LEGO®-Maker: Autoregressive Image-Conditioned LEGO® Model Creation

JIAHAO GE*, MINGJUN ZHOU*, and HANYOU ZHENG, The Chinese University of Hong Kong, Hong Kong, China
HAO XU†, Unicus® Research, Hong Kong, China
CHI-WING FU†, The Chinese University of Hong Kong, Hong Kong, China

Fig. 1. A street view of 3D LEGO® facades generated by LEGO®-Maker. LEGO®-Maker is able to generate LEGO® models from photos (see Figures 9 and 11), considering over *100 unique brick types*, rapidly generating bricks at a speed of 300 bricks per minute, and producing large LEGO® models with several hundreds of bricks, beyond the capabilities of all existing generative approaches. Note that the lamp posts, chairs, bins, and plants are manually added as decorations.

This paper presents LEGO®-Maker, a new learning-based generative model that can effectively consider over 100 unique brick types and rapidly generate hundreds of bricks to create LEGO® models conditioned on images. This work has three major technical contributions that enable it to achieve surpassing capabilities beyond existing generative approaches. First, we design a compact LEGO® tokenization scheme to serialize LEGO® models and bricks into tokens for autoregressive learning. Second, we build LEGO®-Maker, an autoregressive image-conditioned architecture, with a multi-token prediction strategy to encourage pre-considering multiple brick attributes and a rollback mechanism for collision-free generation. Third, we propose an effective data preparation pipeline with a procedural generator to synthesize LEGO® models and a LEGO®-to-real image translator distilled from a large vision language model to translate LEGO® renderings into associated photorealistic images, leveraging rich prior to address the scarcity of image-to-LEGO® data. Extensive evaluations and comparisons are conducted on two object categories, facade and portrait, over metrics in four aspects: geometry, color, semantics, and structural integrity, together with a user study. Experimental results demonstrate the versatility and compelling strengths of LEGO®-Maker in producing structures and details given by the reference image. Also, the evaluation scores manifest that our method clearly surpasses the baselines, consistently for all evaluation metrics.

*indicates joint first authors.
†indicates co-corresponding authors.

Authors' Contact Information: Jiahao Ge, jiahaoge@link.cuhk.edu.hk; Mingjun Zhou, mingjunzhou@link.cuhk.edu.hk; Hanyou Zheng, hyzheng@link.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, China; Hao Xu, hao.xu@maic.fun, Unicus® Research, Hong Kong, China; Chi-Wing Fu, cwfu@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, China.

ACM 1557-7368/2025/12-ART201
https://doi.org/10.1145/3763285

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

Additional Key Words and Phrases: LEGO®, autoregression, machine learning, 3D generation, assembly generation

ACM Reference Format:
Jiahao Ge, Mingjun Zhou, Hanyou Zheng, Hao Xu, and Chi-Wing Fu. 2025. LEGO®-Maker: Autoregressive Image-Conditioned LEGO® Model Creation. *ACM Trans. Graph.* 44, 6, Article 201 (December 2025), 15 pages. https://doi.org/10.1145/3763285

1 Introduction

LEGO® [1] is a versatile and globally-popular construction system, making use of interlocking plastic bricks to assemble 3D objects of almost any kind. We can reproduce objects in the real world, such as buildings, vehicles, animals, flowers, and human characters, as well as objects that are artistic, imaginary, and out-of-the-box.

Building LEGO® models is generally a physical process. Yet, physically building LEGO® models is largely limited by the types and number of bricks that we currently have at hand. Also, finding a specific brick in a large repository can be tedious. To this end, various LEGO® building software tools, such as BrickLink Studio [BrickLink. 2024] and LDraw [Jessiman 1995], have been developed to support virtual LEGO® building. Using these tools, we enjoy not only unlimited use of bricks, with almost any LEGO® color, but also fast access to any specific brick and photorealistic rendering of our results. Besides, one can load a reference image into the 3D virtual space and then virtually arrange LEGO® bricks in the same 3D space to try to match the object in the reference image. This function is called the "reference image" in the BrickLink Studio tool.

[1] LEGO® is a trademark of the LEGO® Group, which does not sponsor, authorize or endorse this work. All information in this paper is collected and interpreted by its authors and does not represent the opinion of the LEGO® Group.

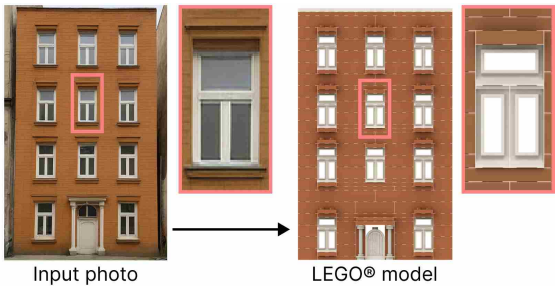


Fig. 2. Our objective is to generate LEGO® models (right) from natural photos (left), reproducing the structure, details, and colors in the inputs.

Earlier in the research community, various methods, *e.g.*, [Gower et al. 1998; Petrovič 2001; Testuz et al. 2013; Winkler 2005] have been developed to leverage computing power to transform 3D shapes into LEGO® brick models. To enhance the results, [Luo et al. 2015] propose to introduce stability analysis and coloring of bricks in the LEGO® model construction process. In these early works, the resulting LEGO® models are basically voxel-based, since only rectangular LEGO® bricks are considered. Therefore, the produced LEGO® models typically lack expressiveness and often fall short of capturing details. More recently, some research works [Zhou et al. 2019] [Xu et al. 2019] [Zhou et al. 2023] start to consider nonrectangular bricks in creating LEGO® models. Yet, they are formulated for specific LEGO® categories. Hence, it is hard to extend them for more general LEGO® construction. Also, they involve combinatorial optimization, which comes at a high cost in computational overhead.

Motivated by recent advances in deep learning, some research works, *e.g.*, [Lennon et al. 2021; Thompson et al. 2020], start to explore neural networks for generating LEGO® models. Pun et al. [2025] further propose a text-to-LEGO® approach to generate LEGO® models from text inputs. Yet, the generated models are voxelized, as only basic rectangular bricks are considered. Another work [Ge et al. 2024a] presents a diffusion model to unconditionally generate simple micro LEGO® buildings, using around 30 brick types to create tiny buildings of assorted styles.

To go beyond the prior works, our goal in this work is to develop a learning-based approach for broader use, capable of considering more diverse brick types and colors, while conditioning the generation with a reference image (see Figure 2), such that we can generate intriguing LEGO® models beyond the existing approaches. Also, we aim at an approach that can be customized to generate LEGO® models beyond a single LEGO® category, while ensuring connectable bricks in the output LEGO® models without bricks collision and floating. Further, we aim to rapidly generate standard-sized LEGO® models (~300 pieces) in a reasonable time frame, say hundreds of bricks in a few minutes. To our best knowledge, no existing research or commercial tools can fully meet these requirements.

Jointly considering all the above requirements in developing a general-purpose image-conditioned LEGO® generator is very challenging. This is mainly due to the scarcity of high-quality LEGO® designs with paired image references. Hence, to make the task more manageable, we consider category-specific generation and focus on two categories: LEGO® facades and portraits. Overall, there are three innovations in this work: First, we design a *LEGO® tokenization scheme* to carefully serialize LEGO® models into tokens, in which we compactly represent bricks, as well as the brick attributes, including brick type, color, orientation, and position. Through this compact representation, we can effectively consider more brick types and colors, as well as enable our approach to generate LEGO® models of larger size and finer details, beyond the existing generative approaches. Second, we formulate the *LEGO®-Maker* model, a new autoregressive pipeline for LEGO® model generation. In the pipeline, we effectively train the model to learn to produce LEGO® brick tokens conditioned by a reference image, while encouraging the model to learn brick-by-brick planning with a multi-token prediction strategy. Third, we design two data-acquisition pipelines to prepare paired image-LEGO® data for model training. The first pipeline includes a highly automated procedural generator for creating facade models and a LEGO®-to-real image translator distilled from GPT-4o to inversely produce associated photorealistic reference images. The second pipeline is a manual designer-driven process for LEGO® portrait models. It is worth noting that the pipelines are general and extensible to handle additional LEGO® categories.

LEGO®-Maker is a versatile approach for LEGO® model generation. In particular, we can condition the generation on a reference image of a facade or human portrait, consider over 100 brick types, and rapidly generate LEGO® models at ~ 300 bricks per minute. Also, we extensively evaluated our approach on metrics in four aspects, *i.e.*, geometry, color, semantics, structural and integrity, together with a user study. The quantitative evaluation demonstrates our method's compelling performance over various baselines, showcasing that it can effectively compose bricks to reproduce prominent structures and details in the reference images. Further, we performed various ablations to show the effectiveness of our designs and conducted floating-brick tests on the generated results to show the structural robustness of the generated LEGO® models.

The technical contributions of this work are summarized below.

(i) We introduce the first approach that can effectively learn to generate large-sized, multi-color LEGO® models from reference images, and demonstrate its versatility and applicability on two object categories, facade and portrait;
(ii) We design the LEGO® tokenization scheme, by which we can compactly encode the brick type, color, orientation, and positions, enabling us to efficiently consider a large number of bricks in the LEGO® model and over 100 types of unique bricks when generating LEGO® models;
(iii) We construct LEGO®-Maker, a new image-conditioned autoregressive architecture to generate LEGO® models, with a multi-token prediction strategy to encourage brick-by-brick planning by pre-considering multiple brick attributes and a rollback mechanism to avoid bricks collision;
(iv) We propose a data preparation pipeline to address data scarcity, with a procedural generator to synthesize LEGO® models and a LEGO®-to-real image translator distilled from GPT-4o to generate associated photorealistic images; and
(v) We extensively evaluate our method against multiple baselines on a rich set of quantitative metrics on geometry, color, semantics, and structure, as well as through a user study.

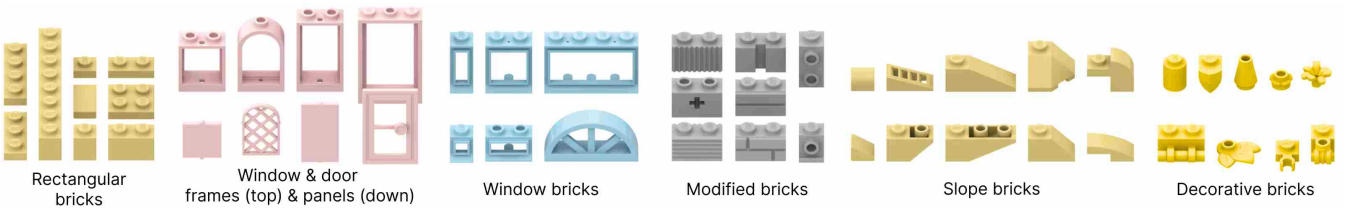


Fig. 3. Example bricks in the LEGO® facade scenario, in which we consider six categories of bricks, from regular rectangular bricks to a wide range of nonrectangular types (left to right), including frames, panels, windows, modified bricks, and slope bricks, as well as bricks for decorative purpose. There are 103 unique brick types in total and we show only a subset of them. Please refer to Supplementary material Part A for the full brick set.

2 Related Work

Image-conditioned 3D Shape Generation. The task of generating LEGO® models from images closely relates to the broader task of generating 3D models from images. Building upon the remarkable progress in 2D image generation, *e.g.*, DALL·E [2021], Imagen [2022], and Stable Diffusion [2022], many methods explore these capabilities in 3D shape generation. DreamFusion [2023] optimizes NeRF [2020] with Score Distillation Sampling (SDS) for 3D generation. Subsequent works [Li et al. 2024a; Seo et al. 2024; Tang et al. 2024; Zhu et al. 2024] explore SDS in various neural field representations. Some other works take a multi-view approach to 3D generation. To improve the view consistency, Zero-1-to-3 [2023] conditions pre-trained diffusion models on explicit camera transformations, whereas [Li et al. 2024b; Qiu et al. 2024; Shi et al. 2023, 2024] incorporate enhanced attention mechanisms. Later, Misra et al. [2025] extend SDS to multi-shape arrangements, arranging rigid shapes in a non-overlapping manner to match a semantic description.

In parallel, large 3D datasets such as Objaverse [Deitke et al. 2023a,b] greatly accelerate the development of data-driven 3D generation. Leveraging these resources, reconstruction models [Hong et al. 2024; Xu et al. 2024; Zhang et al. 2024a], diffusion models [Hui et al. 2024; Lan et al. 2025; Wu et al. 2024; Xiang et al. 2025; Zhang et al. 2024b], and autoregressive models [Chen et al. 2025b] show impressive results in producing high-quality, diverse, and structurally rich 3D assets from images via different latent representations.

Clearly, existing 3D representations and generative paradigms cannot be applied to directly generate LEGO® models. More than conventional surface-based 3D representations, LEGO® models are 3D assemblies of discrete bricks that must be arranged in an interconnected, collision-free manner. Also, the bricks should come from a pre-defined set, with connections governed by rules. Hence, we formulate LEGO® model generation as a step-by-step brick assembly process, analogous to autoregressive generation.

Autoregressive shape generation. Another related topic is autoregressive shape generation, which aims to produce vertices, edges, and faces that make up global shapes with fine geometric details. As a pioneering effort, MeshGPT [Siddiqui et al. 2024] generates triangular meshes by modeling them as sequences of triangles. MeshAnything [Chen et al. 2025a] proposes to learn a discrete vocabulary of mesh tokens using a VQ-VAE then use a shape-conditioned, decoder-only transformer to generate meshes. To improve efficiency and scalability in the generation, EdgeRunner [Tang et al. 2025] employs the mesh compression algorithm EdgeBreaker [Rossignac 2002]; MeshAnything V2 [Chen et al. 2025c] proposes Adjacent Mesh Tokenization; and OctGPT [Wei et al. 2025] presents a serialized octree representation. Leveraging an autoregressive model for meshes, PrimitiveAnything [Ye et al. 2025] propose to learn from and reproduce human-crafted shape decompositions.

In parallel, some recent works study the combination of autoregressive models with procedural content generation (PCG). Latent L-systems [Lee et al. 2023] learn to generate procedural string sequences that define tree structures. FaçAID [Plocharski et al. 2024] transforms static facades into editable procedural grammars. BuildingBlock [Huang et al. 2025] integrates generative models, PCG, and large language models to produce hierarchically consistent 3D building layouts. Overall, this stream of works has great potential for creating complex structures that remain highly editable.

Inspired by these successes, we propose to learn the step-by-step LEGO® brick construction process by modeling an autoregressive framework to generate different categories of LEGO® models.

Computational LEGO® model construction. In the last few decades, various algorithms [Gower et al. 1998; Lee et al. 2018; Luo et al. 2015; Peysakhov et al. 2000; Smal 2008] are proposed for creating LEGO® models. Yet, the earlier works mainly consider rectangular bricks and focus on stability and bricks connectivity. To consider more brick types beyond regular rectangular bricks, Xu et al. [2019] designed models with LEGO® Technic bricks; Zhou et al. [2019] designed houses with sloping and circular bricks; and Zhou et al. [2023] designed LEGO® sketches with LEGO® tiles and plates.

Entering the era of AI, some works, *e.g.*, Thompson et al. [2020] and Lennon et al. [2021], start to exploit learning-based approaches for LEGO® model generation. Yet, all these works produce only voxelized single-color results, without considering nonrectangular bricks and different available brick colors. To call for quality, Ge et al. [2024b] create LEGO® figurine models from human portraits. Yet, the bricks are arranged by retrieval, not generation. Later, Ge et al. [2024a] design a volumetric diffusion approach to generate simple micro LEGO® buildings using 28 different brick types. However, the generation is unconditional and the method can handle only single-color small-sized LEGO® models.

A very recent concurrent work is Pun et al. [2025], which also studies autoregressive models for generating LEGO® models. Compared with our work, this work considers LEGO® assemblies composed of conventional cuboid bricks and focuses on creating physically stable assemblies. Also, it is text-conditioned, without image conditioning and without considering brick colors in the generative

process. Besides, while the approach is effective within its scope, it does not address the creation of larger-scale models.

Our work presents the first approach that can generate larger-scale, multi-color LEGO® models with image conditioning and with brick details. Specifically, we take natural photos as inputs and consider more than **100** types of bricks and **30** different brick colors to generate LEGO® models as large as **35.2**cm in size, significantly beyond the scale and capabilities of all the prior methods.

3 Overview

To put our research objective in mathematical form, we denote $\mathcal{I}$ as the input natural image and train an autoregressive model $\mathcal{F}$ to generate LEGO® model $\mathcal{M}$ that resembles the object given in image $\mathcal{I}$; see Figure 2 for an example. Denoting N as the number of bricks in LEGO® model $\mathcal{M}$ and b_i as the i-th brick in $\mathcal{M}$, $\mathcal{M}$ can then be represented as $\{b_i \mid i \in \{1, 2, ..., N\}\}$. In the autoregressive process, bricks are generated sequentially one at a time:

$$b_i = \mathcal{F}(\mathcal{I}, \{b_{<i}\}).$$

where each brick b_i in the completed LEGO® model consists of four attributes—type, color, orientation, and position.

Requirements on results. We consider the following requirements in the LEGO® model generation. Specifically, the results should

(i) capture the overall structure and local details of the target object in the input image as faithfully as possible;
(ii) be able to use a wide range of bricks to form shapes with intricate details (see some of our employed bricks in Figure 3);
(iii) have regular physical sizes similar to generic LEGO® models, surpassing the scale constraints in the previous works (6.4 cm in [Ge et al. 2024a] and 16 cm in [Pun et al. 2025]);
(iv) preserve the colors in the input image with best effort (this is not supported in previous learning-based methods);
(v) be physically buildable, where the bricks in the final assembly should be collision-free and connectable; and
(vi) be able to generate LEGO® designs within a reasonable time frame; see the performance of our method in Section 7.6.

Fulfilling the above requirements is very challenging. First, there is a substantial domain gap between natural images and LEGO® models. The generative method needs to bridge the gap. Second, the method should be general and flexible, capable of generating LEGO® models of not so small scale, while capable of deploying a wide variety of brick types and colors. Third, the scarcity of paired data between images and LEGO® models makes it difficult to train the generative method, especially for arbitrary generic LEGO® designs.

Aiming to consider a large variety of brick types and colors, rather than being too ambiguous, in this work, we focus on two LEGO® model categories, which are less complex and more manageable:

(i) *LEGO® facade*: Brick number ranges from 100 to 675 (average 300); unique brick type number 103; maximum dimension 35.2 cm; average generation time 55 seconds per model; and
(ii) *LEGO® portrait*: Brick number ranges from 40 to 63; unique brick type number 75; maximum dimension 16.96 cm; average generation time 6 seconds per model.

Excitingly, no previous LEGO® generative methods have ever succeeded in generating LEGO® models of such complexity, in terms of brick types and brick numbers. Yet, our method is able to efficiently generate good-quality LEGO® models for both categories, beyond the capabilities of all existing generative methods.

Our approach. There are three major components in our approach: (i) A LEGO® tokenization scheme to serialize a LEGO® model into bricks and brick attributes for generative learning (Section 4); (ii) LEGO®-Maker, an image-conditioned autoregressive model trained to generate brick tokens for constructing LEGO® models (Section 5); and (iii) We prepare data for model training, especially with our procedural generator and LEGO®-to-real image translator to produce synthetic paired data to address data scarcity (Section 6).

In the following sections, we present our approach primarily using the LEGO® facade category in our examples, while experiments presented in Section 7 will include both facades and portraits.

4 LEGO® Tokenization

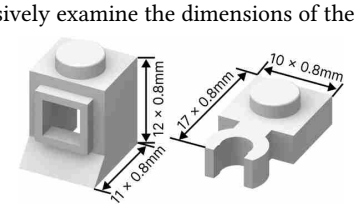
The core to enable autoregressive LEGO® learning is a compact LEGO® tokenization scheme for representing bricks and the associated assembled model. In our approach, instead of treating each LEGO® brick as a single token, we go deeper to consider four basic attributes of bricks in the assembled model, *i.e.*, type, color, orientation, and position. Also, we consider bricks in a bottom-up manner, following a natural LEGO® building order.

Specifically, we represent brick b_i as a tuple of six tokens, *i.e.*, $(t_i, c_i, o_i, x_i, y_i, z_i)$, which denotes b_i's type, color, orientation, and x/y/z position in the assembled model, respectively. To aim for compactness in tokenization, all token values are integer-valued indices in our LEGO® vocabulary $\mathcal{V}$. Below, we shall first define our LEGO® vocabulary, then present the LEGO® tokenization process.

Building LEGO® vocabulary. The LEGO® vocabulary $\mathcal{V}$ is a list of all possible values of each brick attribute. Essentially, $\mathcal{V}$ is a concatenation of all brick types $\mathcal{V}_{type}$, brick colors $\mathcal{V}_{color}$, brick orientations $\mathcal{V}_{orient}$, and brick positions $\mathcal{V}_{pos}$. Considering the LEGO® facade category, we construct brick types $\mathcal{V}_{type}$ and brick colors $\mathcal{V}_{color}$ by recruiting 20 professional designers and getting their help to decide the set of brick types (103 for facade) and brick colors (37 standard LEGO® colors) that best match the facades in a real photo collection. Here, the majority of colors in $\mathcal{V}_{color}$ are solid colors, except for a few transparent colors for windows.

For brick orientations $\mathcal{V}_{orient}$, observing that bricks are connected mostly by the studs and tubes on top/bottom/side faces, for simplicity, we consider only the 24 unique axis-aligned orientations in 3D. Extending to arbitrary orientations will be our future work.

For brick positions $\mathcal{V}_{pos}$, we define brick position (x, y, z) as a displacement vector from the LEGO® model's origin, *i.e.*, the minimum x/y/z coordinates of the model's bounding box. Importantly, to avoid floating point values in the tokens, which are tedious to compute and hard to predict, we extensively examine the dimensions of the employed brick types and connections, and find that the greatest common factor of the dimensions is 0.8 mm, considering also the irregularly-shaped bricks such as those



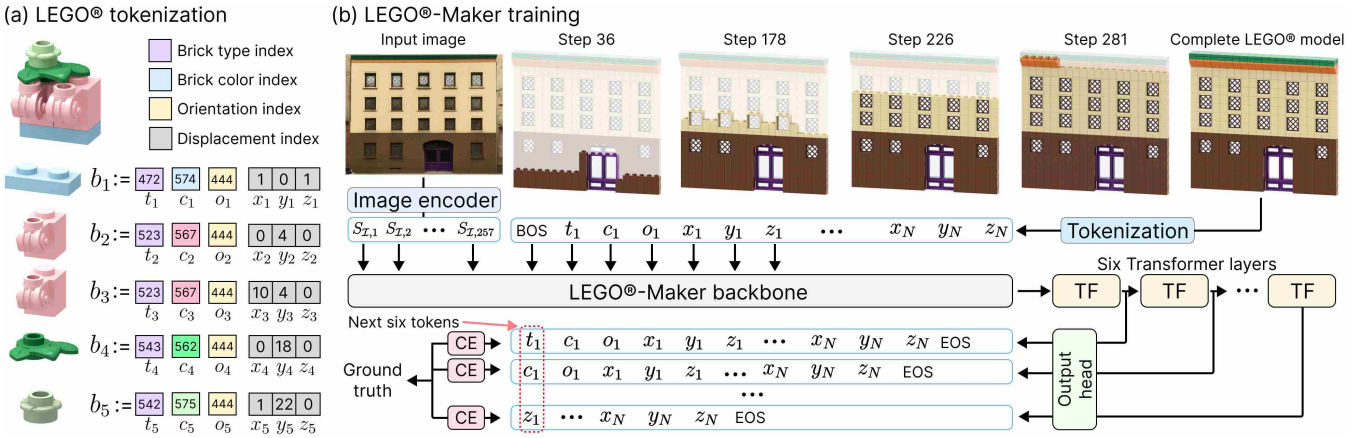


Fig. 4. (a) LEGO® tokenization serializes a LEGO® model brick-by-brick in a bottom-up manner into a sequence of tokens (type, color, orientation, and position), which are integer indices in our LEGO® vocabulary. (b) LEGO® generation: taking the image embedding from input image and brick token sequence from associated LEGO® model, we train LEGO®-Maker (backbone, six transformer (TF) layers, and output head) to predict tokens autoregressively. Also, we adopt a multi-token prediction strategy to compute losses for predicting six consecutive future tokens altogether to aim for brick-by-brick planning by pre-considering more brick attributes in the generation. Note that BOS means beginning of sentence; EOS means end of sentence; and CE means cross entropy.

shown in the inset. Each position token can be expressed as an integer value, which multiples 0.8 mm to obtain the x/y/z displacement from origin. Empirically, the position tokens range from 0 to 439, as the largest dimension observed in the LEGO® facade data samples (see Section 6) is 35.2 cm, which is 440 × 0.8 mm. Also, we need three special separator tokens $\mathcal{V}_{sep} = \{BOS, EOS, PAD\}$ to denote the beginning of sentence, end of sentence, and padding, respectively. Hence, the overall LEGO® vocabulary $\mathcal{V} = \mathcal{V}_{type} \cup \mathcal{V}_{color} \cup \mathcal{V}_{orient} \cup \mathcal{V}_{pos} \cup \mathcal{V}_{sep}$, *i.e.*, 103 + 37 + 24 + 440 + 3 = 607 for the LEGO® facade category. For the LEGO® portrait category, the size of the LEGO® vocabulary is 503.

LEGO® tokenization. With the LEGO® tokenization scheme, we can then tokenize a given LEGO® model in the following two steps. In the first step, we need to sort all the bricks in the model in a bottom-up manner, along the z-axis from the xy ground plane. To do so, we sort the bricks first by their minimum z values, then by their minimum y values, followed by their minimum x values.

In the second step, we iterate through the sorted bricks, from b_1 to b_N. For each brick b_i, we set its type token t_i, color token c_i, and orientation token o_i as the indices to values in $\mathcal{V}_{type}$, $\mathcal{V}_{color}$, and $\mathcal{V}_{orient}$, respectively. Next, we set the brick's position tokens x_i, y_i, and z_i based on the brick's displacements from the model's origin. Also, we include token *BOS* at the beginning and token *EOS* at the end of the token sequence; see Figure 4 for an example. Note that when we train a batch of sequences of different lengths, we additionally append *PAD* after the *EOS*, such that all sequences in the same batch will have the same length. Formally, given LEGO® model $\mathcal{M}$, the final token sequence $S_{\mathcal{M}}$ is given as

$$S_{\mathcal{M}} = \text{Tokenizer}(\mathcal{M}),$$

where $S_{\mathcal{M}}$ is a sequence that has $6N + 2$ tokens altogether.

5 LEGO®-Maker architecture

Our pipeline to train the LEGO®-Maker architecture has three major steps, as illustrated in Figure 4(b): (i) encoding the input image; (ii) tokenizing the associated LEGO® model (Section 4); and (iii) training LEGO®-Maker (including the LEGO®-Maker backbone, six transformer layers, and a shared output head) to autoregressively predict respective tokens. The details of how the data samples are prepared will be presented in Section 6. In this section, we shall first elaborate on the details of encoding the input image and training LEGO®-Maker, then provide additional details in our approach.

Image encoding. We extract features from input image $\mathcal{I}$ using the DINOv2 encoder [Oquab et al. 2024]. In detail, the encoder output is an image embedding $S_{\mathcal{I}}$ of length 257. As shown in Figure 4(b), we incorporate the image embedding into the token sequence by prepending $S_{\mathcal{I}}$ before the tokenized LEGO® model embedding $S_{\mathcal{M}}$, resulting in a combined sequence $S_{\mathcal{I}} \oplus S_{\mathcal{M}}$.

LEGO®-Maker training. The training of LEGO®-Maker is performed token by token. Specifically, at each decoding step t, we take the partial sequence $S_{\mathcal{I}} \oplus S_{\mathcal{M}<t}$ as input to produce a probability distribution over the LEGO® vocabulary for predicting the token at step t. During the training, we supervise LEGO®-Maker with the ground-truth token sequence obtained from the associated LEGO® model, where the objective is to minimize the negative log-likelihood of the correct tokens across the entire sequence.

Formally, given the token sequence $(S_{\mathcal{M},1}, S_{\mathcal{M},2}, \ldots, S_{\mathcal{M},T})$, where step $t \in \{1, 2, ..., T\}$, the training loss is defined as

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(S_{\mathcal{M},t} \mid S_{\mathcal{I}}, S_{\mathcal{M}<t}), \tag{1}$$

where $S_{\mathcal{M}<t}$ represents all the tokens before step t. During the inference, we can employ the trained LEGO®-Maker model to generate

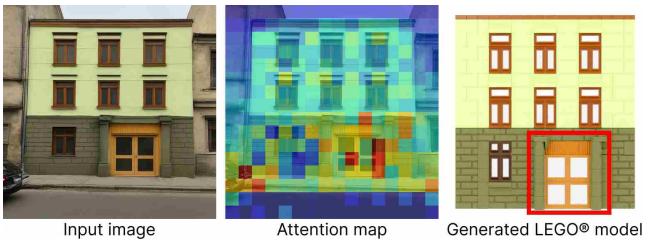


Fig. 5. Visualizing the attention map (last layer of the backbone) in our LEGO®-Maker model when generating the LEGO® bricks in the facade entrance marked by the red box. From the attention map, we can see that the model attends mainly to the ground level and entrance regions in the input image when generating the LEGO® bricks in the facade entrance.

tokens autoregressively from an image embedding, starting from the *BOS* token until arriving at the *EOS* token.

It is worth noting that at each token decoding step, the model attends to the entire image sequence S_I, allowing it to adaptively align the generated brick with the visual structure. This attention-guided conditioning helps the model preserve key geometric and semantic features, such as facade layout and colors of specific elements, as illustrated in Figure 5. Also, at the beginning of this research, we have considered segmenting the input image into semantic regions, such as windows, wall, and doors, and then using the segmentation to spatially guide the LEGO® generation. However, since the generated LEGO® models need not strictly follow the semantic contents in the 2D image space, we thereby do not incorporate strong semantic guidance in the image space in our pipeline design.

LEGO®-Maker backbone. The backbone is from the Open Pre-trained Transformer (OPT) 1.3B model [Zhang et al. 2022]. It comprises 24 layers of standard OPT decoder with a hidden size of 2,048. As the original model is pre-trained on natural languages, we train the model from scratch instead of using the pre-trained weights.

Multi-token prediction. In standard autoregressive modeling, the decoder predicts a single token in each forward step. To improve the coherence in generating LEGO® bricks, we adopt a multi-token prediction strategy [Gloeckle et al. 2024; Liu et al. 2024a] to predict the next six tokens altogether within a single forward pass. To do so, we deploy six dedicated transformer layers stacked on top of the backbone decoder (Figure 4(b)). During the training, the hidden states produced by the LEGO®-Maker backbone are successively consumed by these additional layers, each responsible for predicting one of the next six tokens. By doing so, the LEGO®-Maker model can better capture local brick dependencies more effectively.

Formally, we denote $\mathcal{F}_{\text{backbone}}$ as the backbone transformer, $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_5$ as the six sequential transformer layers, and $\mathcal{H}$ as the shared output head. At the decoding step t,

$$S_{\mathcal{M},t+i} = \mathcal{H}\Big((\mathcal{F}_i \circ \mathcal{F}_{i-1} \circ \ldots \circ \mathcal{F}_0 \circ \mathcal{F}_{\text{backbone}})(S_{\mathcal{M}<t})\Big),\ i = 0, \ldots, 5.$$

Accordingly, the loss term presented in Eq. (1) shall be updated to account for the joint prediction of the six tokens:

$$\mathcal{L} = -\frac{1}{6}\sum_{t=1}^{T}\sum_{i=0}^{5} \log P(S_{\mathcal{M},t+i} \mid S_I, S_{\mathcal{M}<t}). \tag{2}$$

By incorporating the next six tokens into the training supervision, the LEGO®-Maker model can learn to jointly pre-consider multiple brick attributes when predicting the next tokens. Note that we adopt MTP only in training. At inference, we still generate LEGO® models token-by-token to maximize the model's pre-consideration ability. After a sequence of tokens is generated, we use them to index the next brick attribute in our LEGO® vocabulary and employ the combined attributes to reconstruct to a complete LEGO® model.

Rollback mechanism. We observe that the LEGO®-Maker model exhibits improved learning of assembly patterns, as the training dataset scales up. However, there are no strict collision constraints when predicting tokens. Hence, to avoid potential collisions, we introduce a rollback mechanism. After predicting each brick, we perform a collision detection between the brick and all the previous bricks. If a collision occurs, we mask the corresponding token options and regenerate the brick. This adaptive mechanism is rarely needed in practice. Experimentally, our model can generate sequences of hundreds to thousands of bricks without any collision. So, only around 28% of the generated LEGO® models trigger this mechanism in the generation. Also, all the generated models can become collision-free, thanks to this mechanism. See Section 7 for details.

6 Dataset Preparation

This section presents our pipelines for preparing training data for LEGO®-Maker. Specifically, we focus on two categories in this work: facades and portraits. For the facade category, we employ a highly automated pipeline, creating a procedural generator and a LEGO®-to-real image translator to inversely prepare 40,000 paired LEGO® facade models with natural images. For the portrait category, we utilized a designer-driven pipeline to create 3,300 data pairs.

Discussion on category-specific datasets. The reason we opt for category-specific datasets is due to large variation in the employed LEGO® bricks and also assembly patterns utilized in different LEGO® categories. For instance, LEGO® mosaics employ mainly flat plates connected by studs, whereas LEGO® buildings make use of windows, doors, and various decorative bricks. Also, the scarcity of paired image-LEGO® models renders this task too challenging in training a single model for diverse categories.

6.1 LEGO® Facade Dataset

For facades, we first create an initial dataset of 2,000 LEGO® facade-image pairs, which are meticulously created by professional designers; see Figure 6(a). Then, we aim to expand this dataset by automatically generating more paired samples; see Figure 6(b).

Inversely to the main task, *i.e.*, image-to-LEGO®, our key idea here is to develop a procedural generator to first automatically generate a large volume of LEGO® facade models, then train a LEGO®-to-real image translator to inversely produce photorealistic images of the generated LEGO® models. By then, we can obtain not only a reference image for each LEGO® model but also 3D LEGO® models composed of connecting bricks for training LEGO®-Maker.

LEGO® facade procedural generator. To start, we identify bricks in LEGO® facades and basic architectural components (windows,

Fig. 6. Expansion of the initial designer-crafted dataset (a), which contains 2,000 pairs of reference images (green boxes) and facade models (blue boxes), to a larger dataset of 38,000 pairs (b). The augmented dataset maintains a consistent style with the original dataset. Facade images in (a) are created by GPT-4o.

Fig. 7. Comparing LEGO®-to-real image translation results produced by different methods. GPT-4o [OpenAI 2024] produces realistic textures but falls short of maintaining coherent facade layouts. SD 3.5 [Esser et al. 2024], paired with ControlNet [Zhang et al. 2023], maintains general structures, but struggles with color fidelity and photorealism. Our LEGO®-to-real image translator can better preserve both the layout structure and appearance.

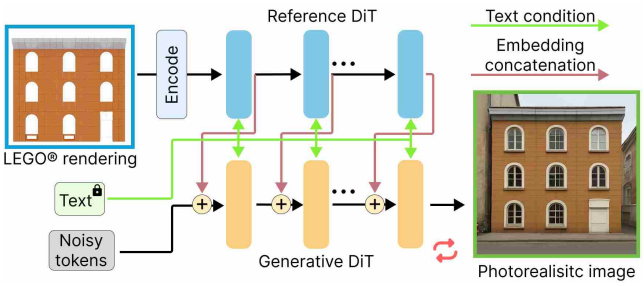


Fig. 8. Our LEGO®-to-real image translator architecture. The Reference DiT blocks extract image features from the LEGO® rendering, whereas the Generative DiT blocks iteratively denoise the noisy tokens to produce a photorealistic image. Guided by the features from the Reference DiT and embedded text condition, the generation process can better align the generated image with the layout structure in the LEGO® rendering.

doors, walls, etc.) in the initial dataset. Then, we design the generator with various parameters on the facade layout, *e.g.*, floor levels from ground to roof, window and door locations, as well as component type, wall texture, color, etc. After that, we can arrange LEGO® bricks accordingly to generate LEGO® facade models. Please see Supplementary Material Part C for implementation details.

LEGO®-to-real image translator. To ensure good coherence between the paired LEGO® models and reference images in the dataset, we exploited various image generation tools, *i.e.*, GPT-4o [OpenAI 2024], FLUX [Labs 2024], and Stable Diffusion 3.5 [Esser et al. 2024] with ControlNet [Zhang et al. 2023]. As Figure 7 shows, although the images generated by the existing tools appear mostly natural, they are not well aligned with the LEGO® facade models, especially in the layout arrangement of windows and doors.

Inspired by the dual U-Net architecture in recent style transfer works [Ge et al. 2024b; Hu 2024], we adopt a dual-DiT diffusion architecture tailored for our image translation. As Figure 8 illustrates, our architecture comprises two DiT modules with identical architecture, both initialized from the Stable Diffusion (SD) 3.5 backbone to leverage its image prior. In detail, the reference DiT extracts structural features from the rendering of the LEGO® model, whereas the generative DiT utilizes these features to guide the image denoising and photorealistic synthesis. Given a LEGO® model rendering, we first extract image features using a convolutional layer. Concurrently, we initialize a noise sequence of the same dimension. In each denoising step, the reference DiT branch processes the image features via a sequence of DiT blocks to produce reference features. Next, at each generative DiT block, we concatenate the noisy tokens with the reference feature produced by the same-layer reference DiT block, and then send this concatenated feature to the Generative DiT block. To ensure compatibility with the pre-trained SD 3.5 backbone, we retain its text condition by fixing the text prompt as "A photorealistic image of the front view of a facade."

In addition, to improve the robustness of LEGO®-to-real, we further leverage the powerful cross-modality capability of GPT-4o [OpenAI 2024]. Specifically, we create 1,000 LEGO® facade models using our procedural generator and translate their renderings to photorealistic images using GPT-4o. Then, we identify 200 good-quality

Fig. 9. A gallery of LEGO® facade models generated by our method, alongside with the reference image on the left of each generated LEGO® model.

LEGO®-image pairs and include them to train the LEGO®-to-real image translator alongside with the designer-crafted dataset.

Dataset expansion. Overall, we expand the designer-crafted dataset by first employing the procedural generator to produce 38,000 LEGO® facade models by randomly varying the parameters in the procedural generator. Then, we employ the LEGO®-to-real image translator to create a photorealistic image for each generated LEGO® model. See Figure 6 for example data samples.

6.2 LEGO® portrait dataset

To demonstrate the versatility of LEGO®-Maker, we further create a dataset for the portrait category. To do so, we recruited ten professional designers to create 3,300 portrait models, each associated with a reference human photo. In short, the design process starts with a template model with fixed face and body styles. The designers can then creatively make modifications on the hair, and incorporate additional objects such as hat, adjust the garment colors, etc., to better align with the reference photo.

7 Results and experiments

7.1 Implementation Details

We implemented LEGO®-Maker using PyTorch and the Transformers library. For the facade dataset, we randomly split it into 38,000/2,000 samples for training/testing and trained LEGO®-Maker on eight NVIDIA RTX 4090 GPUs using the Adam optimizer with a learning rate of 1×10^{-4} and a batch size of 64, for 60 epochs. The entire training was completed in around two days. For the portrait dataset, we randomly split it into 3,000/300 samples for training/evaluation and trained LEGO®-Maker on a single RTX 4090 GPU using the same optimizer and learning rate, with a batch size of 12, for 150 epochs. The training was completed in about three hours. For the LEGO®-to-real image translator, we implemented it using the Diffusers library and trained it on eight NVIDIA A100 GPUs using the Adam optimizer, with a learning rate of 1×10^{-5} and a batch size of 32, for 100 epochs. The training took around three hours to complete.

7.2 LEGO® generation Results

Figures 9 and 11 present 15 LEGO® facade models and five LEGO® portrait models created by our LEGO®-Maker from the reference images in the testing sets. These results demonstrate that our method is able to effectively generate LEGO® models that reproduce rich variety of features, intricate details, and component colors given in the reference images. Also, each generated model comprises LEGO® bricks of diverse shapes and colors, and these bricks are arranged to connect coherently in varied configurations.

Gallery of LEGO® facades. From the generated LEGO® facades shown in Figure 9, we can see that the essential architectural components, *e.g.*, windows and doors, maintain accurate counts and positions. By effectively leveraging the limited LEGO® brick color palette, our approach can roughly reproduce the original colors seen in the input images, including good distinctions between the ground and upper floors, window frames, wall surfaces, and decorative details. Additionally, the generated facades reflect authentic architectural symmetry, both vertically and horizontally.

Moreover, our method demonstrates high diversity in architectural styles. Beyond basic rectangular windows and doors (see the three examples shown in the 3rd column of Figure 9), the generated facades showcase various design elements such as balconies

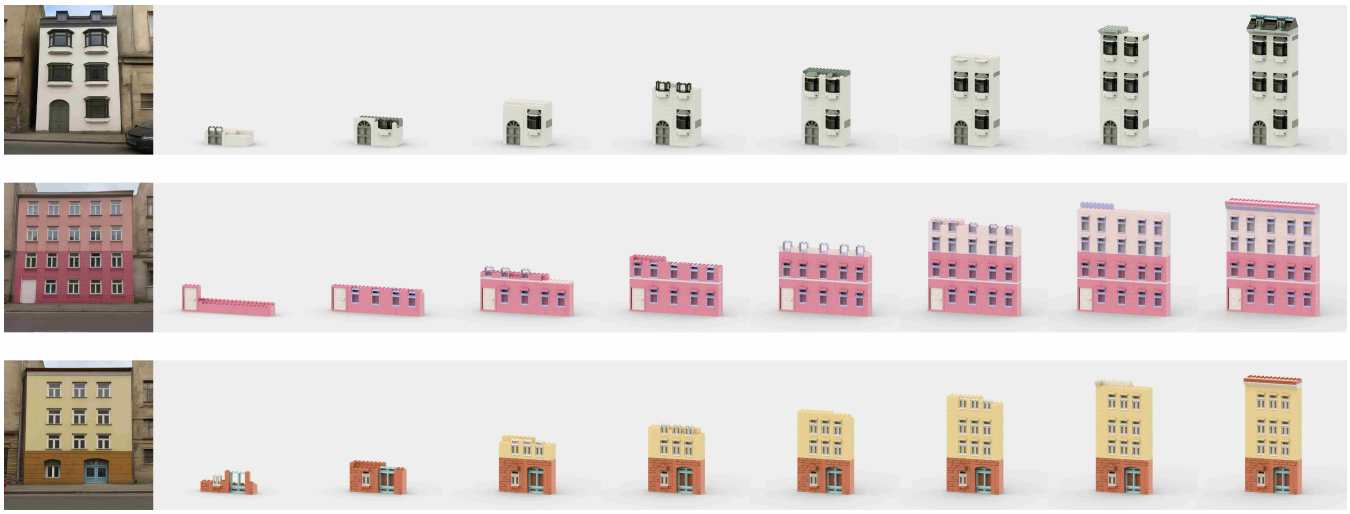


Fig. 10. Screenshots of assembling three example LEGO® facade models generated by LEGO®-Maker.

(row 1 column 1), windows with hoods (row 1 column 4), cross-bar panes (row 3 column 1,4), and more intricate window designs (row 1 column 1,2,5; row 2 column 4,5; row 3 column 5). Entrance doors flanked by decorative columns (row 3 column 5) further enrich the variety. Roof styles range from sloped to flat and protruding, with additional details such as chimneys, attics, and cornices.

Figure 10 shows screenshots of assembling facade models generated by LEGO®-Maker, similar to instruction manuals. The generated facades are not only visually plausible but also structurally feasible, in which new bricks are placed to connect with the previous bricks, while avoiding brick collisions and floating. Readers are referred to the supplementary video for the associated animations.

Gallery of portrait models. Figure 11 presents five LEGO® portraits generated from human photos. Besides LEGO® models, we also generate decals on the body part of the LEGO® model, by following the generative method in [Ge et al. 2024b] to extract decals from the input image. Since the training data is based on a common template, all portraits share the same face and body parts, with hairstyle as the primary variation. Our method effectively generates different hair styles and colors, handling photos taken from varied angles and lighting conditions. Also, it is able to capture prominent hair features—such as long, nape-length, or neckline styles—by skillfully arranging bricks to replicate different hair styles.

7.3 Compare with Baseline methods

We evaluate our approach by comparing it with three baseline methods to demonstrate the effectiveness of our approach.

Baseline #1: Cuboid-based method. We replace every noncuboid brick in the training data with a cuboid brick of same size and connectivity, aiming at preserving the overall geometry. We then train our proposed model on this modified dataset and generate outputs with only cuboid bricks. This baseline highlights the expressiveness gained by using diverse nonrectangular bricks.

Baseline #2: Retrieval-based method. We precompute CLIP [Radford et al. 2021] features for all natural images in the training set. In the evaluation, we extract the CLIP feature of the input image, find the nearest neighbor by cosine similarity among the training features, and output the LEGO® model paired with the retrieved image. This baseline helps to reveal that our results are *not* created by simple memorization or retrieval.

Baseline #3: Basic OPT. We remove the proposed multi-token prediction (MTP) strategy, the rollback mechanism, and the procedural data-synthesis pipeline. Since only the facade dataset has a procedural pipeline, this ablation is conducted on the facade dataset. Unlike Section 7.1, we train with only 2,000 human-designed LEGO® facades, while keeping all other training settings unchanged.

7.3.1 Visual comparison. We apply the baselines and our method to generate LEGO® models for 2,000 facade evaluation samples and 300 portrait evaluation samples. Figure 12 presents a visual comparison of the facade models. After training, our method learns to closely mimic human designs, especially in window details, roof shapes, overall layout, and color accuracy. In contrast, Baseline #1 produces only basic shapes and color matches, failing to capture intricate details such as window boundary frames. Baseline #2 benefits from the large training set of 38,000 models and can retrieve designs with similar high-level semantic features like color distribution and general window styles. However, it struggles to reproduce fine-grained details and often fails to match the exact number and layout of basic elements, such as windows, resulting in less accurate facade compositions. Baseline #3 fails to generate buildable LEGO® models with coherent brick connections, often producing models with disconnected bricks, collisions, and structural artifacts.

7.3.2 Quantitative Comparison. We employ the following four sets of metrics and a user study to quantitatively evaluate all methods:

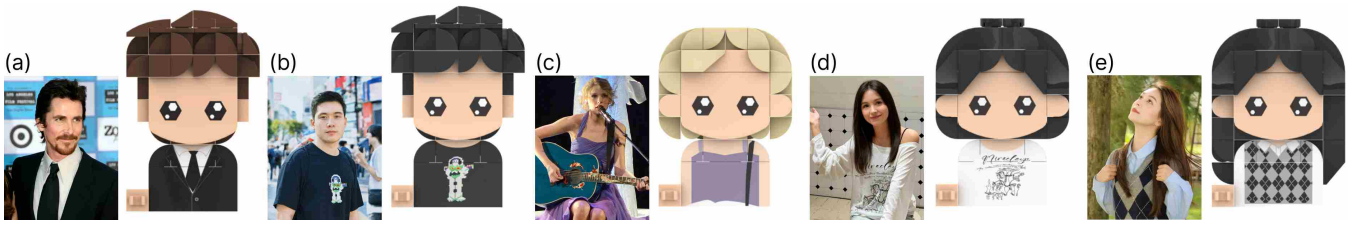


Fig. 11. LEGO® portraits models generated by LEGO®-Maker alongside with the reference image on the left of each generated model. Image (a) ©Asim Bharwani. Image (c) ©dephisticate. Images (b, d, & e) created by GPT-4o.

Table 1. Quantitative comparison of LEGO®-Maker with Baseline #2 (retrieval-based method) and Baseline #3 (basic OPT). Evaluation metrics include geometric, color, and semantic similarity against the ground-truth models. Structural integrity is measured by the percentage of models containing unconnected bricks. Additionally, three human perception scores (min: 1.0, max: 5.0) from a user study are reported.

		Geometry		**Color**	**Semantic**	**Structural Integrity**	**Human Assessment**		
Category	Method	CD ↓	EMD ↓	Color Metric ↑	Semantic Metric ↑	Floating Models ↓	Similarity Rating ↑	Color Rating ↑	Aesthetic Rating ↑
Facade	LEGO®-Maker (ours)	**0.0269**	**90.41**	**0.772**	**0.902**	0.05%	**3.24**	**3.74**	**3.84**
	Baseline #2 (Retrieval)	0.0774	188.24	0.680	0.873	**0%**	2.54	1.08	3.12
	Baseline #3	0.0737	173.85	0.649	0.822	6.25%	1.18	1.42	1.08
Portrait	LEGO®-Maker (ours)	**0.0308**	**96.81**	**0.623**	**0.810**	16.0%	**3.48**	**3.94**	**3.74**
	Baseline #2 (Retrieval)	0.0571	151.50	0.339	0.736	**0%**	2.08	2.54	2.72

Geometric metrics. To measure the geometric similarity between the generated models and ground-truth models, we compute the Chamfer Distance (CD) and Earth Mover's Distance (EMD). First, we sample 1,024 points on the frontal surfaces of the ground-truth models by casting rays from the camera position used in Figures 9 and 11, recording the depth from camera to surface points. Similarly, we sample points on the generated models to obtain comparable depth sets. We further normalize the depth values to $[0, 1]$ based on the minimum and maximum depths in the entire dataset. This amplifies the geometric features encoded in small depth variations, such as decorative brick details, window ledges, and wall textures. Then, we calculate CD and EMD between these normalized depths to robustly measure the geometric similarity.

Color metric. Color consistency is measured by computing color histograms from rendered images of the generated and ground-truth models, counting pixels per color bin. We compute the histogram intersection to quantify the overlap between the color distributions. This pixel-level metric (normalized to $[0, 1]$) assesses both global and local color fidelity, where a higher score means a better match.

Semantic metrics. Leveraging pretrained large language models, we define semantic faithfulness, separately for facades and portraits.

For facades, we use Grounding DINO [Liu et al. 2024b] to detect windows and doors in both input images and generated model renderings. Each detected element is represented by its type (window or door) and its row and column indices. Two bounding boxes are assigned the same row or column if their projections overlap on the vertical or horizontal axis, respectively. This forms an element set $E = (t_i, r_i, c_i)_{i=1}^{D}$, where D denotes the number of detected elements; t_i denotes type; r_i denotes row indices; and c_i denotes column indices. Semantic consistency between a LEGO® model $E_{\mathcal{M}}$ and input image E_{image} is computed as $S = \frac{|E_{\mathcal{M}} \cap E_{image}|}{\min(|E_{\mathcal{M}}|, |E_{image}|)}$, where a value closer to 1 indicates a better layout agreement.

For the LEGO® portrait dataset, we design a series of yes or no questions centering on the attributes shared by the human portraits and LEGO® renderings, *e.g.*, whether the subject in the portrait has long hair or short hair, or whether the subjects wear a hat, etc. Then, we ask GPT-4o [OpenAI 2024] to answer these questions, collect the set of answer values for all attributes, then define the semantic metric as the percentage of input-generated pairs with consistent attribute answers, reflecting semantic alignment.

Structural integrity metrics. Since our rollback mechanism helps to avoid brick collisions, our evaluation focuses on connectivity. It is worth noting that full buildability encompasses brick connectivity, collision avoidance, and physical stability. In this work, we have not explicitly considered physical stability and leave it for future work; see Section 8. Specifically, a LEGO® model is fully connected, if every brick connects to the main body without floating (unconnected) bricks. Such checking can be done by building a connectivity graph, with bricks as nodes and adjacent-bricks connections as edges. A model is considered structurally sound, if its connectivity graph is fully connected (*i.e.*, only one connected component), indicating that there are no floating parts. Table 1 reports the percentage of generated LEGO® models with floating bricks.

We compute evaluation scores on 2,000 facade and 300 portrait test samples, comparing models generated by our method against the baseline methods. Table 1 summarizes the results, in which each value reports an average across all models in the corresponding test set for each LEGO® category and method. Baseline #1 is excluded from the quantitative comparison due to its consistently poor performance with cuboid-shaped models, which are visually and

Fig. 12. Visual comparison of the LEGO® models designed in different approaches: our approach (2nd column) and three baseline methods (3rd to 5th columns). We also put the ground-truth models in the last column, as a reference.

intuitively inferior. In addition, since we do not apply data synthesis for portrait models, Baseline #3 experiments were not conducted on the portrait dataset.

From Table 1, it is clear that our approach outperforms all baselines on the geometric, color, semantic, and structural integrity metrics. For facades, the retrieval method (Baseline #2) performs better than Baseline #3 on the color and semantic metrics, as retrieval-based models at least preserve prominent visual features from the inputs, whereas Baseline #3 often fails to produce valid LEGO® constructions at this scale (~300 bricks). Notably, only 0.05% of the facade models generated by our method contain floating bricks—just one failure case out of 2,000 samples. In contrast, the failure rate rises to 16% for portrait models, likely due to the compact brick placement in these models, where even slight brick misplacements can cause disconnections from the main body of the model.

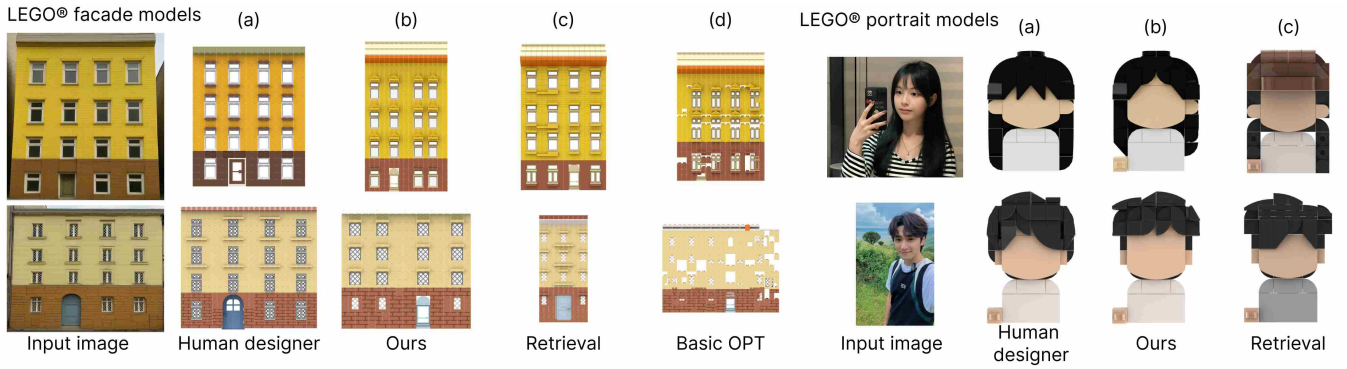
7.4 User study

To further evaluate the design quality of our approach based on human perception, we conduct a user study by recruiting ten participants and asking them to assess the LEGO® models produced by different methods. First, we randomly selected five facade images and five portrait images from our dataset. For each facade image, we obtained four LEGO® models created by (i) expert human designers, (ii) our approach, (iii) the retrieval-based method (Baseline #2), and (iv) Baseline #3. For each portrait image, we similarly gathered three LEGO® models, excluding Baseline #3, which is not applicable because we do not synthesize data for LEGO® portrait models. The participants were asked to rate each LEGO® model based on three criteria, image similarity, color accuracy, and overall model aesthetic, using a rating scale from 1 (worst) to 5 (best).

Figure 13 shows representative results from the four input images and shows the average participant scores for each method. Detailed average scores for all evaluated models are presented in Table 1 for comprehensive comparison. Overall, the results show that

Table 2. Comparison between LEGO®-Maker and ablated methods.

Category	Facade		Portrait	
Method	CD ↓	EMD ↓	CD ↓	EMD ↓
Ours	**0.0269**	**90.41**	0.0308	**96.81**
Ours w/o rollback	0.0270	90.81	**0.0282**	98.60
Ours w/o MTP	0.0296	92.63	0.0332	100.56
Ours w/o both	0.0300	93.50	0.0343	102.11
Rectangular bricks only	0.082	170.68	0.0469	119.64



Average human rating	Image similarity	4.36	3.24	2.54	1.18
	Color accuracy	4.52	3.74	1.08	1.42
	Model aesthetic	4.86	3.84	3.12	1.08

Average human rating	Image similarity	4.30	3.48	2.08
	Color accuracy	4.20	3.94	2.54
	Model aesthetic	4.72	3.74	2.72

Fig. 13. Human ratings on LEGO® models produced by different methods. We invited ten participants to assess the quality of the LEGO® models produced by (a) expert human designers, (b) our method, (c) a retrieval-based baseline, and (d) a basic OPT method. Participants rated each model based on image similarity, color accuracy, and overall aesthetic quality. The reported scores represent the average ratings (from 1 (worst) to 5 (best)) of all participants on each method. Note that the input images for LEGO® portrait model generation are created by GPT-4o.

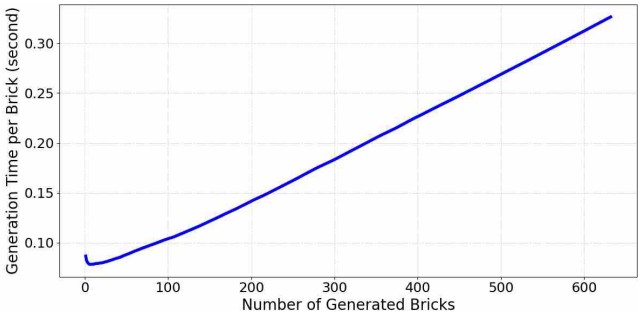


Fig. 14. Plot of brick generation time. For a LEGO® facade models with 675 piece, we plot the generation time (y-axis) per brick as a function of the number of bricks generated so far (x-axis).

our method consistently outperforms the baselines for all aspects, aligning with the results of the quantitative evaluation presented in Section 7.3. The visual comparisons also demonstrate that compared with the baselines, the LEGO® models generated by our method more closely approach the quality of the human designs, particularly on color fidelity and aesthetic appeal. Additional details of the user study can be found in Supplementary Material Part D.

7.5 Ablation study

To evaluate the effectiveness of the multi-token prediction (MTP) and the rollback mechanism in our approach, we conduct the following ablation study based on the LEGO® facade and portrait datasets: (i) generating the results without the rollback mechanism; (ii) training LEGO®-Maker without MTP; (iii) generating the results without both techniques; and (iv) substituting the LEGO® bricks in the generated models with only rectangular bricks. Table 2 shows the CD and EMD metrics of the ablated methods.

Removing rollback mechanism. The CD and EMD only decrease slightly without the rollback mechanism. This is reasonable, since rollback focuses on improving the buildability of the final results. We observed that without the rollback, the model places different types of bricks in the same place, leading to overlapping. However, the overlapping is usually not salient in the appearances of the generated models. When generating the 2,000 testing cases, about 28% of bricks trigger the rollback mechanism during inference.

Without multi-token prediction. To design the baseline without MTP, we preserve only the first output head in the autoregressive model and train the model with the vanilla loss term (1) in Section 5. Table 2 shows that our full method outperforms the ablated one on both CD and EMD. Besides the geometry metrics, we find that **61**% of models need rollback without MTP, while the percentage is reduced to **28**% when we train the model with MTP. This finding validates the strong local dependencies given by the MTP.

Rectangular bricks only. To validate the ability of our LEGO® tokenization scheme, which can scale our method up to over 100 types of brick, we substitute the bricks in the generated models with the rectangular bricks with the same bounding boxes. Figure 12 shows the visual comparison between our method. Our method makes full use of the expressivity of various types of bricks. The geometry metrics are also reported in Table 2.

7.6 Generation speed

Lastly, we evaluate the generation speed of our model by measuring the time it takes to generate bricks. Figure 14 plots the average time

to generate a brick versus the number of generated bricks. From the plot, we can see that the time taken increases roughly linearly with the number of bricks. The underlying reason is that each new token has to attend to all the previously generated tokens, thereby gradually slowing down the overall generation.

Consequently, the total generation time increases quadratically with the number of bricks. Our method generates 100 bricks in about 11 seconds, 300 bricks in one minute, and 600 bricks in roughly three minutes. This is significantly faster than [Ge et al. 2024a], which requires around 40 seconds to generate 100 bricks.

8 Conclusion, Limitations, and Future Works

In conclusion, this paper introduces the first autoregressive model LEGO®-Maker that can generate standard-scale, multi-color LEGO® models conditioned on natural images. Importantly, it is capable of reproducing the structure and details in the given reference images, considering diverse brick types and colors, and being versatile in generating LEGO® models of both facade and portrait categories.

The main technical contributions of this work are summarized below: (i) the overall architecture that enables effective learning and generating LEGO® models with more than 600 bricks, considering over 100 brick types, 30 colors, and a maximum size of 35.2 cm; (ii) the LEGO® tokenization scheme, which serializes LEGO® models brick-by-brick in a bottom-up manner, encoding the brick type, color, orientation, and position, to support efficient autoregressive learning; (iii) LEGO®-Maker, equipped with the multi-token prediction strategy and rollback mechanism, making our model effective in pre-considering multiple brick attributes and achieving collision-free LEGO® model generation; and (iv) an effective data preparation pipeline, comprising a procedural generator and a LEGO®-to-real image translator, leveraging 3D priors from GPT-4o to bridge the domain gap between LEGO® models and natural images.

Trained on the LEGO® facade and portrait datasets, we demonstrate that LEGO®-Maker can successfully generate a rich variety of high-quality LEGO® models of varying layouts, model sizes, and vivid colors. We extensively evaluate our approach on metrics in four aspects, *i.e.*, geometry, color, semantics, and structural integrity, together with a user study. Comparing our method with three alternative baselines, LEGO®-Maker showcases compelling capabilities of generating detailed results, while faithfully reproducing the structures and colors in the reference images.

Limitations. Though the LEGO® models generated by our method rival those created by experienced human designers, they are still far from the level of complexity and expressiveness in commercial LEGO® products. First, this work has not explicitly incorporated physical stability in its method. Second, due to the challenges of acquiring high-quality LEGO® models paired with images, we only tested LEGO®-Maker on two object categories. Third, we consider colors in a relatively limited color palette following the standard LEGO® models, while we can consider continuous color distribution when tokenization and create more colorful LEGO® models for virtual showing. Finally, we limit the brick orientation to axis-aligned options, while generic LEGO® models, when incorporating diverse connection mechanisms such as hinge connections, may exhibit arbitrary orientations to enhance the model expressiveness.

Future work. Inspired by the recent success of autoregressive models and our results, we foresee a vast space for conditional generation of LEGO® models in terms of generality, complexity, and diversity. To this end, large datasets featuring human-level LEGO® models and real natural images are necessary. Future research directions could focus on (i) expanding the model categories by curating larger-scale datasets of diverse LEGO® designs paired with associated images, text descriptions, or 3D meshes; (ii) enriching our generative approach with multi-modal capability by incorporating text descriptions and geometric conditions (*e.g.*, point clouds, depth maps, etc.), offering higher generation flexibility; (iii) extending the framework to synthesize larger scenes composed of multiple LEGO® models, enabling reconstructions of city-scale LEGO® scenes; and (iv) integrating stability constraints into the generative process with physics-based simulations and stress analysis.

Acknowledgments

We would like to express our sincere gratitude to the anonymous reviewers for their insightful comments and feedback. We are also deeply thankful to the designer team, as well as Shiquan Zhang, Xinyan Chen, and Shaoxuan Yuan, for their help to prepare the data samples. Our appreciation extends further to all the participants who generously took part in the user study. This work is supported by the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK 14201921).

References

BrickLink. 2024. BrickLink - Studio. https://www.bricklink.com/v2/build/studio.page

Yiwen Chen, Tong He, Di Huang, Weicai Ye, Sijin Chen, Jiaxiang Tang, Zhongang Cai, Lei Yang, Gang Yu, Guosheng Lin, and Chi Zhang. 2025a. MeshAnything: Artist-created mesh generation with autoregressive transformers. In *International Conference on Learning Representations (ICLR)*.

Yongwei Chen, Yushi Lan, Shangchen Zhou, Tengfei Wang, and Xingang Pan. 2025b. SAR3D: autoregressive 3D object generation and understanding via multi-scale 3D VQVAE. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 28371–28382.

Yiwen Chen, Yikai Wang, Yihao Luo, Zhengyi Wang, Zilong Chen, Jun Zhu, Chi Zhang, and Guosheng Lin. 2025c. MeshAnything V2: Artist-created mesh generation with adjacent mesh tokenization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. 2023a. Objaverse-XL: A universe of 10m+ 3D objects. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. 2023b. Objaverse: A universe of annotated 3D objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 13142–13153.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. 2024. Scaling rectified flow Transformers for high-resolution image synthesis. In *Proceedings of the International Conference on Machine Learning (ICML)*. 12606–12633.

Jiahao Ge, Mingjun Zhou, Wenrui Bao, Hao Xu, and Chi-Wing Fu. 2024b. Creating LEGO® figurines from single images. *ACM Trans. Graph. (SIGGRAPH 2024)* 43, 4 (2024), 1–16.

Jiahao Ge, Mingjun Zhou, and Chi-Wing Fu. 2024a. Learn to create simple LEGO® micro buildings. *ACM Trans. Graph. (SIGGRAPH ASIA 2024)* 43, 6 (2024), 1–13.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*. 15706–15734.

Rebecca A. H. Gower, Agnes E. Heydtmann, and Henrik G. Petersen. 1998. LEGO®: Automated model construction. In *European Study Group with Industry*. 81–94.

Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. 2024. LRM: Large reconstruction model for single image to 3D. In *International Conference on Learning Representations (ICLR)*.

Li Hu. 2024. Animate anyone: Consistent and controllable image-to-video synthesis for character animation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 8153–8163.

Junming Huang, Chi Wang, Letian Li, Changxin Huang, Qiang Dai, and Weiwei Xu. 2025. BuildingBlock: A hybrid approach for structured building generation. In *SIGGRAPH 2025 Conference Papers* (Vancouver, BC, Canada).

Ka-Hei Hui, Aditya Sanghi, Arianna Rampini, Kamal Rahimi Malekshan, Zhengzhe Liu, Hooman Shayani, and Chi-Wing Fu. 2024. Make-A-Shape: A ten-million-scale 3D shape model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. 20660–20681.

James Jessiman. 1995. LDraw.Org - LDraw file format specification. https://ldraw.org/article/218.html.

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

Yushi Lan, Shangchen Zhou, Zhaoyang Lyu, Fangzhou Hong, Shuai Yang, Bo Dai, Xingang Pan, and Chen Change Loy. 2025. GaussianAnything: Interactive point cloud flow matching for 3D object generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Jae Joong Lee, Bosheng Li, and Bedrich Benes. 2023. Latent L-Systems: Transformer-based tree generator. *ACM Trans. Graph.* 43, 1, Article 7 (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, Yamin Beveridge, Matthewand Arefeen, Nikhil Singh, and Iddo Drori. 2021. Image2Lego: Customized LEGO® set generation from images. *arXiv:2108.08477* (2021).

Jiahao Li, Hao Tan, Kai Zhang, Zexiang Xu, Fujun Luan, Yinghao Xu, Yicong Hong, Kalyan Sunkavalli, Greg Shakhnarovich, and Sai Bi. 2024b. Instant3D: Fast text-to-3D with sparse-view generation and large reconstruction model. In *International Conference on Learning Representations (ICLR)*.

Weiyu Li, Rui Chen, Xuelin Chen, and Ping Tan. 2024a. SweetDreamer: Aligning geometric priors in 2D diffusion for consistent text-to-3D. In *International Conference on Learning Representations (ICLR)*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv:2412.19437* (2024).

Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. 2023. Zero-1-to-3: Zero-shot one image to 3D object. In *International Conference on Computer Vision (ICCV)*. 9298–9309.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. 2024b. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *European Conference on Computer Vision (ECCV)*. 38–55.

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. (SIGGRAPH ASIA 2015)* 34, 6 (2015), 1–18.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*. 405–421.

Vihaan Misra, Peter Schaldenbrand, and Jean Oh. 2025. ShapeShift: Towards text-to-shape arrangement synthesis with content-aware geometric constraints. *arXiv:2503.14720* (2025).

OpenAI. 2024. GPT-4o system card. https://arxiv.org/abs/2410.21276. Accessed: 2025-05-12.

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, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. 2024. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)* (2024).

Pavel Petrovič. 2001. Solving LEGO® brick layout problem using evolutionary algorithms. In *Proc. NIK (Norsk Informatikkonferanse)*. 87–97.

Maxim Peysakhov, Vlada Galinskaya, and William C Regli. 2000. Representation and evolution of LEGO®-based assemblies. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*. 1089.

Aleksander Plocharski, Jan Swidzinski, Joanna Porter-Sobieraj, and Przemyslaw Musialski. 2024. FaçAID: A Transformer model for neuro-symbolic facade reconstruction. In *SIGGRAPH Asia 2024 Conference Papers*.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2023. DreamFusion: Text-to-3D using 2D diffusion. In *International Conference on Learning Representations (ICLR)*.

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Lingteng Qiu, Guanying Chen, Xiaodong Gu, Qi Zuo, Mutian Xu, Yushuang Wu, Weihao Yuan, Zilong Dong, Liefeng Bo, and Xiaoguang Han. 2024. RichDreamer: A generalizable normal-depth diffusion model for detail richness in text-to-3D. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 9914–9925.

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 International Conference on Machine Learning (ICML)*. 8748–8763.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *Proceedings of the International Conference on Machine Learning (ICML)*. 8821–8831.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 10684–10695.

Jarek Rossignac. 2002. Edgebreaker: Connectivity compression for triangle meshes. *IEEE transactions on visualization and computer graphics (TVCG)* 5, 1 (2002), 47–61.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Raphael Gontijo-Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. 2022. Photorealistic text-to-image diffusion models with deep language understanding. In *International Conference on Neural Information Processing Systems (NeurIPS)*. 36479–36494.

Junyoung Seo, Wooseok Jang, Min-Seop Kwak, Inès Hyeonsu Kim, Jaehoon Ko, Junho Kim, Jin-Hwa Kim, Jiyoung Lee, and Seungryong Kim. 2024. Let 2D diffusion model know 3D-consistency for robust text-to-3D generation. In *International Conference on Learning Representations (ICLR)*.

Ruoxi Shi, Hansheng Chen, Zhuoyang Zhang, Minghua Liu, Chao Xu, Xinyue Wei, Linghao Chen, Chong Zeng, and Hao Su. 2023. Zero123++: A single image to consistent multi-view diffusion base model. arXiv:2310.15110 [cs.CV]

Yichun Shi, Peng Wang, Jianglong Ye, Long Mai, Kejie Li, and Xiao Yang. 2024. MVDream: Multi-view diffusion for 3D generation. In *International Conference on Learning Representations (ICLR)*.

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 *Conference on Computer Vision and Pattern Recognition (CVPR)*. 19615–19625.

Eugene Smal. 2008. *Automated Brick Sculpture Construction.* Thesis. Stellenbosch : Stellenbosch University.

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 *International Conference on Learning Representations (ICLR)*.

Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. 2024. DreamGaussian: Generative gaussian splatting for efficient 3D content creation. In *International Conference on Learning Representations (ICLR)*.

Romain Testuz, Yuliy Schwartzburg, and Mark Pauly. 2013. Automatic generation of constructible brick sculptures. In *Eurographics (short paper)*. 81–84.

Rylee Thompson, Ghalebi Elahe, Terrance DeVries, and Graham W. Taylor. 2020. Building LEGO® using deep generative models of graphs. *Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems* (2020).

Si-Tong Wei, Rui-Huan Wang, Chuan-Zhi Zhou, Baoquan Chen, and Peng-Shuai Wang. 2025. OctGPT: Octree-based multiscale autoregressive models for 3D shape generation. In *SIGGRAPH 2025 Conference Papers*.

David V. Winkler. 2005. Automated brick layout. In *Proc. BrickFest*. 145–166.

Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Jingxi Xu, Philip Torr, Xun Cao, and Yao Yao. 2024. Direct3D: Scalable image-to-3D generation via 3D latent diffusion Transformer. In *International Conference on Neural Information Processing Systems (NeurIPS)*. 121859–121881.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. 2025. Structured 3D latents for scalable and versatile 3D generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. 2019. Computational LEGO® technic design. *ACM Trans. Graph. (SIGGRAPH Asia 2019)* 38, 6, Article 196 (2019), 14 pages.

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

Jingwen Ye, Yuze He, Yanning Zhou, Yiqin Zhu, Kaiwen Xiao, Yong-Jin Liu, Wei Yang, and Xiao Han. 2025. PrimitiveAnything: Human-crafted 3D primitive assembly generation with auto-regressive Transformer. In *SIGGRAPH 2025 Conference Papers*.

Chubin Zhang, Hongliang Song, Yi Wei, Chen Yu, Jiwen Lu, and Yansong Tang. 2024a. GeoLRM: Geometry-aware large reconstruction model for high-quality 3D gaussian generation. *International Conference on Neural Information Processing Systems (NeurIPS)* (2024), 55761–55784.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 3813–3824.

Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. 2024b. CLAY: A controllable large-scale generative model for creating high-quality 3D assets. *ACM Trans. Graph. (SIGGRAPH 2024)* 43, 4 (2024), 1–20.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open pre-trained transformer language models. *arXiv:2205.01068* (2022).

Jie Zhou, Xuejin Chen, and Ying-Qing Xu. 2019. Automatic generation of vivid LEGO® architectural sculptures. In *Computer Graphics Forum*, Vol. 38. 31–42.

Mingjun Zhou, Jiahao Ge, Hao Xu, and Chi-Wing Fu. 2023. Computational design of LEGO® sketch art. *ACM Trans. Graph. (SIGGRAPH Asia 2023)* 42, 6 (2023), 1–15.

Junzhe Zhu, Peiye Zhuang, and Sanmi Koyejo. 2024. HiFA: High-fidelity text-to-3D generation with advanced diffusion guidance. In *International Conference on Learning Representations (ICLR)*.