LegoACE: Autoregressive Construction Engine for Expressive LEGO® Assemblies

HAO XU, State Key Lab of CAD&CG, Zhejiang University, China
YUQING ZHANG, State Key Lab of CAD&CG, Zhejiang University, China
YIQIAN WU, State Key Lab of CAD&CG, Zhejiang University, China
XINYANG ZHENG, Tsinghua University, China
YUTAO LIU, VAST, China
XIANGJUN TANG, KAUST, Saudi Arabia
YUNHAN YANG, University of Hong Kong, China
DING LIANG, VAST, China
YINGTIAN LIU, Tsinghua University, China
YUANCHEN GUO, VAST, China
YANPEI CAO*, VAST, China
XIAOGANG JIN*, State Key Lab of CAD&CG, Zhejiang University, China

Fig. 1. Our LegoACE generates diverse and expressive LEGO models with robust brick connectivity (left). To demonstrate real-world applicability, on the right we present our sequentially generated outputs (first and third rows) with their corresponding physical LEGO assemblies (second and fourth rows), which faithfully follow LegoACE 's sequential generation procedure and exhibit stable structural integrity in both cases.

Automated LEGO design is challenging due to the extensive variety of LEGO brick types and the necessity of constructing semantically meaningful models from individually meaningless components. Current automatic LEGO generation methods face two key challenges: i) They typically rely on explicit modeling of brick connectivity to ensure structural validity. However, this requires extensive manual annotation, which is labor-intensive as the variety of LEGO primitives increases. This limits training data diversity, restricting the variety of LEGO bricks that can be effectively utilized. ii) To facilitate learning within neural networks, current methods often employ either volume or text-based descriptions to represent LEGO models. However, volumetric representations are computationally expensive and hamper large-scale generative training, while text-based approaches rely on large language models and dedicated text-to-brick mapping rules, introducing a semantic gap between language tokens and 3D brick structures.

To address these challenges, we propose LegoACE, an autoregressive construction engine for expressive LEGO assembly generation conditioned on text prompts or multi-view normal maps. By leveraging the sequential nature of LEGO assemblies, LegoACE implicitly learns connection relationships by modeling the conditional distribution of each brick's position, orientation, and type based on previously placed bricks. This formulation enables us to construct a large-scale dataset, LegoVerse, which comprises over 55,000

*Corresponding author.

SA Conference Papers '25, December 15–18, 2025, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-2137-3/2025/12
https://doi.org/10.1145/3757377.3763881

unique models with 9,314 different brick types. To precisely represent LEGO primitives while minimizing computational overhead, we propose a tokenization strategy, LEGO Native Tokenization Algorithm, which transforms unstructured bricks into compact tokens encoding position, rotation, and type. This enables a decoder-only transformer to autoregressively generate LEGO models based on conditional inputs. Furthermore, to enhance model performance, we apply Direct Preference Optimization (DPO) to fine-tune LegoACE. Our experimental results show that LegoACE demonstrates the ability to generate diverse and expressive LEGO assemblies with robust brick connectivity, significantly advancing the state-of-the-art in LEGO model generation.

CCS Concepts: • **Applied computing → Computer-aided manufacturing**.

Additional Key Words and Phrases: 3D Generation, LEGO, Auto-regressive Generation

ACM Reference Format:
Hao Xu, Yuqing Zhang, Yiqian Wu, Xinyang Zheng, Yutao Liu, Xiangjun Tang, Yunhan Yang, Ding Liang, Yingtian Liu, Yuanchen Guo, Yanpei Cao, and Xiaogang Jin. 2025. LegoACE: Autoregressive Construction Engine for Expressive LEGO® Assemblies. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3757377.3763881

1 INTRODUCTION

Creating semantically meaningful LEGO models from basic primitives typically requires following official building instructions, which are time-consuming to design and restrict flexibility and creativity. Therefore, automating the construction of a LEGO model is a challenging yet meaningful task.

However, existing LEGO model generation methods rely on explicitly modeling the connection relationships between LEGO bricks, requiring annotating each LEGO primitive to specify the exact locations of its connection units (studs and tubes). As the variety of LEGO primitives increases, the annotation process becomes increasingly labor-intensive, posing a significant barrier to scaling such methods to large datasets. Consequently, recent methods are trained on a relatively small dataset [Ge et al. 2024a] ($\sim$ 400) or encompass only a limited set of brick types [Chung et al. 2021; Ge et al. 2024a; Pun et al. 2025] (<30), thereby restricting their diversity, expressiveness, and generalizability. Moreover, unlike text or images, LEGO models lack an inherently structured representation that can be processed by neural networks. While volumetric representations [Ge et al. 2024a] have been proposed to be modeled using generative methods such as 3D diffusion models, their high computational demands significantly hinder the training of large-scale models. Additionally, designing algorithms that reliably convert volumes back into LEGO bricks while preserving connectivity remains a significant challenge. Concurrent with our work, BrickGPT [Pun et al. 2025] explores representing LEGO models using natural language, but its reliance on dedicated text-to-brick mapping rules and the semantic gap between language tokens and 3D brick structures limits output diversity and fidelity.

In this paper, we propose LegoACE, a novel autoregressive pipeline for generating expressive LEGO models conditioned on text prompts or multi-view normal maps. Inspired by the serialized nature of LEGO assemblies, we propose to implicitly learn the connection relationships by modeling the conditional distribution of each brick's position, orientation, and type, conditioned on the bricks that have already been placed. By eliminating the need for exhaustive manual annotations, this implicit formulation enables the creation of a large-scale LEGO dataset, LegoVerse, comprising **55,000** unique models and **9,314** brick types, which far surpasses existing collections and supports complex axis-aligned rotational transformations (Tab. 1). To obtain an efficient representation that can be effectively processed by the neural networks, we introduce a novel LEGO Native Tokenization Algorithm. It discretizes each brick's position and rotation into compact tokens, explicitly encoding spatial transformations while preserving representation compactness. We propose that focusing on the inter-brick relationships can achieve a similar effect to explicitly using geometric information, while offering higher efficiency. Consequently, we encode only brick-type information instead of the full geometry, significantly reducing computational cost while preserving robust assembly quality. We train a decoder-only transformer to autoregressively generate LEGO models, facilitating implicit modeling of LEGO connections and offering greater control over the generation process. We support two conditioning modalities, text prompts and multi-view normal maps, which are encoded using CLIP [Radford et al. 2021] and DINOv2 [Oquab et al. 2024], respectively. Finally, to ensure that the generated models align more closely with human preferences, we incorporate Direct Preference Optimization (DPO) [Rafailov et al. 2023] to fine-tune the trained LegoACE. Our approach achieves high-quality generation of diverse and semantically meaningful LEGO structures with robust brick connectivity. Our code and models can be found at https://xh38.github.io/LegoACE/.

Our contribution can be summarized as follows:

- We propose a novel autoregressive pipeline, LegoACE, for generating diverse and visually rich LEGO models using a wide variety of LEGO bricks, significantly improving diversity and realism compared to previous approaches.
- We introduce a novel LEGO Native Tokenization Algorithm that converts unstructured LEGO models into compact tokens, enabling the autoregressive pipeline to effectively process and generate LEGO structures based on various input conditions..
- We construct a LEGO dataset, LegoVerse, comprising over 55,000 unique models and 9,314 brick types, addressing the scarcity of publicly available LEGO data.

2 RELATED WORK

2.1 LEGO Generation

The automatic construction of LEGO models has been a long-standing challenge since the LEGO Group first introduced the problem. Optimization based LEGO generation considers constructing LEGO models as a combinatorial optimization problem [Gower et al. 1998; Liu et al. 2024b; Luo et al. 2015; Petrovic 2001], while search-based methods explore the solution space to find an appropriate solution [Stephenson 2016; Winkler 2005]. Flexible inputs, such as images or 3D models, are also supported through iterative refinement and deformation, which align the generated LEGO models with the input data [Xu et al. 2019; Zhou et al. 2019, 2023]. To address the

irregularity of LEGO models, prior work [Lee et al. 2018; Testuz et al. 2013] voxelizes the models to simplify processing and then reconstructs the LEGO structure from the voxel representation.

From a machine learning perspective, LEGO models should be transformed into representations that learning algorithms can effectively interpret and process. The inter-brick relationships could be formulated as a graph [Ma et al. 2023; Thompson et al. 2020], which can be generated by generative graph models. Others [Liu et al. 2024a] treat the LEGO model construction as an action sequence and use reinforcement learning to fit the generation process. There are also retrieval-based generation methods for the production of LEGO models [Ge et al. 2024b]. By representing LEGO models with 3D volumes, VAE or diffusion models can learn the distribution of LEGO structures in the volumetric space [Ge et al. 2024a; Lennon et al. 2021], but the high computational demands hinder large-scale training. To achieve a more compact representation, BrickGPT [Pun et al. 2025] uses LLM to describe LEGO models in natural language. But this approach requires dedicated text-to-brick mapping rules and suffers from a semantic gap between language and 3D structures.

The aforementioned methods all rely on explicit annotation of brick connectivity, imposing a labor-intensive burden that restricts the scale and diversity of LEGO datasets. In contrast, we implicitly learn the connection relationships using an autoregressive framework, thereby eliminating the need for connectivity annotations. As a result, we can consider a much broader range of brick types and generate more diverse models than ever before. Furthermore, our novel LEGO Native Tokenization Algorithm is able to effectively represent LEGO bricks while significantly reducing computational overhead.

2.2 Auto-Regressive 3D Generation

Neural networks, particularly convolutional ones, are optimized for grid-like data but struggle to handle unstructured mesh data. Recently, many approaches have attempted to generate meshes directly in an auto-regressive manner to address this challenge. Since a mesh is composed of vertices and faces, which carry different semantic meanings, prior work uses two separate transformers to independently learn their respective distributions [Nash et al. 2020]. To simplify this process with a single transformer, a neural representation of coordinates is proposed [Chen et al. 2024a]. MeshGPT [Siddiqui et al. 2024] reduces sequence length by tokenizing the mesh representation and learning mesh tokens with a GPT-like transformer. Subsequent methods have explored various directions, including conditioning strategies to control the generation process [Chen et al. 2025, 2024b; Tang et al. 2025; Zhao et al. 2025b], more efficient tokenization techniques to scale up the number of generated mesh faces [Weng et al. 2024b], functionality-specific designs [Gao et al. 2024], and coarse-to-fine generation approaches [Weng et al. 2024a].

Similar to meshes, LEGO models also possess irregular structures. However, the aforementioned methods cannot be directly applied to the LEGO model due to its unique challenges, such as discrete axis-aligned rotations and a diverse set of brick types.

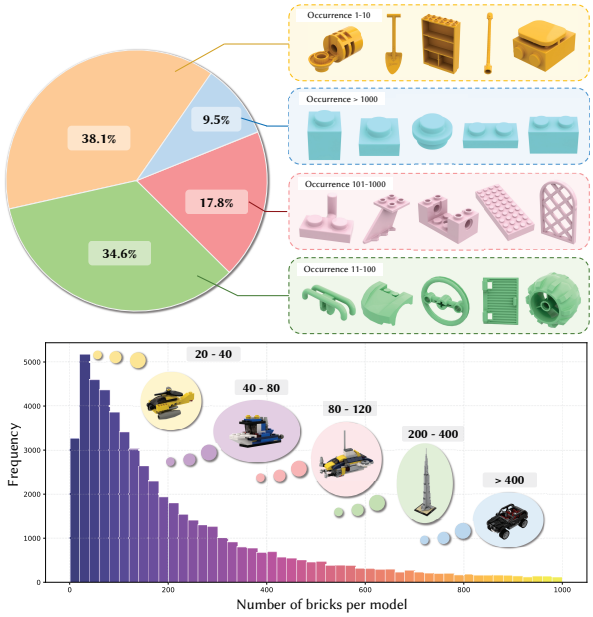


Fig. 2. **LegoVerse Dataset.** We present the distribution of brick occurrence frequencies (top) and the distribution of models by brick count (bottom), demonstrating the richness and diversity of our LegoVerse dataset.

Table 1. **Comparison of LEGO model datasets.** Our dataset comprises the largest collection of models and brick types and supports complex rotation transformation.

Dataset	# of models	# of bricks	# of rotation types
RAD-1k [2021]	1,000	2	2
LEGO micro-building [2024a]	400	28	20
StableText2Lego [2025]	47,000	8	2
LegoVerse (Ours)	55,000	9,314	48

3 METHOD

The overview of our pipeline is illustrated in Fig. 3. We first construct a large-scale LegoVerse dataset and a corresponding LEGO brick library (Sec. 3.1). Then we introduce an efficient tokenization algorithm that automatically encodes LEGO bricks into discrete tokens which could be processed by neural network (Sec. 3.2). Building on this representation, we present LegoACE, an autoregressive model that generates physically plausible LEGO structures conditioned on either text prompt or multi-view normal images (Sec. 3.3).

3.1 LegoVerse Dataset

Unlike mesh generation models, which benefit from a wealth of available datasets, the lack of datasets remains a major challenge in LEGO generation models training. As shown in Table 1, the LEGO micro-building dataset [Ge et al. 2024a] comprises only 400 models built from 28 brick types, focuses solely on miniature architectural structures, and only supports 20 axis-aligned rotations. BrickGPT [Pun et al. 2025] introduces a significantly larger dataset, StableText2Lego, with over 40,000 models. However, it uses only 8 types of basic bricks and includes only upward-facing rotations, limiting the diversity of geometric configurations.

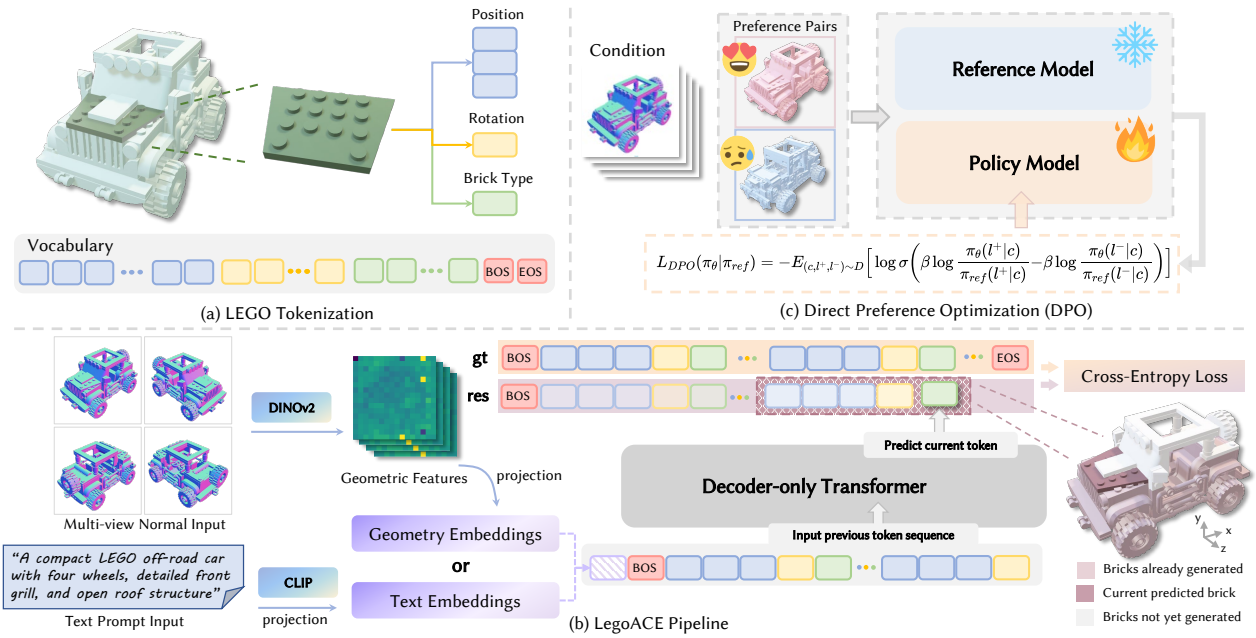


Fig. 3. **Overview of our pipeline. (a)** We begin by tokenizing the LEGO model into a sequence of tokens. For a single LEGO brick, we convert its position, rotation, and type attributes into tokens within a unified vocabulary. The full LEGO model is tokenized by processing each brick individually, following a predefined *z-x-y* spatial ordering. **(b)** LegoACE supports two types of conditional inputs. Geometry embeddings are derived from multi-view normal maps via DINOv2, while text embeddings are obtained using CLIP. These embeddings are prepended to the token sequence to provide conditional guidance. A decoder-only Transformer then autoregressively generates the token sequence, proceeding token by token. For each brick, the tokens are generated in the order of *position*, *rotation*, and *brick type*. In the figure, the currently predicted token is highlighted with a shaded background. Once the entire sequence is generated, the condition embeddings are discarded, and the remaining tokens are detokenized into the final LEGO model. **(c)** To further fine-tune the trained LegoACE conditioned on normal maps, we apply DPO and construct a preference dataset. We duplicate the trained LegoACE into a frozen reference model and a learnable policy model. The preference pairs are then utilized to optimize the policy model, with regularization achieved through the frozen reference model.

To address the lack of a comprehensive dataset, we collect a new dataset, LegoVerse, comprising about 55,000 models, spanning diverse LEGO categories, including buildings, vehicles, characters, spaceships, animals, furniture, and more. As shown in Fig. 2, we summarize key statistics for LegoVerse, illustrating the distribution of brick occurrence frequencies and the distribution of models by brick count. We also showcase representative brick types and example models from our dataset. Furthermore, the distribution of brick counts per model reveals a wide range of structural complexity, from simple builds with a few dozen bricks to intricate constructions composed of over thousands of bricks.

Our dataset comprises 9,314 unique brick types, encompassing a broad range of axis-aligned orientations. In addition to frequently used bricks, our LegoVerse dataset includes a variety of specialized LEGO components such as wheels, steering wheels, doors, and windows, highlighting its richness and diversity. For all brick types utilized across the models in our dataset, we construct a brick library $\mathcal{T}$ for subsequent use in in Sec. 3.2. Further details on the brick library's construction can be found in Sec. B of our supplementary file.

3.2 LEGO Native Tokenization

Compared to other 3D representations like meshes or volumes, LEGO models are composed of discrete bricks drawn from a predefined library and assembled under connectivity constraints. Bricks cannot be abstracted as points or cubes, as such representations fail to capture their full characteristics. A complete description requires not only the position of each brick, but also its orientations and geometric properties. To effectively tokenize LEGO models while preserving their natural structure, as shown in Fig. 3 (a), we propose a novel non-learning LEGO Native Tokenization method that explicitly represents each brick's attributes.

LEGO models are tokenized on a per-brick basis. Each brick's position, rotation, and brick type are encoded into a token set as:

$$\mu = [\mathcal{P}, \mathcal{R}, \mathcal{T}], \tag{1}$$

where $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{T}$ represent tokens for position, rotation, and brick type, respectively. The tokens of all LEGO models in our LegoVerse dataset are concatenated to form a unified vocabulary:

$$\mathcal{V} = \mathcal{V}_{\mathcal{P}} \cup \mathcal{V}_{\mathcal{R}} \cup \mathcal{V}_{\mathcal{T}}, \tag{2}$$

where $\mathcal{V}_{\mathcal{P}}$, $\mathcal{V}_{\mathcal{R}}$ and $\mathcal{V}_{\mathcal{T}}$ denote the vocabularies of position, rotation, and brick type tokens, respectively.

In the following sections, we detail the computation of position, rotation, and brick type tokens, along with the tokenization order in a LEGO model.

Position tokens. We define the position of a brick as a relative translation in world coordinates, represented by $\mathbf{p} = [x, y, z]^T$, applied to its original position in the brick library $\mathcal{T}$. Developing an effective tokenization scheme for these continuous positions

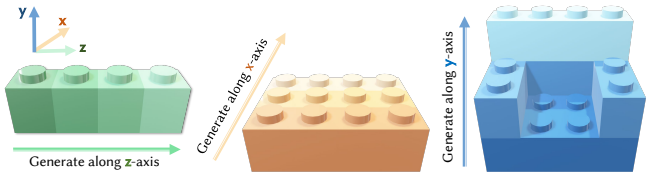


Fig. 4. **The tokenization order.** LEGO models are tokenized following the z-x-y order: first along the z-axis (dark green to light green), then along the x-axis (dark yellow to light yellow), and finally along the y-axis (dark blue to light blue).

requires careful consideration of both geometric precision and computational efficiency. While the original relative translation provides precise spatial information, its unbounded nature creates challenges for discrete token representation, including infinite theoretical vocabulary size and lack of translation invariance.

We introduce a relative linear encoding scheme to overcome these limitations. First, we establish a local coordinate system by computing the minimum bounding box corner $\mathbf{p}_{min} = [x_{min}, y_{min}, z_{min}]^T$ over all bricks in the model that we are processing. Then, we perform precision-preserving quantization by converting positions to the position token $\mathcal{P}$ (represented as integer LDU[1] offsets) through:

$$\mathcal{P} = \lfloor \mathbf{p} - \mathbf{p}_{min} \rceil = \left[\lfloor x - x_{min} \rceil, \lfloor y - y_{min} \rceil, \lfloor z - z_{min} \rceil \right], \quad (3)$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer LDU value.

Rotation tokens. Representing brick orientations presents unique challenges due to the continuous nature of 3D rotations. Given that LEGO bricks predominantly feature axis-aligned orientations in the LEGO models, the nine-dimensional parameterization of rotation matrices $r \in \mathcal{R}^{3\times3}$ is both redundant and incompatible with discrete token representations.

To address this, we construct a restricted axis-aligned rotation set $\mathcal{V}_{\mathcal{R}}$, containing only $N_{\text{rot}} = 48$ canonical orientations (including mirrored cases) that cover all axis-aligned permutations of LEGO bricks. For non-axis-aligned rotations encountered in raw data, we project them to the closest valid orientation in $\mathcal{V}_{\mathcal{R}}$ via:

$$\mathcal{R} = \underset{\mathbf{R}_i \in \mathcal{V}_{\mathcal{R}}}{\operatorname{argmin}} \left\| \mathbf{r} - \mathcal{M}_{\text{rot}}^{-1}(\mathbf{R}_i) \right\|_F, \quad (4)$$

where $||\cdot||_F$ is the Frobenius Norm measuring the distance between rotation matrices, and $\mathcal{M}_{\text{rot}}$ is a predefined mapping from the rotation matrix r_i to rotation token R_i. We leverage the indices of the closest match as our rotation tokens $\mathcal{R}$.

Brick type tokens. We assign a brick type identifier $\mathcal{T} \in \mathcal{N}$ for each brick in the brick library $\mathcal{T}$. This enables a deterministic mapping from each brick in the LEGO model to a corresponding discrete brick type token $\mathcal{T}$. This abstraction discards redundant geometric details that drive up computational cost, instead focusing on relationships among brick types, which are governed by specific connectivity rules. As shown in Sec. 4, this simplified formulation is effective for generating stable LEGO models and greatly reduces computational overhead.

[1]The LDraw Unit (LDU) is the unit of measurement used in the LDraw format, which is introduced in detail in Sec. A of the supplementary material

Tokenization Order. For the sequence order of the entire LEGO model, we adopt the z-x-y ordering, as shown in Fig. 4. The underlying model is generated sequentially from bottom to top, with each LEGO brick being placed one by one. By learning this sequential distribution, we achieve implicit learning of the inter-brick relationships between LEGO bricks.

3.3 LegoACE

3.3.1 Model Architecture. We convert the LEGO models into tokens, which can be learned by the auto-regressive generation model, as shown in Fig. 3 (b). Then we adopt the GPT-like Transformer architecture LLaMA [Touvron et al. 2023a,b] as our backbone. To condition token sequence generation on condition information, we assign the condition embeddings at the beginning of the sequence, followed by the special `BOS` token, which marks the beginning of the token sequence. In this paper, we focus on conditional generation tasks based on two types of input: textual descriptions and multi-view normal maps. For textual descriptions, we utilize CLIP [Radford et al. 2021] to encode the input into rich semantic feature representations. For the multi-view normal maps, we leverage DINOv2 [Oquab et al. 2024] to extract geometric features that capture both fine-grained details and high-level semantics. The resulting conditional features from both modalities are subsequently projected into the model's embedding space through a learnable linear layer.

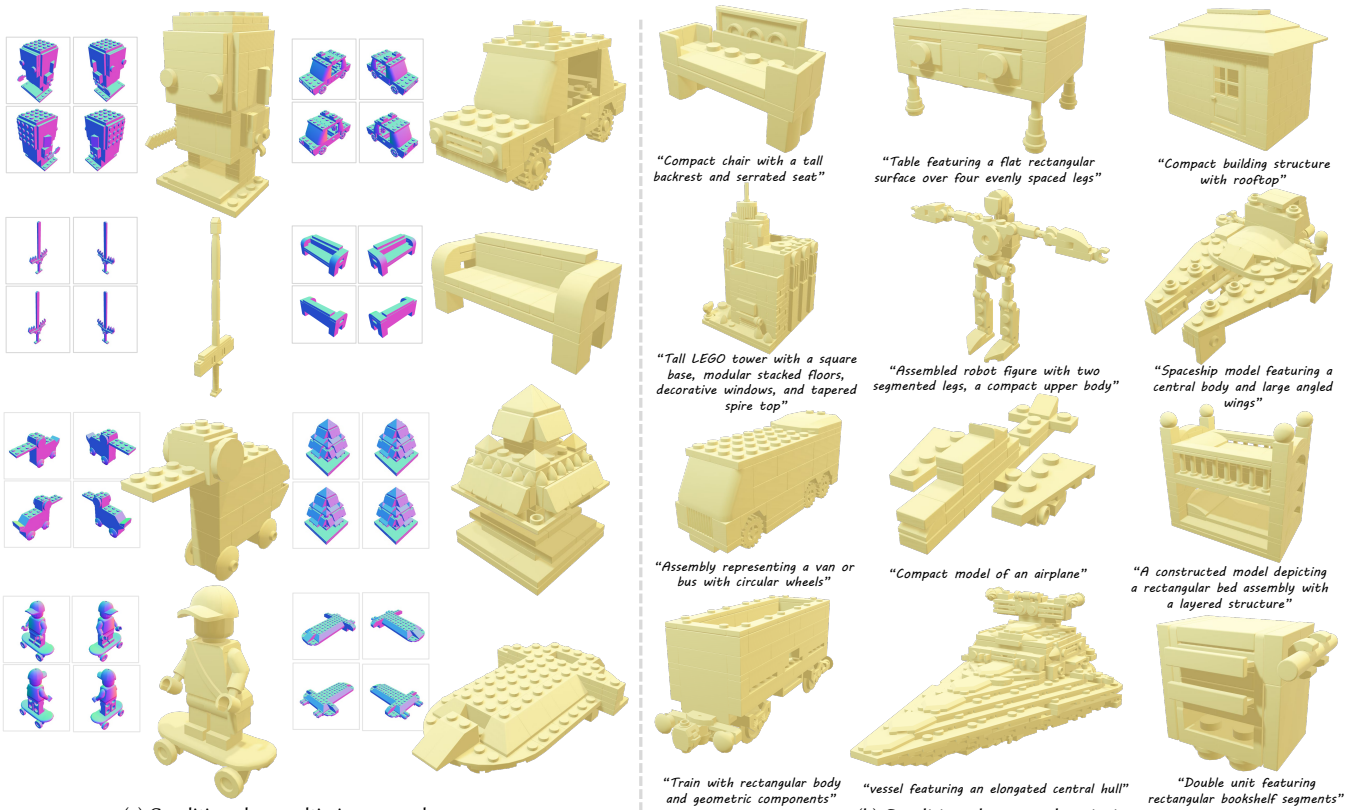
We sample the next token from the predicted probability distribution of the next token based on the previous token sequence and the condition embeddings, which implicitly includes the connectivity between bricks:

$$p\left(\mu_t | c, \mu_0, ..., \mu_{t-1}\right) = G\left(c, \{\mu_0, \mu_1, ..., \mu_{t-1}\}\right), \quad (5)$$

where μ_k denotes the k-th token in the sequence, c represents the condition embedding, and G denotes our LegoACE.

3.3.2 Model Training and Inference. We process LEGO models in the LegoVerse dataset (Sec. 3.1) by tokenizing their bricks based on the predefined order (Sec. 3.2). For multi-view normal map conditioning, we render normal maps from the ground-truth LEGO models to serve as conditional input during training. Leveraging the ability to render normal maps from any subset of bricks and inspired by the serialized nature of LEGO models, we propose a data augmentation method for multi-view normal map conditioning. For each LEGO model, we randomly extract a subsequence based on the predefined order and render the corresponding normal maps for that partial structure. For text descriptions, we obtain prompts using Gemini-2-Flash [Google 2024] to generate captions from rendered RGB images of LEGO models. Since meaningful prompts cannot be reliably extracted from incomplete models, the text-conditioned version of LegoACE is trained only on complete text–model pairs without data augmentation.

Our unified token vocabulary includes different attributes of LEGO bricks. To ensure valid LEGO model generation, during the inference process, we selectively enable only the relevant subset of vocabulary. For example, when generating position tokens, we only consider the position-related part of the vocabulary, while the rotation and type parts are masked out. The `EOS` token, which marks



(a) Conditioned on multi-view normal maps (b) Conditioned on text descriptions

Fig. 5. Our generated results based on (a) multi-view normal maps, and (b) text descriptions.

the end of the sequence, is only unmasked after the full set of tokens representing a LEGO brick has been generated.

3.3.3 Post-training with DPO. After training, LegoACE is capable of generating diverse LEGO models. While conditioning LegoACE on normal maps generally yields plausible results, the outputs occasionally diverge from the input normals due to the diverse range of LEGO brick types and the inherent complexity of their combinations. To further enhance the quality and alignment of models generated from multi-view normal maps, we subsequently apply DPO[Rafailov et al. 2023] to fine-tune the trained LegoACE model, as shown in Fig. 3 (c).

DPO aligns a fine-tuned model with human preferences while preventing from deviating too far from the original network, achieved by comparing preferred and dispreferred samples using two copies of the original model: a frozen reference model and a trainable policy model. The loss is computed as follows:

$$\mathcal{L}_{DPO}\left(\pi_\theta|\pi_{ref}\right) = \\ -\mathbb{E}_{(c,l^+,l^-)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(l^+|c)}{\pi_{ref}(l^+|c)} - \beta\log\frac{\pi_\theta(l^-|c)}{\pi_{ref}(l^-|c)}\right)\right], \quad (6)$$

where β is the coefficient that balances preferred and dispreferred terms, c is the input condition. l^+ and l^- are positive and negative labels, $\mathcal{D}$ refers to the dataset consisting of data pairs of condition, positive output, and negative output$\{c, l^+, l^-\}$.

Since multi-view normal maps provide strong geometric priors, generated results that more closely resemble the ground truth are considered more aligned with user preferences. To capture this, we use Chamfer Distance as the similarity metric to construct preference pairs. Given the input images, we generate two LEGO models and form two image-model pairs. If one model exhibits a significantly smaller Chamfer Distance to the ground truth than the other, it is considered to have a substantial advantage in geometric fidelity. We then assign a positive label to the LEGO model with the lower Chamfer Distance. Based on this preference rule, we construct a preference dataset required for DPO training.

4 EXPERIMENTS

4.1 Implementation Details

For the text encoder CLIP and the image encoder DINOv2 used to encode conditional inputs, we use the implementation from the transformers library [Wolf et al. 2020]. The vocabulary includes 1,280 position tokens, 48 rotation tokens (24 axis-aligned rotations and their reflections), and 9,314 type tokens, along with 2 special tokens BOS and EOS. Our model contains 243M parameters.

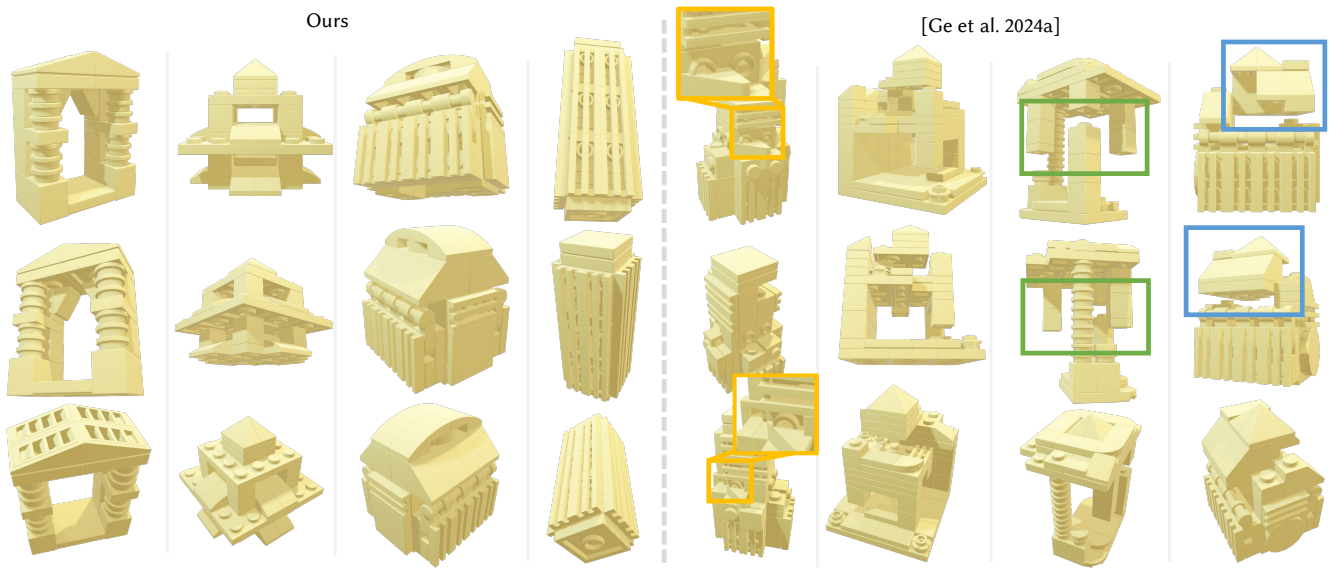


Fig. 6. **Unconditional qualitative comparison.** We compare our method with the unconditional baseline [Ge et al. 2024a], with both models trained on the *LEGO Micro Building* dataset. [Ge et al. 2024a] exhibits issues such as intersections (highlighted with orange boxes and zoomed-in views), asymmetrical and semantically implausible structures (highlighted with green boxes), as well as disconnected components that make the generated models invalid (highlighted with blue boxes). In contrast, our method consistently produces LEGO structures with plausible geometry and robust connectivity.

Table 2. **Unconditional quantitative comparison.** We evaluate our method and the baseline [Ge et al. 2024a] using multiple metrics, including distance to the ground truth (1-NNA, MMD), runtime, and connection rate. Our method not only achieves consistently better numerical results across all metrics but also requires significantly less training time. This demonstrates the high efficiency and strong representational capacity of our serialized representation.

Method	MMD ↓		1-NNA ~50%		COV ↑		Runtime ↓		Connected
	CD	EMD	CD	EMD	CD	EMD	Train	Test	Rate ↑
Ge et al.	11.57	2.28	65.22%	65.68%	12.54	1.66	20 h	3 min	53%
Ours	3.52	0.15	56.63%	52.00%	9.55	1.23	10 min	3 sec	82%

The pre-training of multi-view normal map conditioned LegoACE is conducted on 8 A100 GPUs for two days. We randomly selected 50,000 models as the training set and used the remaining 5,000 models for validation. The learning rate follows a cosine decay, starting from 1e-4 and decaying to 1e-5. For post-training with DPO, we create 5,000 data pairs. The DPO training process takes 15 minutes on 8 A100 GPUs for 3 epochs. The text-conditioned LegoACE is also trained on 8 A100 GPUs for two days, using 54,000 models as the training set and the remaining 1,000 models for validation. During inference, we use both top-k ($k = 10$) [Fan et al. 2018] and top-p ($p = 0.95$) [Holtzman et al. 2020] sampling in conjunction with our dynamic masking strategy.

4.2 Results

Fig. 5 presents LEGO assemblies generated by LegoACE conditioned on multi-view normal input (Fig. 5 (a)) and text input (Fig. 5 (b)). These examples demonstrate our model's capacity to produce diverse and semantically aligned designs, ranging from simple characters and animals to intricate vehicles and robotic structures. This expressive power across both conditioning modalities stems from our extensive dataset and novel tokenization strategy.

4.3 Comparison

We compare our method with an unconditional LEGO generation baseline [Ge et al. 2024a]. To ensure a fair comparison, we train an unconditional version of LegoACE on their open-source dataset, LEGO micro-building. Note that while [Ge et al. 2024a] leverages explicit connection information in the training dataset, our method processes the same dataset using our own tokenization approach, without relying on any explicit connectivity annotations. The training details are discussed in Sec. C of the supplementary file.

We also evaluate LegoACE against both text-conditioned and image-conditioned baselines. For text conditioning, we compare with the text-to-LEGO model BrickGPT [Pun et al. 2025] and general text-to-3D methods Hunyuan3D 2.5 [Zhao et al. 2025a], Shap-E [Jun and Nichol 2023] and TRELLIS [Xiang et al. 2024]. For image conditioning, since no LEGO-specific image-to-3D methods exist, we compare with general image-to-3D models such as SPAR3D [Huang et al. 2025], TripoSG [Li et al. 2025], and Hi3DGen [Ye et al. 2025]. Finally, to enable a fair comparison, for methods that cannot generate LEGO models, we apply a legolization process[2] to convert the generated general meshes into LEGO-compatible models.

Qualitative Comparison. As shown in Fig. 6, the baseline unconditional method [Ge et al. 2024a], despite being trained with explicitly annotated connection relationships in volumetric representations, exhibits intersections, asymmetrical and semantically implausible

[2]https://www.cgtrader.com/free-3d-models/scripts-plugins/modelling/legoit-blender-addon-lego

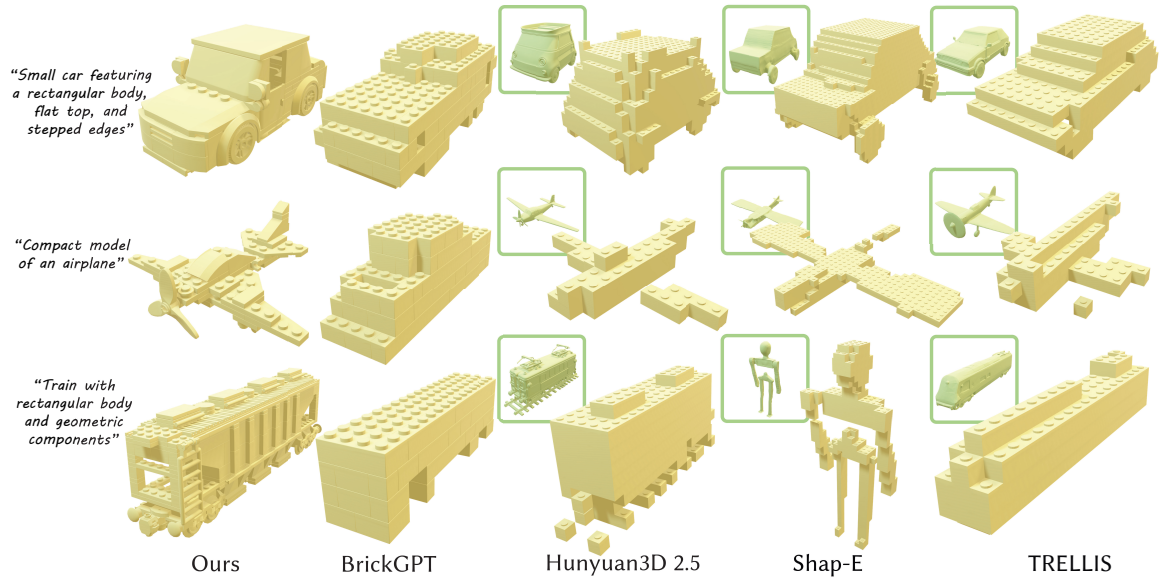


Fig. 7. **Text-conditional qualitative comparison.** We compare our method with the text-to-LEGO method BrickGPT [Pun et al. 2025], general text-to-3D methods Hunyuan3D 2.5 [Zhao et al. 2025a], Shap-E [Jun and Nichol 2023] and TRELLIS[Xiang et al. 2024], all of which share the same text input. For these general-purpose models, we convert the generated meshes (shown in the top-left corners and highlighted with green boxes) into LEGO-compatible models for fair comparison. It should be noted that the airplane out-of-distribution of BrickGPT.

structures, as well as brick disconnectivity that makes the generated models invalid. In contrast, our LegoACE consistently generates LEGO models with robust brick connectivity.

This suggests that by serializing LEGO models, the autoregressive model effectively learn brick connection relationships in an implicit manner.

For text conditioning, as shown in Fig. 7, all methods use the identical text prompts sampled from the StableText2Lego [Pun et al. 2025] dataset and the validation set of our LegoVerse. BrickGPT, which shares a similar objective with our work, can generate LEGO models only with basic brick types. This limitation results in a loss of detail. For example, car wheels are represented using basic rectangular bricks, whereas our LegoACE successfully utilizes actual LEGO wheel components. This improvement is enabled by our curated LegoVerse dataset and novel brick tokenization method, which together enhance the model's understanding of diverse brick types. Moreover, although general 3D generation methods are capable of producing highly detailed and realistic 3D meshes, their outputs cannot be effectively transformed into well-connected and LEGO-compatible structures, making them unsuitable for real-world applications.

For image conditioning, as shown in Fig. 8, we use LEGO-style images generated by the GPT-4. To feed our multi-view-normal conditioned LegoACE, we render normal maps from meshes generated by Hunyuan3D 2.5 using the same reference images. While the results of general image-to-3D methods cannot be effectively converted into LEGO-compatible assemblies, our LegoACE generates structures that faithfully follow the input normal maps and are assembled with appropriate LEGO bricks.

Quantitative Comparison. Since general 3D generation methods cannot natively produce LEGO-compatible structures, and the mesh-to-LEGO legalization tools fall outside the scope of this paper, we limit our quantitative comparison in this section to techniques designed specifically for LEGO.

For the unconditional LEGO generation, following [Chen et al. 2021; Ge et al. 2024a; Hertz et al. 2022], we compute the Minimum Matching Distance (MMD), 1-NNA and COV to quantitatively evaluate the generated results. We generate 400 models for each method as the set of generated data, and select 400 models from the LEGO micro-building dataset as the set of real data. For each generated or real LEGO model, we sampled 2,048 points, and the MMD, 1-NNA and COV values are computed based on Chamfer Distance (CD) and Earth Mover's Distance (EMD) between the sampled point clouds. We also evaluate the training and testing time for each method. For connectivity evaluation, we sample 100 models from each method. By manually inspecting each model for issues such as overlapping bricks and floating components, we compute the connected rate, which denotes the proportion of models that achieved perfect connectivity. As shown in Table 2, our method significantly outperforms the volume-based diffusion approach in both MMD and 1-NNA metrics, indicating better output quality and geometric fidelity. The slightly inferior performance of our method on the COV metric may be attributed to the considerably smaller parameter size, which in turn could constrain the diversity of the generated results. As for runtime, our method converges after only 10 minutes of training, whereas the diffusion method [Ge et al. 2024a] requires up to 20 hours of training. Our method also achieves higher inference performance, requiring only 3 seconds to infer a model, while [Ge et al. 2024a] takes 3 minutes, including 2.5 minutes for

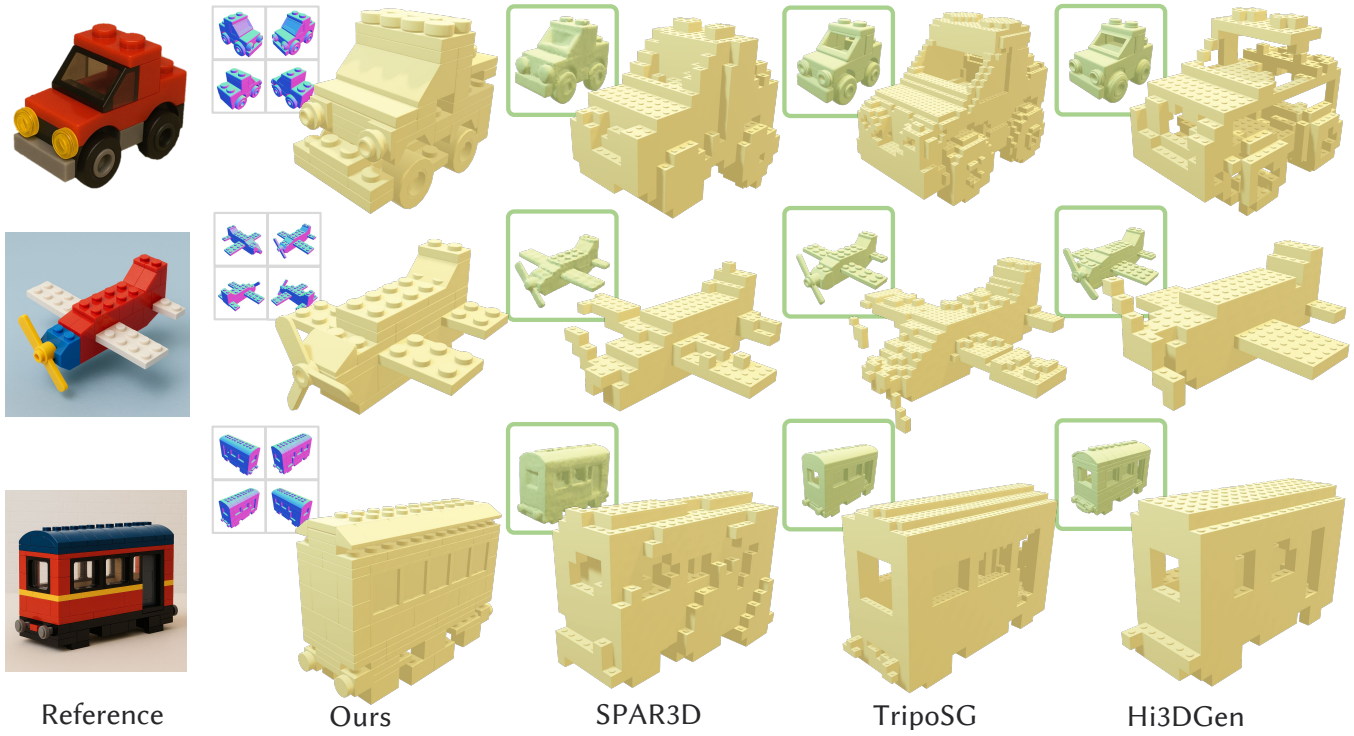


Fig. 8. **Image-conditional qualitative comparison.** We compare our method with several image-to-3D models, including SPAR3D [Huang et al. 2025], TripoSG [Li et al. 2025], and Hi3DGen [Ye et al. 2025]. We generate LEGO-style reference images using the GPT-4 image model to serve as inputs for these baseline methods. Since our method requires multi-view normal maps as input, we leverage Hunyuan3D 2.5 [Zhao et al. 2025a] to generate LEGO-style meshes from the RGB reference images, and then render multi-view normal maps from these meshes to serve as input to our model. For these general-purpose models, we convert the generated meshes (shown in the top-left corners and highlighted with green boxes) into LEGO-compatible models for fair comparison.

inference and 0.5 minutes for post-processing. Regarding the connection rate, our approach, which implicitly learns the inter-brick relationships, enables more efficient and stable generation of models with appropriate layouts. As a result, we achieve a connection rate of 82%, significantly outperforming volume-based methods that rely on explicit geometric information, which only reach 53%. This demonstrates the advantage of modeling inter-brick relationships for producing well-connected structural layouts.

For the text-conditioned variant of LegoACE, we sample prompts from the StableText2Lego [Pun et al. 2025] dataset and the validation set of our LegoVerse. Using these shared prompts, we generate 180 models respectively with LegoACE and BrickGPT. To assess semantic alignment, we compute the CLIP score between the input text prompts and the generated results. Our method achieves a CLIP score of 23.25, compared to 21.92 for BrickGPT, demonstrating superior alignment with the input text and stronger semantic expressiveness. To evaluate human preferences, we conducted a user study with 35 participants who each compared 20 text-conditioned models generated by our method and by BrickGPT. Participants were asked to assess the aesthetics and structural reasonableness of each model with respect to the input text. Our method achieved a preference score of 87%, significantly outperforming BrickGPT, which received only 13%. Also, compared to BrickGPT's requirement of 10 tokens per brick, our strategy uses only 5 tokens, yielding a more compact and efficient representation while significantly enhancing output quality. Moreover, their tokenization scheme requires a unique text description for each brick type, making it impractical for large-scale models; whereas our approach scales effortlessly to extensive datasets without additional manual effort.

Table 3. **Ablation study on subsequence data augmentation and DPO.** We evaluate the generated results using the Earth Mover's Distance (EMD) and Chamfer Distance (CD) with ground truth, and additionally measure human preference scores to assess the semantic quality of the outputs.

Method	Base line	+ Subseq.	+ DPO (Ours)
EMD↓	9.71	6.56	5.92
CD ↓	13.23	10.32	7.96

4.4 Ablation Study

As described in Sec. 3.3, we propose a subsequence sampling strategy to augment the training dataset when using normal maps as the conditional inputs, and employ Direct Preference Optimization (DPO) to improve the alignment with the input conditional inputs.

In this section, we present ablation studies to evaluate the effectiveness of the subsequence sampling strategy and DPO, evaluating reconstruction quality using Chamfer Distance (CD) and Earth

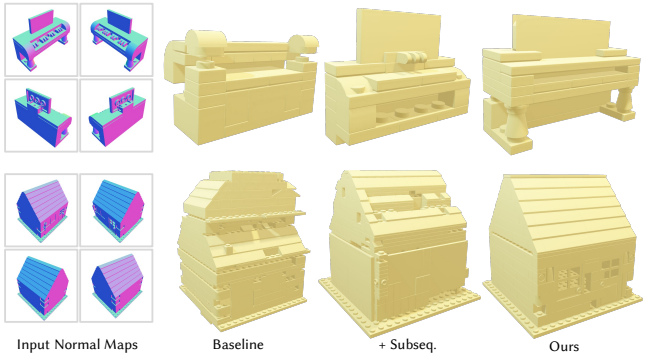


Fig. 9. **Ablation study on subsequence data augmentation and DPO.** Without subsequence augmentation, the generated outputs exhibit notable deviations from the intended input. Incorporating augmentation enhances structural fidelity—for example, improving alignment in elements such as the music stand of a piano and the roof of a house. The application of DPO further refines fine-grained details, including the shape of the piano legs and the contour of the roof.

Mover's Distance (EMD). We compare our full model with two variants: (1) a baseline model trained without subsequence sampling or DPO, and (2) a model trained with subsequence sampling but without DPO. As shown in Tab. 3 and Fig. 9, the reduced distance to the ground truth demonstrates that the subsequence sampling strategy enhances the quality and geometric fidelity of the generated models, while the incorporation of DPO further improves these results by fine-tuning the model using a preference dataset.

5 LIMITATIONS AND CONCLUSIONS

5.1 Limitations

Although our approach produces promising results in LEGO model generation, it still has several limitations. First, our dataset, while comprising 55,000 models, is relatively small for training a high-fidelity 3D generative model, potentially constraining performance. Second, since the model learns connectivity implicitly and lacks explicit structural constraints, some generated instances contain disconnected components. Strengthening structural connectivity thus remains an important avenue for future work.

5.2 Conclusions

We present a novel approach LegoACE for LEGO model generation. To facilitate expressive model training, we create a large-scale LEGO dataset, LegoVerse, comprising **55,000** unique models and **9,314** brick types. By introducing a LEGO native tokenization method, we convert each LEGO model into a sequence of tokens, enabling learning through state-of-the-art autoregressive generative models. Our method supports a wide range of input conditions, and we have thoroughly validated its effectiveness under unconditional generation, multi-view normal inputs, and text prompts. It is capable of constructing diverse models using an unprecedented variety of LEGO bricks.

ACKNOWLEDGMENTS

Xiaogang Jin was supported by the National Natural Science Foundation of China (Grant Nos. 62472373, 62036010).

REFERENCES

Sijin Chen, Xin Chen, Anqi Pang, Xianfang Zeng, Wei Cheng, Yijun Fu, Fukun Yin, Yanru Wang, Zhibin Wang, Chi Zhang, Jingyi Yu, Gang Yu, Bin Fu, and Tao Chen. 2024a. MeshXL: Neural Coordinate Field for Generative 3D Foundation Models. In *Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024.*

Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Xin Chen, Zhongang Cai, Lei Yang, Gang Yu, Guosheng Lin, and Chi Zhang. 2025. MeshAnything: Artist-Created Mesh Generation with Autoregressive Transformers. In *The Thirteenth International Conference on Learning Representations, ICLR 2025.*

Yiwen Chen, Yikai Wang, Yihao Luo, Zhengyi Wang, Zilong Chen, Jun Zhu, Chi Zhang, and Guosheng Lin. 2024b. MeshAnything V2: Artist-Created Mesh Generation With Adjacent Mesh Tokenization. *arXiv preprint arXiv:2408.02555* (2024).

Zhang Chen, Yinda Zhang, Kyle Genova, Sean Ryan Fanello, Sofien Bouaziz, Christian Häne, Ruofei Du, Cem Keskin, Thomas A. Funkhouser, and Danhang Tang. 2021. Multiresolution Deep Implicit Functions for 3D Shape Representation. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021.* 13067–13076.

Hyunsoo Chung, Jungtaek Kim, Boris Knyazev, Jinhwi Lee, Graham W. Taylor, Jaesik Park, and Minsu Cho. 2021. Brick-by-Brick: Combinatorial Construction with Deep Reinforcement Learning. In *Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021.* 5745–5757.

Angela Fan, Mike Lewis, and Yann N. Dauphin. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Volume 1: Long Papers*, Iryna Gurevych and Yusuke Miyao (Eds.). 889–898.

Daoyi Gao, Yawar Siddiqui, Lei Li, and Angela Dai. 2024. MeshArt: Generating Articulated Meshes with Structure-guided Transformers. *arXiv preprint arXiv:2412.11596* (2024).

Jiahao Ge, Mingjun Zhou, Wenrui Bao, Hao Xu, and Chi-Wing Fu. 2024b. Creating LEGO Figurines from Single Images. *ACM Trans. Graph.* 43, 4 (2024), 153:1–153:16.

Jiahao Ge, Mingjun Zhou, and Chi-Wing Fu. 2024a. Learn to Create Simple LEGO Micro Buildings. *ACM Trans. Graph.* 43, 6 (2024), 249:1–249:13.

Google. 2024. Gemini 2.0 Flash. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash.

Rebecca Gower, Agnes Heydtmann, and Henrik Petersen. 1998. Lego: Automated model construction. *European Study Group with Industry* (1998).

Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2022. SPAGHETTI: editing implicit shapes through part aware generation. *ACM Trans. Graph.* 41, 4 (2022), 106:1–106:20.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. In *8th International Conference on Learning Representations, ICLR 2020.*

Zixuan Huang, Mark Boss, Aaryaman Vasishta, James M. Rehg, and Varun Jampani. 2025. SPAR3D: Stable Point-Aware Reconstruction of 3D Objects from Single Images. *arXiv preprint arXiv:2501.04689* (2025).

Heewoo Jun and Alex Nichol. 2023. Shap-E: Generating Conditional 3D Implicit Functions. *arXiv preprint arXiv:2305.02463* (2023).

Seung-Mok Lee, Jae Woo Kim, and Hyun Myung. 2018. Split-and-Merge-Based Genetic Algorithm (SM-GA) for LEGO Brick Sculpture Optimization. *IEEE Access* 6 (2018), 40429–40438.

Kyle Lennon, Katharina Fransen, Alexander O'Brien, Yumeng Cao, Matthew Beveridge, Yamin Arefeen, Nikhil Singh, and Iddo Drori. 2021. Image2Lego: Customized LEGO Set Generation from Images. *arXiv preprint arXiv:2108.08477* (2021).

Yangguang Li, Zi-Xin Zou, Zexiang Liu, Dehu Wang, Yuan Liang, Zhipeng Yu, Xingchao Liu, Yuan-Chen Guo, Ding Liang, Wanli Ouyang, and Yan-Pei Cao. 2025. TripoSG: High-Fidelity 3D Shape Synthesis using Large-Scale Rectified Flow Models. *arXiv preprint arXiv:2502.06608* (2025).

Ruixuan Liu, Alan Chen, Weiye Zhao, and Changliu Liu. 2024a. Physics-Aware Combinatorial Assembly Planning using Deep Reinforcement Learning. *arXiv preprint arXiv:2408.10162* (2024).

Ruixuan Liu, Kangle Deng, Ziwei Wang, and Changliu Liu. 2024b. StableLego: Stability Analysis of Block Stacking Assembly. *IEEE Robotics Autom. Lett.* 9, 11 (2024), 9383–9390.

Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. 2015. Legolization: optimizing LEGO designs. *ACM Trans. Graph.* 34, 6 (2015), 222:1–222:12.

Lin Ma, Jiangtao Gong, Hao Xu, Hao Chen, Hao Zhao, Wenbing Huang, and Guyue Zhou. 2023. Planning Assembly Sequence with Graph Transformer. In *IEEE International Conference on Robotics and Automation, ICRA 2023.* 12395–12401.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. 2020. PolyGen: An Autoregressive Generative Model of 3D Meshes. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, Vol. 119. 7220–7229.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. 2024.

DINOv2: Learning Robust Visual Features without Supervision. *Trans. Mach. Learn. Res.* 2024 (2024).

Pavel Petrovic. 2001. Solving lego brick layout problem using evolutionary algorithms. In *Proceedings to Norwegian Conference on Computer Science*.

Ava Pun, Kangle Deng, Ruixuan Liu, Deva Ramanan, Changliu Liu, and Jun-Yan Zhu. 2025. Generating Physically Stable and Buildable Brick Structures from Text. In *ICCV*.

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. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021 (Proceedings of Machine Learning Research, Vol. 139)*. 8748–8763.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*.

Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. 2024. MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024*. 19615–19625.

Ben Stephenson. 2016. A Multi-Phase Search Approach to the LEGO Construction Problem. In *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016*. 89–97.

Jiaxiang Tang, Zhaoshuo Li, Zekun Hao, Xian Liu, Gang Zeng, Ming-Yu Liu, and Qinsheng Zhang. 2025. EdgeRunner: Auto-regressive Auto-encoder for Artistic Mesh Generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*.

Romain Pierre Testuz, Yuliy Schwartzburg, and Mark Pauly. 2013. Automatic generation of constructable brick sculptures. *Eurographics 2013-Short Papers* (2013), 81–84.

Rylee Thompson, Elahe Ghalebi, Terrance DeVries, and Graham W Taylor. 2020. Building lego using deep generative models of graphs. *arXiv preprint arXiv:2012.11543* (2020).

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023).

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).

Haohan Weng, Yikai Wang, Tong Zhang, C. L. Philip Chen, and Jun Zhu. 2024a. PivotMesh: Generic 3D Mesh Generation via Pivot Vertices Guidance. *arXiv preprint arXiv:2405.16890* (2024).

Haohan Weng, Zibo Zhao, Biwen Lei, Xianghui Yang, Jian Liu, Zeqiang Lai, Zhuo Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, et al. 2024b. Scaling mesh generation via compressive tokenization. *arXiv preprint arXiv:2411.07025* (2024).

David V. Winkler. 2005. Automated Brick Layout. In *Proc. BrickFest*. 145–166.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online, 38–45.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. 2024. Structured 3D Latents for Scalable and Versatile 3D Generation. *arXiv preprint arXiv:2412.01506* (2024).

Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. 2019. Computational LEGO technic design. *ACM Trans. Graph.* 38, 6 (2019), 196.

Chongjie Ye, Yushuang Wu, Ziteng Lu, Jiahao Chang, Xiaoyang Guo, Jiaqing Zhou, Hao Zhao, and Xiaoguang Han. 2025. Hi3DGen: High-fidelity 3D Geometry Generation from Images via Normal Bridging. *arXiv preprint arXiv:2503.22236* (2025).

Ruowen Zhao, Junliang Ye, Zhengyi Wang, Guangce Liu, Yiwen Chen, Yikai Wang, and Jun Zhu. 2025b. DeepMesh: Auto-Regressive Artist-mesh Creation with Reinforcement Learning. *arXiv preprint arXiv:2503.15265* (2025).

Zibo Zhao, Zeqiang Lai, Qingxiang Lin, Yunfei Zhao, Haolin Liu, Shuhui Yang, Yifei Feng, Mingxin Yang, Sheng Zhang, Xianghui Yang, Huiwen Shi, Sicong Liu, Junta Wu, Yihang Lian, Fan Yang, Ruining Tang, Zebin He, Xinzhou Wang, Jian Liu, Xuhui Zuo, Zhuo Chen, Biwen Lei, Haohan Weng, Jing Xu, Yiling Zhu, Xinhai Liu, Lixin Xu, Changrong Hu, Tianyu Huang, Lifu Wang, Jihong Zhang, Meng Chen, Liang Dong, Yiwen Jia, Yulin Cai, Jiaao Yu, Yixuan Tang, Hao Zhang, Zheng Ye, Peng He, Runzhou Wu, Chao Zhang, Yonghao Tan, Jie Xiao, Yangyu Tao, Jianchen Zhu, Jinbao Xue, Kai Liu, Chongqing Zhao, Xinming Wu, Zhichao Hu, Lei Qin, Jianbing Peng, Zhan Li, Minghui Chen, Xipeng Zhang, Lin Niu, Paige Wang, Yingkai Wang, Haozhao Kuang, Zhongyi Fan, Xu Zheng, Weihao Zhuang, YingPing He, Tian Liu, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, Jingwei Huang, and Chunchao Guo. 2025a. Hunyuan3D 2.0: Scaling Diffusion Models for High Resolution Textured 3D Assets Generation. *arXiv preprint arXiv:2501.12202* (2025).

Jie Zhou, Xuejin Chen, and Ying-Qing Xu. 2019. Automatic Generation of Vivid LEGO Architectural Sculptures. *Comput. Graph. Forum* 38, 6 (2019), 31–42.

Mingjun Zhou, Jiahao Ge, Hao Xu, and Chi-Wing Fu. 2023. Computational Design of LEGO® Sketch Art. *ACM Trans. Graph.* 42, 6 (2023), 201:1–201:15.