Figure 6 shows a simple example. The selected shared stud has local coordinate $(u^p_i, v^p_i) = (1, 0)$ on the parent brick and $(u^c_i, v^c_i) = (0, 0)$ on the child brick. Since the child is attached above the parent, we have $s_i = 0$. For a parent brick with $(h_{p(i)}, w_{p(i)}) = (4, 2)$ and a child brick with $h_i = 2$, the corresponding tokens are $f_i = 0 \times 2 \times 4 + 0 \times 4 + 1 = 1$ and $m_i = 0 \times 2 + 0 = 0$.

During detokenization, given the parent state and the child size (h_i, w_i), we first recover the local attachment variables from f_i and m_i: $s_i = \left\lfloor f_i / (h_{p(i)} w_{p(i)}) \right\rfloor$, $r_i = f_i \bmod (h_{p(i)} w_{p(i)})$, $u^p_i = r_i \bmod h_{p(i)}$, $v^p_i = \left\lfloor r_i / h_{p(i)} \right\rfloor$, $u^c_i = m_i \bmod h_i$, and $v^c_i = \lfloor m_i / h_i \rfloor$. The child position is then recovered as $x_i = x_{p(i)} + u^p_i - u^c_i$, $y_i = y_{p(i)} + v^p_i - v^c_i$, and $z_i = z_{p(i)} + (1 - 2s_i)$. Thus, the attachment tokens provide a reversible local representation of each child brick relative to its parent.

A.2 Structure-aware tree tokenization and detokenization algorithm

Algorithm 1: Structure-aware Tree Tokenization

Input: A brick assembly $\mathcal{B} = \{b_i\}_{i=1}^{N}$, where $b_i = (x_i, y_i, z_i, h_i, w_i)$
Output: A token sequence $\mathbf{s}$

Construct a vertical attachment graph $\mathcal{G} = (\mathcal{B}, \mathcal{E})$;
foreach *pair of bricks* (b_i, b_j) **do**
  if $|z_i - z_j| = 1$ *and* $\Omega_i^{xy} \cap \Omega_j^{xy} \neq \emptyset$ **then**
    Add edge (b_i, b_j) to $\mathcal{E}$;
Select root brick $b_{root} = \arg\min_{b_i}(z_i, y_i, x_i)$;
Initialize queue $Q \leftarrow [b_{root}]$;
Initialize visited set $\mathcal{V} \leftarrow \{b_{root}\}$;
Initialize token sequence $\mathbf{s} \leftarrow [\texttt{BOS}, x_0, y_0, z_0, h_0, w_0]$;
while Q *is not empty* **do**
  Pop current parent brick b_p from Q;
  Find unvisited neighbors $\mathcal{C}(b_p)$ of b_p in $\mathcal{G}$;
  Sort $\mathcal{C}(b_p)$ by the parent-side connector index f;
  foreach *child brick* $b_c \in \mathcal{C}(b_p)$ **do**
    Compute parent-side connector index f;
    Compute child-side anchor index m;
    Append (f, h_c, w_c, m) to $\mathbf{s}$;
    Add b_c to $\mathcal{V}$;
    Push b_c into Q;
  Append `EOP` to $\mathbf{s}$;
Remove redundant trailing `EOP` tokens;
Append `EOS` to $\mathbf{s}$;
return $\mathbf{s}$;

Algorithm 2: Structure-aware Tree Detokenization

Input: A token sequence $\mathbf{s}$
Output: A reconstructed brick assembly $\hat{\mathcal{B}}$

Construct root brick $\hat{b}_{root} = (x_0, y_0, z_0, h_0, w_0)$;
Initialize $\hat{\mathcal{B}} \leftarrow \{\hat{b}_{root}\}$ and queue $Q \leftarrow [\hat{b}_{root}]$;
while $\mathbf{s}$ *still has remaining tokens* **do**
  if Q *is empty* **then**
    break;
  Pop current parent brick $\hat{b}_p$ from Q;
  Initialize child list $\mathcal{C} \leftarrow \emptyset$;
  while *next token is not* `EOP` *and tokens remain* **do**
    Read child tuple (f, h_c, w_c, m);
    Decode parent-side connector (s_f, u_p, v_p) from f;
    Decode child-side anchor (u_c, v_c) from m;
    Recover child horizontal position: $x_c \leftarrow x_p + u_p - u_c$, $y_c \leftarrow y_p + v_p - v_c$;
    if $s_f = 0$ **then**
      Set $z_c \leftarrow z_p + 1$;
    else
      Set $z_c \leftarrow z_p - 1$;
    Construct child brick $\hat{b}_c = (x_c, y_c, z_c, h_c, w_c)$;
    Add $\hat{b}_c$ to $\hat{\mathcal{B}}$ and $\mathcal{C}$;
  if *next token is* `EOP` **then**
    Consume `EOP`;
  Push all bricks in $\mathcal{C}$ into Q;
return $\hat{\mathcal{B}}$;

Table 2: **Codebook size and sequence length comparison.** The reported average sequence length is computed on the training set.

Tokenization	Codebook size	Avg. sequence length ↓
BrickGPT-style tokenization	28	1050
Structure-aware tree tokenization	65	**1044**

A.3 Structure-Aware Tree Tokenization Codebook Size and Length Analysis

Codebook size. We compare the codebook size of the conventional BrickGPT[21]-style tokenization and our structure-aware tree tokenization. For the BrickGPT-style baseline, each brick is represented by five tokens, e.g. (h, w, x, y, z). The coordinate tokens share a discrete resolution of 20, corresponding to coordinate values in the $20 \times 20 \times 20$ workspace. The brick-size tokens share five possible values, $\{1, 2, 4, 6, 8\}$, and three additional special tokens are used for `BOS`, `EOS`, and `PAD`. Thus, the BrickGPT-style codebook size is

$$|\mathcal{V}_{\text{BrickGPT}}| = 20 + 5 + 3 = 28 \tag{22}$$

Our structure-aware tree tokenization uses the same coordinate tokens, brick-size tokens, and special tokens. In addition, it introduces one `EOP` token, parent-side attachment tokens f, and child-side anchor tokens m. The f-token set is determined by the maximum parent footprint area and two vertical attachment directions, while the m-token set is determined by the maximum child footprint area. Given our brick library

$$\{1 \times 1, 1 \times 2, 1 \times 4, 1 \times 6, 1 \times 8, 2 \times 2, 2 \times 4, 2 \times 6\} \tag{23}$$

the maximum footprint area is

$$A_{\max} = 2 \times 6 = 12 \tag{24}$$

Therefore, the f-token set has size $2A_{\max} = 24$, and the m-token set has size $A_{\max} = 12$. The total codebook size of our method is

$$|\mathcal{V}_{\text{ours}}| = 20 + 5 + 3 + 1 + 24 + 12 = 65 \tag{25}$$

Sequence length. Let N be the number of bricks in a structure. In the BrickGPT-style tokenization, each brick is represented by five tokens. Ignoring padding, the sequence length is

$$T_{\text{BrickGPT}} = 5N + 2 \tag{26}$$

where the two extra tokens correspond to `BOS` and `EOS`.

In our structure-aware tree tokenization, the root brick is encoded by five absolute tokens $(x_0, y_0, z_0, h_0, w_0)$, while each non-root brick is encoded by four relative tokens (f, h, w, m). In principle, an `EOP` token can be inserted after each parent's child group. However, in our experiments, we omit redundant `EOP` tokens for leaf nodes with no children, since there is no child group to terminate. Let I denote the number of non-leaf parent nodes that emit `EOP`. The sequence length of our tokenization is then

$$T_{\text{ours}} = 1 + 5 + 4(N - 1) + I + 1 = 4N + I + 3 \tag{27}$$

where the first and last terms correspond to `BOS` and `EOS`. Since $I \leq N - 1$, we have

$$T_{\text{ours}} \leq 5N + 2 \tag{28}$$

which shows that our representation is no longer than the coordinate-ordered representation, while explicitly encoding parent-child attachment relations.

As shown in Table 2, our tokenization uses a larger codebook due to the additional attachment tokens, but achieves a slightly shorter average sequence length.

B Additional Results and Analysis

B.1 Additional Qualitative Results

Figure 7 shows additional generation results on diverse input shapes. BrickAnything generally preserves the global shape structure while producing physically buildable brick assemblies.



Figure 7: **Additional qualitative results.** We provide more examples of BrickAnything on diverse input shapes. The generated brick structures preserve the overall geometry of the input point clouds while satisfying brick-level construction constraints.

B.2 Failure Cases

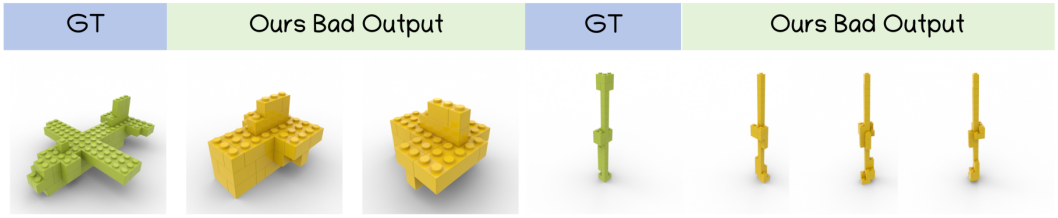

Figure 8: **Failure cases.** We show representative failure cases of BrickAnything.

Figure 8 shows representative failure cases. Failures mainly occur for inputs whose geometry depends on slender components, smooth curved surfaces, or subtle local details, where the limited workspace resolution and restricted brick library make accurate reconstruction difficult.

B.3 Image- and Text-to-Brick Results

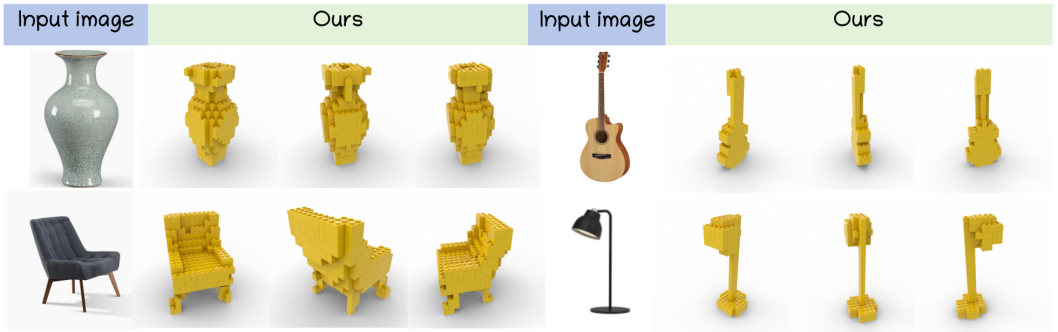

Figure 9: **Image-to-Brick results.** Input images are converted into 3D geometry, which is used to condition BrickAnything. The generated assemblies preserve the major geometric characteristics of the input objects.

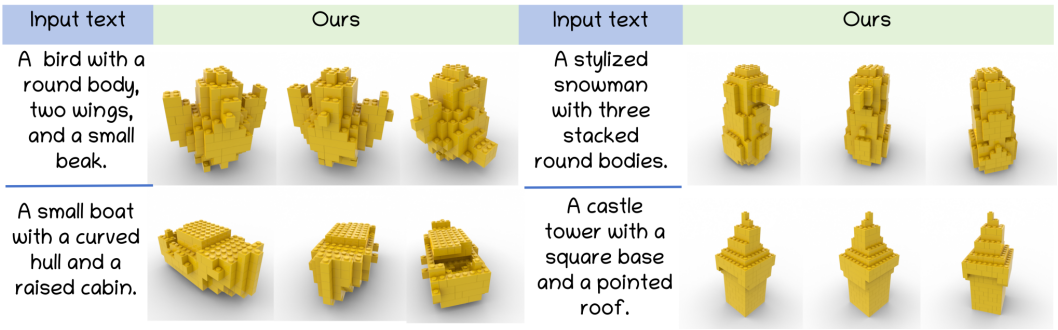

Figure 10: **Text-to-Brick results.** Text prompts are converted into 3D geometry to guide brick assembly generation. The results illustrate the extension of BrickAnything to text-conditioned generation.

Although BrickAnything takes point clouds as input, it can be extended to text- and image-conditioned brick generation through an external 3D generation model. Specifically, we use Hunyuan3D 2.5 [54] to convert text prompts or input images into 3D meshes, which are subsequently sampled into point clouds with normals and processed by BrickAnything. Qualitative results are shown in Fig. 9 and Fig. 10.

Table 3: **Quantitative evaluation of Text-to-Brick and Image-to-Brick generation.** CLIP_s and CLIP_a denote CLIP similarity evaluated on stable assemblies and all assemblies, respectively. Stable denotes the percentage of physically stable assemblies.

Method	$\text{CLIP}_s \uparrow$	$\text{CLIP}_a \uparrow$	Stable (%) $\uparrow$
Text-to-Brick			
BrickAnything	**0.338**	**0.342**	97
BrickGPT	0.274	0.274	**100**
Image-to-Brick			
BrickAnything	0.7438	0.7396	98

Evaluation Protocol. To quantitatively evaluate these extensions, we render each generated brick assembly from eight uniformly distributed viewpoints with fixed rendering settings. We use OpenCLIP ViT-L/14 with OpenAI pretrained weights to measure semantic and visual consistency.

For Text-to-Brick, we evaluate 100 descriptions, including 50 from the BrickGPT test set and 50 generated by an LLM, covering diverse object categories. Following BrickGPT, each description is formatted as "A LEGO model of {input text}." We average the text-image CLIP similarity over eight views and evaluate BrickGPT under the same protocol.

For Image-to-Brick, we randomly select 100 ShapeNet objects and use one rendered view of each object as input. Image-image CLIP similarity is averaged over the eight rendered views of the generated brick assembly.

Results. As shown in Table 3, BrickAnything achieves higher CLIP similarity than BrickGPT on Text-to-Brick generation, indicating improved text-to-structure semantic consistency. Meanwhile, BrickAnything maintains high physical stability, with stable assembly rates of 97% and 98% for Text-to-Brick and Image-to-Brick, respectively. These results demonstrate the applicability of BrickAnything to higher-level input modalities through explicit 3D geometric guidance.

C Training Details

We pre-train BrickAnything on 8 NVIDIA A100 80GB GPUs with a batch size of 16 per GPU, giving a global batch size of 128. The model is trained for 50 epochs, taking approximately one day. For each mesh, we randomly sample 8192 surface points with normal vectors as the input shape representation. The autoregressive decoder is trained with fp16 mixed precision using AdamW. We use a base learning rate of 1×10^{-4}, a final learning rate of 6×10^{-5}, weight decay 0.1, cosine scheduling, first-epoch warmup from 1×10^{-6}, and gradient clipping with maximum norm 1.0. We also apply random scaling, shifting, and rotation augmentations during training.

For DPO post-training, we sample 4K training shapes and construct 16K curated preference pairs using the proposed buildability-aware reward. The pre-sampled point clouds corresponding to each mesh are directly loaded during DPO training, without additional mesh resampling. We fine-tune the model for 10 epochs on 8 NVIDIA A100 80GB GPUs with a per-GPU batch size of 8. The DPO coefficient is set to $\beta = 1$, and the auxiliary SFT loss weight is set to $\lambda = 1$. We use a base learning rate of 5×10^{-7}, a final learning rate of 1×10^{-7}, a warmup learning rate of 1×10^{-7}, gradient clipping of 10.0, and reserve 10% of the preference pairs for validation.

D Detailed Ablation Analysis

We provide a detailed analysis of the ablated variants in Table 1 and Figure 5.

Effect of validity-constrained decoding. Validity-constrained decoding is critical for ensuring that the generated token sequence can be converted into a legal brick structure. When validity-constrained decoding, rollback, and DPO are all removed, the model achieves strong geometric scores, reaching 0.1271 CD and 0.602 IoU on the challenging subset. However, its buildability is substantially weakened, with the stable and valid rates dropping to 60.6% and 68.6%, respectively. A similar trend is observed on the stable subset, where this variant obtains 0.1265 CD and 0.798 IoU, but only

reaches 63.4% stable rate and 71.6% valid rate. This indicates that high geometric overlap alone does not guarantee legal or physically realizable brick assemblies. As shown in Figure 5, unconstrained generation may produce invalid token combinations or spatial collisions, which can further lead to detokenization failures.

Effect of rollback. Adding validity-constrained decoding restores the valid rate to 100% on both subsets, confirming its effectiveness in preventing illegal placements. However, without rollback, the stable rate remains limited: 72.0% on the challenging subset and 91.0% on the stable subset. This shows that local validity constraints can ensure legal brick placements, but they cannot fully guarantee global physical stability. Rollback complements validity-constrained decoding by identifying unstable generated structures and resampling structurally related brick placements. With rollback, the variant without DPO improves the stable rate from 72.0% to 82.4% on the challenging subset, and from 91.0% to 100% on the stable subset. Thus, validity-constrained decoding mainly addresses structural legality, while rollback is necessary for correcting stability failures.

Effect of buildability-aware DPO. DPO improves both the BrickGPT-style baseline and our structure-aware tree-tokenized model. For the BrickGPT-style baseline, DPO improves the stable rate from 74.0% to 76.0% on the challenging subset, increases IoU from 0.544 to 0.553, and reduces the average number of rollbacks from 6.991 to 6.750. On the stable subset, where both variants already achieve 100% stable and valid rates, DPO still improves IoU from 0.721 to 0.742 and reduces rollback average from 3.916 to 3.620.

DPO also benefits our tree-tokenized model. Compared with the variant without DPO, the full model improves IoU from 0.573 to 0.586 on the challenging subset and from 0.751 to 0.788 on the stable subset. It also reduces the average number of rollbacks from 0.444 to 0.422 and from 0.216 to 0.184, respectively. These results suggest that buildability-aware preference optimization improves the generation distribution itself, leading to more geometry-faithful structures and reducing the need for rollback-based correction.

Effect of structure-aware tree tokenization. Compared with the BrickGPT-style coordinate-ordered tokenization, our structure-aware tree tokenization consistently improves fidelity, stability, and correction efficiency. Without DPO, our model improves the challenging-subset IoU from 0.544 to 0.573, increases the stable rate from 74.0% to 82.4%, and reduces rollback average from 6.991 to 0.444. With DPO, BrickAnything further improves over the BrickGPT-style baseline, increasing IoU from 0.553 to 0.586, improving the stable rate from 76.0% to 83.4%, and reducing rollback average from 6.750 to 0.422. On the stable subset, BrickAnything also improves IoU from 0.742 to 0.788 and reduces rollback average from 3.620 to 0.184, while maintaining 100% stable and valid rates. These results indicate that explicitly modeling local attachment relations yields more coherent brick sequences and substantially reduces the correction burden during inference.

Summary. Overall, the ablation results show that the three components address complementary aspects of buildable brick generation. Validity-constrained decoding ensures local legality, rollback corrects global stability failures, and buildability-aware DPO improves the model distribution toward more faithful and stable generations. In addition, structure-aware tree tokenization provides a stronger sequence representation than coordinate-ordered tokenization, leading to higher fidelity, higher stability, and fewer rollback corrections.

D.1 Physical Assembly Demonstration

To further demonstrate the practical buildability of BrickAnything, we physically construct several generated assemblies using real bricks, as shown in Fig. 11. These examples cover diverse object geometries, including thin structures and complex shapes. The physical constructions preserve the main geometric characteristics of the input point clouds, demonstrating the feasibility of translating our generated assemblies into tangible brick structures.

E Limitations and Future Work

BrickAnything is currently evaluated in a restricted $20 \times 20 \times 20$ workspace with eight standard brick types. While this setting enables controlled comparison, it also constrains the resolution, part diversity,

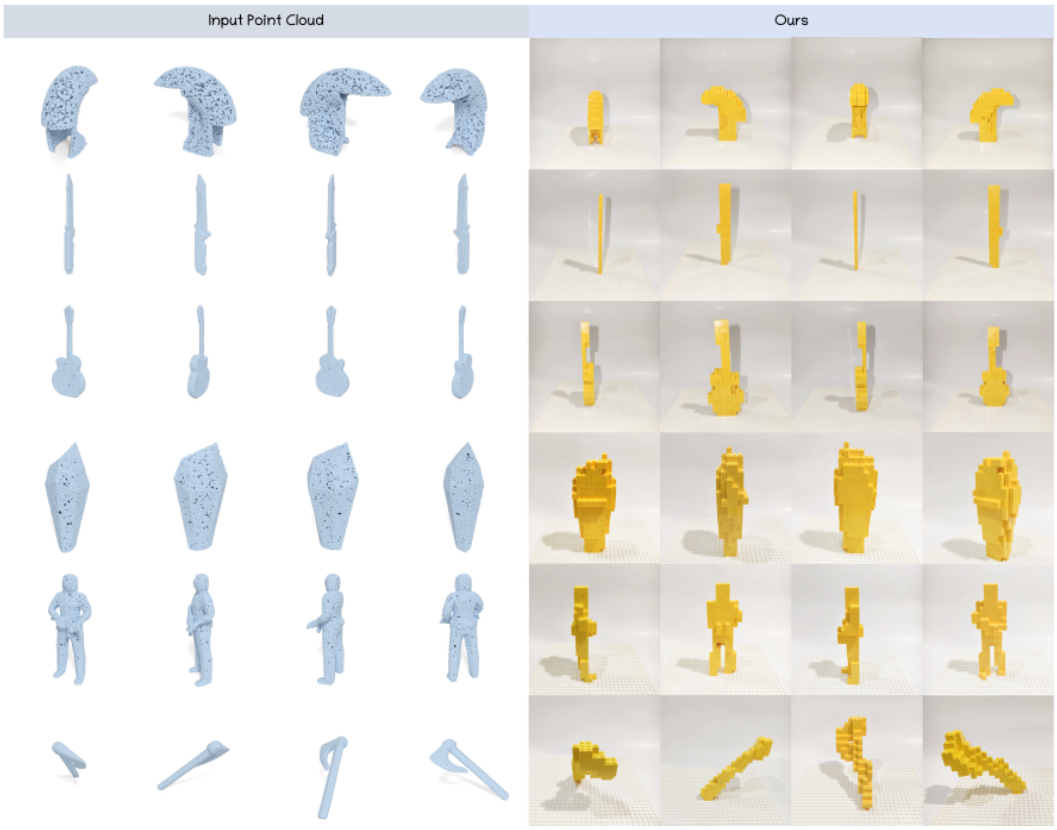

Figure 11: **Physical assembly demonstrations.** We physically construct brick assemblies generated by BrickAnything. The left columns show the input point clouds from different viewpoints, while the right columns show photographs of the corresponding physical constructions. These examples demonstrate the practical buildability of the generated brick structures.

and geometric complexity of the generated structures. The failure cases in Fig. 8 further suggest that our current representation is less effective for shapes whose geometry relies on slender components, subtle local variations, or smooth continuous surfaces. In such cases, the coarse voxelized workspace and limited set of rectangular brick primitives may force the model to produce over-simplified or structurally distorted approximations. Future work will extend the framework to larger workspaces, richer brick libraries, and more diverse connection types.

Our training data is generated by applying Legolization to curated mesh collections, and incorporating human-designed brick models could improve data diversity and construction realism. In addition, our buildability-aware reward is hand-designed from geometric fidelity and stability metrics. Learning more adaptive reward models or incorporating real assembly feedback may further improve physical buildability.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: We have clearly stated the claims made in the abstract and introduction, accurately reflecting the paper's contributions and scope.

   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: We discuss the limitations in Appendix E, including the restricted workspace resolution, limited brick library, reduced performance on complex geometries, dependence on Legolization-generated data, and the hand-designed buildability-aware reward.

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: Our paper does not include theoretical results, and therefore, this question is not applicable to our work.

   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: We provide a detailed description of the model and experimental settings in our paper, ensuring that readers have the necessary information to reproduce the main experimental results. Additionally, we plan to release the code to further enhance reproducibility. and facilitate verification of our results.

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 the code in the supplementary material, but the full processed dataset is not included. The paper and supplementary material describe the data construction and evaluation procedures, and we will release the processed data and detailed reproduction instructions upon acceptance.

   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: We have provided all the necessary details regarding the training and testing process, including data splits, network structure, hyperparameters, and the type of optimizer used.

   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: [N/A]

   Justification: We conducted our experiments and baseline experiments on the same training and testing datasets to ensure a fair comparison.

   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: We provided sufficient information on the computer resources needed to reproduce the experiments in the implementation details section.

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: Our research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

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: We discussed the social impact of this work in the introduction and conclusion sections.

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: No such risks.

    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: We will release the code, data, and models publicly upon the acceptance of the paper.

    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: [N/A]

    Justification: We did not submit any new assets at the time of submission. However, we plan to release well-documented code after the paper's acceptance.

    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: Our research does not involve crowdsourcing nor research with human subjects.

    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: Our research does not involve crowdsourcing nor research with human subjects.

    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: [N/A]

    Justification: The core method development in our research does not involve LLMs as any important, original, or non-standard components.

    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.