BrickNet: Graph-Backed Generative Brick Assembly

Peter Kulits[1,2] Cordelia Schmid[1]
[1]Inria, École Normale Supèrieure, CNRS, PSL Research University [2]Max Planck Institute for Intelligent Systems, Tübingen

https://kulits.github.io/BrickNet

Figure 1. We finetune an LLM to autoregressively generate LEGO-brick build sequences. To enable this, we introduce a large-scale dataset of LDraw [15] structures composed of parts (samples above), as well as a novel graph-backed parametrization to represent arbitrary objects.

Abstract

We train a language model to generate LEGO®-brick build sequences. While prior work has been restricted to discrete, voxel-like towers, we consider a much broader set of pieces, encompassing thousands of part types with diverse connection semantics. To enable this, we first collect a large-scale dataset of over 100,000 human-designed LDraw brick objects and scenes. The complexity of our setting makes it challenging to autoregressively assemble structures that satisfy physical constraints. When predicting block pose directly, build sequences quickly become invalid after a small number of steps. Although pieces are placed in 3D space, it is the spatial relationships of the parts which define the whole. With this in mind, we design a graph-based program representation that parametrizes structure through connectivity, improving the physical grounding of generated sequences. To enable future applications, we make our dataset and models available for research purposes.

1. Introduction

Many objects are naturally described at the level of parts and their configuration. Recent work in 3D generation has explored representations that encode these relationships explicitly, including part graphs [25] and executable shape programs [16, 28]. The resulting generative problem is then to produce objects whose global geometry and compositional structure are consistent.

Sequential LEGO assembly offers a compact instance of this broader problem. A brick structure is defined not only by the arrangement of its parts but also by its construction. Each part added must satisfy discrete rules, which constrain the set of valid continuations. At the same time, the domain is also quite expressive, encompassing thousands of part types with rich connection diversity and semantics (Fig. 1).

However, while a number of approaches for generative brick assembly have been proposed, they have been restricted to toy subsets of this representational space. Typically as-

suming a discrete grid and only a handful of part types, triangular meshes can be voxelized to produce training samples (Fig. 2a). While useful as initial demonstrations, we argue such domains lack the expressivity which makes LEGO a particularly compelling domain for sequential generation.

One reason that this richer generation setting has not been studied is the lack of suitable training data, which can not as easily be bootstrapped. To address this, we propose BrickNet, the first large-scale dataset of human-designed brick structures. Curated from publicly available online sources, our dataset encompasses 320,808 samples, 9,743 part variants, and cumulatively 40,549,969 placed bricks.

Yet, from the increased expressivity arise representational challenges. When restricted to a simple grid, up is up, the rotational frame is constant, and predicting coordinates seems natural. However, real-world samples do not adhere to these assumptions. Take, for example, the dragonfly in Fig. 2b. To autoregressively assemble it from, say, the parts from 1 to 5, a model must keep track of the 6-DoF pose of each block in-between, and accumulate transforms along a shifted rotational frame, which becomes a problem of precision.

Instead, our insight is to make connectivity first-class. To do so, we annotate each part with typed connectors and pairing semantics. Then, we design a compact parametrization of structure that represents spatial relationships between parts through a graph of their connectivity. In doing so, arbitrary structures can be serialized as a spanning tree, which can be executed to recover the original spatial transforms.

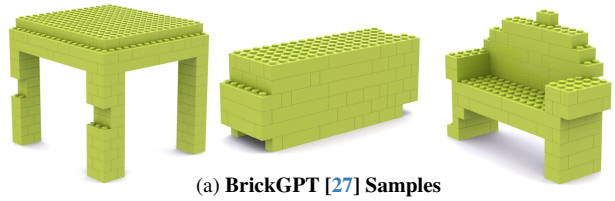
In summary, our primary contributions are:

1. A large-scale human-designed LDraw-structure dataset
2. Typed connectivity annotations for each of the parts
3. A graph-backed parametrization of part connectivity
4. Autoregressive models trained on our representation

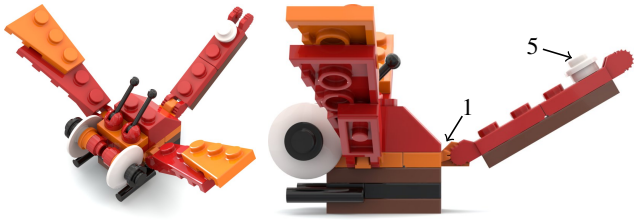
2. Related Work

Bricks and Assembly. LEGO-brick assembly has been explored as both a generative and a reconstruction task. Existing generative approaches have been restricted to discrete grids with small part vocabularies. Some place bricks sequentially [11, 17], while others first predict a full volume and then decompose it into valid placements [1, 10]. Similarly to our work, Peysakhov and Regli [26] employ a graph parametrization of brick structure, but model only snap-fit connections between rectangular blocks and evolve structures using genetic algorithms rather that learning composition. Thompson et al. [31] adopt this representation to train an autoregressive graph neural network, but instead are restricted to a single brick type and grid-aligned output positions. BrickGPT [27] serializes block placements as text to finetune a language model to autoregressively produce build sequences as we do, but predict voxel positions.

For reconstruction, Chung et al. [6] train a reinforcement learning agent to assemble structures given multi-view images. Guo et al. [12] model assembly order as a breadth-first tree and train a model in a self-supervised manner. Wang et al. [35] address a different input modality, inferring 3D block pose from visual instruction manuals by learning 2D–3D correspondence. Walsman et al. [33, 34] task an agent in a simulator to disassemble and reassemble structures.



(a) **BrickGPT [27] Samples**



(b) **Human-Designed Sample**

Figure 2. **Motivation.** To teach a model to autoregressively generate brick structures in a discrete, voxelized domain, it is intuitive to train it to regress 3D coordinates (Fig. 2a). However, doing so becomes more difficult when dealing with the complexities of real-world objects (Fig. 2b). Starting at the orange hinge plate (1) and placing bricks down to the white stud (5) at the end requires maintaining a high degree of numerical precision across steps.

Program Synthesis for 3D Structure. A parallel line of work models 3D structure through executable programs rather than raw geometry. CSGNet [28] and InverseCSG [8] recover constructive solid geometry trees from images or 3D input. ShapeAssembly [16] generates hierarchical part-placement programs that can be executed to produce 3D shapes. StructureNet [25] models part-compositional structure through graphs of hierarchical relationships. In a similar vein, we serialize brick structure as a spanning tree over a typed connectivity graph, where each edge is an instruction that can be realized to produce a local SE(3) transformation between parts.

LLMs and 3D Understanding. LLMs have recently demonstrated their versatility and potential in various tasks within 3D domains, expanding their utility beyond conventional text-based applications. These tasks encompass a broad spectrum, including answering questions about 3D scenes [9, 13], planning and generating motion in 3D environments [13, 22, 40], reconstructing scenes [2, 19, 20], synthesizing 3D scenes from textual descriptions [14, 29, 39], editing procedural models (those generated by sets of rules or algorithms) using natural language instructions [18], and learning multi-modal representations that bridge text and 3D domains [13, 37].

(a) **Stud** (b) **Hinge** (c) **Axle** (d) **Ball** (e) **Fixed**

Figure 3. **Connectivity Semantics.** We broadly model five types of connectivity between bricks. Stud (Fig. 3a) connections, after defining which stud connects to which hole, have at most one degree of freedom. Hinge (Fig. 3b) connections have a degree of rotational freedom, and often the ability to be flipped (binary). Axle (Fig. 3c) connections inherit the same freedom as hinges, but can also be offset along their principal axis. Ball (Fig. 3d) connections have three degrees of rotational freedom. Fixed (Fig. 3e) connections have no degrees of freedom.

3. Representation

3.1. Preliminaries

We build on the LDraw [15] ecosystem, a community-maintained standard and library that is ubiquitous for digital LEGO modeling. The part library consists of over 24,000 CAD replicas, covering most real-world pieces. Structure is represented in plain text as a set of references to these pre-defined parts, with attributes of instance color, 3D position, and rotation matrix:

```
color x y z a b c d e f g h i part
```

Positional units are expressed in LDU (LDraw Unit), where 1 LDU $\approx$ 0.4 mm and the diameter of a stud is 6 LDU.

Prior work has employed simplified representations. Most generative-assembly approaches directly predict voxel occupancy then decompose it into grid-aligned brick positions [1]. Some have also used graphs for generation [31], but model only discrete structures. BrickGPT [27], the only existing text-serialized approach we are aware of, uses eight brick types and restricts placement to a discrete $20\times20\times20$ grid:

```
1x2 (7,6,0)
```

In contrast, Walsman et al. [33, 34] – that do not target a generative problem – make use of a broader set of parts. They use the LDCad [24] snap system to define connection sites and task an agent to visually select positions on pieces that should connect. While much more general than a grid-based approach, the system lacks broad connector coverage.

3.2. Connectivity

In this section, we introduce our connector semantics. We categorize connections between bricks into five families:

Stud. The most common type of connection is that between a stud and a hole. As visualized in Fig. 3a, once connected, at most one degree of rotational freedom is possible. After keeping track of which stud is attached to which hole, only one rotational yaw parameter must be stored to fully represent the SE(3) transformation between two parts, greatly reducing dimensionality. It is also true that two or more stud connections between parts determine rotation exactly, but we concern ourselves only with single-contact connections.

Within stud-type connectors, we subdivide into standard stud, open stud, hole, tube, and post. Any stud type may pair with holes or tubes, but only open studs may connect to the posts found on the underside of some parts. While a tube may also fit between four studs, the connection is non-standard and we do not model it. We also do not model "illegal" connections (see Berard [3] for discussion).

Hinge. In addition to the one degree of rotational freedom represented in stud-type connections, hinges must also encode a boolean "flip." This is akin to separating a pair, rotating one 180 degrees in the axial, and re-connecting (Fig. 3b). Hinges also have a broader range of in/on subtype pairs, which we model separately.

Axle. Axle connections extend hinges in that they involve one degree of rotation and a flip but also require parametrizing the "slide" along the axis (Fig. 3c). Within the category, we separately model pins, axles, pin sockets, axle sockets, bars, and clips. Pins pair only with pin sockets, but axles can pair with both pin and axle sockets. Clips connect with bars.

Ball. Ball-type connections similarly generalize hinges, but instead of a binary 180-degree "flip," have two additional degrees of rotational freedom (Fig. 3d). We subtype connections into "towball" and socket, and "technic" and socket.

Fixed. Finally, some connections are "fixed" and allow for no meaningful freedom between the two parts in a pair, so knowing which connector is paired with which is sufficient to represent the spatial relationship. One such example is the attachment of a wheel hub to a tire (Fig. 3e).

3.3. Connector Annotation

To annotate the LDraw part library with our connector taxonomy, we perform a mixture of procedural annotation and manual authoring. Bricks in the LDraw system are generally and conveniently defined by varying levels of subcomponents. By identifying a "stud.dat" in the primitive hierarchy after verifying the scale in the composed rotation matrix,

a precise connector position can be inferred. While we reviewed all annotations, many edge cases exist, and not all connector sites unambiguously represent one type or another.

During stud-site filtering, we performed collision checks to determine whether a piece could indeed be connected at a given site. However, we found collision estimation to be somewhat uniquely difficult in this setting. First, even assuming perfect LDraw modeling, real parts can only connect to one another through some degree of stress-based plastic deformation. This means that, even in a pair of parts that are well-connected, some level of collision is a given. In traditional collision estimation, the amount of overlap between two meshes can be computed. However, brick connections are very non-convex, and if a tube tightly connected to a stud collides with it on all sides, the depenetration solution is to remove the stud from the tube. Additionally, the parts are largely non-watertight, so methods such as VHACD [23] cannot be applied to decompose them into convex colliders. To resolve these issues, we devise a multi-stage pipeline to attempt to render the part library watertight. Following this process, we applied a modified version of PFPOffset [5] to inset all part-mesh faces by 0.25 LDU (0.1 mm). These meshes are then used for standard collision detection in both filtering and, later, generation evaluation.

3.4. Graph

Given an unordered set of part instances, we follow the above matching semantics to pair connectors and form edges. As designed, each edge defines the full local SE(3) transform between two paired parts. Together, the edges form a graph of connectivity. As each edge is sufficient, the full structure can be compactly and interpretably represented by a spanning tree of the graph. See Fig. 4a for a visualization of a parsed object and Fig. 4b for a view colored by step order.

Starting with an arbitrary root, we sample a sequence of build steps that define the part to add and specify how it is to be connected onto the existing structure. These build steps define a program that, when executed, produces a set of parts with 6-DoF pose. During serialization, we round rotational parameters to the nearest degree and the slide scalar to the nearest LDU. See Fig. 4c for a sample build sequence.

4. Dataset

We collect a large-scale dataset of publicly available LDraw-format structures online. This includes 40,549,969 instances of 9,743 unique parts across 320,808 samples.

We define two overlapping sets from this, BrickNet-PT (pretraining) and BrickNet-SFT (fine-tuning). BrickNet-SFT contains 67,185 samples, with 1,774,387 cumulative instances, between four and 100 parts that fulfill part-color and part-type diversity criteria. Each sample was additionally required to have zero collisions across the assembled

(a) **Object** (b) **Path-Colored**

```
a plate 1x2 | purple
b brick 1x2 | purple
a stud hole b stud b 90
c brick 1x2 | purple
a stud hole a stud b 90
. . .
z plate 2x2 wheels holder | purple
w stud hole d stud b 90
. . .
ac wheel rim 6.4x8 | light grey
z axle bar b clip a flip 270 0
. . .
ah tyre 6 50x8 | black
ac fixed e in b a
```

(c) **Build Sequence**

Figure 4. **Graph Visualization.** After encoding relative transformations between parts into their connectivity, we arrive at connected graphs. From these graphs, we can sample iterative build instructions (spanning trees), that begin at a root part, add another part, define an edge that connects that part with the existing structure, and on. For example, see the dark red piece at the top of the render in Fig. 4b. This corresponds to part 0 in the graph, the root node. From that, it has two neighbors, both `brick 1x2`, which are added to the structure. Each bracketed item corresponds to a discrete placement "action."

structure. We render each sample from eight views each and generate captions using Gemini 2.5 [7].

We hold out 512 additional samples meeting the same criteria for evaluation. Evaluation samples are drawn from source files containing a single object.

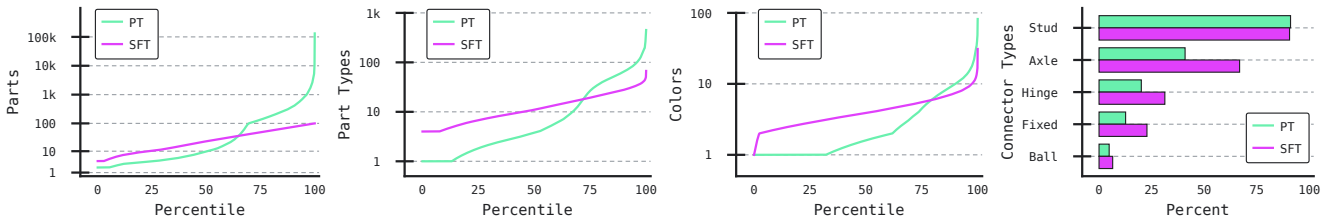


Figure 5. **Dataset Plots.** We compute statistics over both BrickNet subsets. While SFT samples are capped at 100 parts, the BrickNet-PT set is long-tailed and can include thousands of parts (Fig. 5a). The number of unique parts (Fig. 5b) and colors (Fig. 5c) per object are also similarly tailed. We additionally compute the proportion of samples containing an instance of each connection-type class (Fig. 5d).

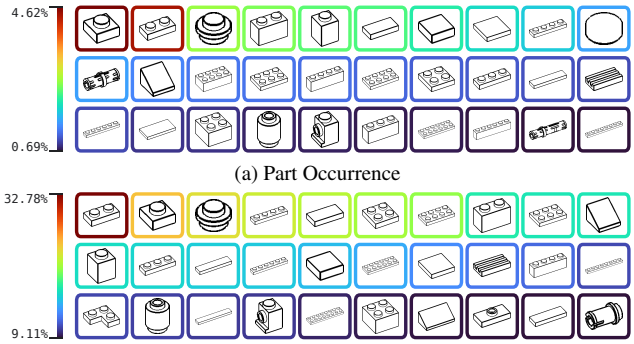


Figure 6. **Part Frequency.** We compute relative part frequency (Fig. 6a) and the proportion of samples each part occurs in (Fig. 6b).

Table 1. **Dataset Statistics**. We present BrickNet, a large-scale dataset of brick structures. In contrast to the voxelized BrickGPT [27], our samples include thousands of distinct part types. Relative to the Official Model Repository (OMR) [15] data used in Walsman et al. [33], our set is much broader.

Dataset	Samples	Parts	Color	Captions	Real
BrickGPT [27]	28,259	8	✗	✓	✗
OMR [33]	1,814	5,005	✓	✗	✓
BrickNet-PT	320,808	9,743	✓	✗	✓
BrickNet-SFT	67,185	6,457	✓	✓	✓

The remaining data is considerably more long-tailed. We include it with the SFT train samples and label it BrickNet-PT. It contains more than an order of magnitude more part instances, but often in the form of very large structures. See Tab. 1 and Figs. 5 and 6 for dataset statistics.

5. Evaluations

5.1. Unconditional Generation

Our first evaluation concerns unconditional sequence generation. Given a beginning-of-sequence token, the task is to produce a physically valid, collision-free structure. For this, we use BrickNet-PT and sample build sequences of up to 100 pieces each. We sample them in proportion to the square root of the number of pieces in a given structure. While the internet-curated objects themselves exhibit numerous part–part collisions, we perform collision detection at path-sampling time to ensure training paths are valid.

We serialize the sampled sequences in both our proposed graph-backed form described in Sec. 3.4 and a naive non-graph pose-based form of:

```
part | color x y z yaw pitch roll
```

Table 2. **Unconditional Generation**. We sample 2^{16} full-length sequences (100 parts) from each of our models. We evaluate validity of each build prefix by whether it can be, first, realized as a connected structure, and second, be free of placement collisions. The numbers reported represent the average number of successful build steps until an invalidating step. We report results for both nucleus sampling (NS) and ancestral sampling (AS).

	Connectivity (AS / NS)		Collision (AS / NS)	
Size	Graph	Pose	Graph	Pose
0.6b	45.0 / 94.1	13.6 / 31.8	11.0 / 16.0	8.1 / 14.5
1.7b	53.7 / 95.1	15.1 / 35.5	11.7 / 16.6	8.8 / 16.1
4b	49.9 / 96.9	18.3 / 45.1	12.2 / 18.0	10.6 / 20.3
8b	56.6 / 97.0	17.4 / 44.9	12.7 / 18.7	10.2 / 20.1
14b	55.2 / 96.9	20.3 / 49.9	13.0 / 19.1	11.6 / 22.4

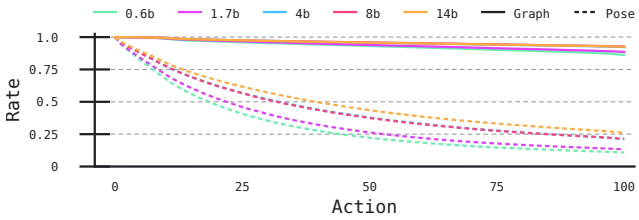


Figure 7. **Connectivity Survival.** On the nucleus-sampled sequences produced from our unconditional models, we compute the proportion of generations which "survived" at least k placement actions before an unparseable or unsupported action was sampled.

(a) BrECS [1]

(b) Ours

Figure 8. **Unconditional Samples.** Random samples drawn from either BrECS [1] (Fig. 8a) or our model (Fig. 8b). 🔎 Zoom in for details.

Using these sequences, we finetune the 0.6b-, 1.7b-, 4b-, 8b-, and 14b-parameter Qwen 3 [38] instruct models. We cap sequence length at 4,096 tokens and train using only a standard next-token-prediction cross-entropy loss:

$$p(x) = \prod_{i=1}^{n} p\left(s_i | s_1, \ldots s_{i-1}\right) \tag{1}$$

After training, we sample 2^{16} build sequences from each model at full temperature, suppressing the EOS token to force full-length generation. With true full-temperature ancestral sampling (AS), we observe that the majority of generations for both the pose and graph models are invalidated by the sampling of garbage tokens. At a sequence length of 4096, if only 0.1% of next-token probability mass lies on invalid completions, the chance of sampling a valid sequence is less than 1.7%. We find that nucleus sampling (NS) with a top-k of 20 and top-p of 0.95 alleviates this, with average connectivity validity doubling (Tab. 2), which we adopt for subsequent evaluations.

Following the change in sampling, the graph-backed approach achieves an average parseability / connectivity validity of 94 or more placement actions, while the direct pose approach does not reach 50. To better understand this, we plot survival in Fig. 7 and observe steep validity falloff. In

Table 3. **Text-Conditioned Generation**. Generation results on our 512-sample evaluation set. P_{inv} represents the proportion of invalid placements. VQAScore [21], PE [4], and SigLIP 2 [32] are measures of image–text similarity computed on object renders.

Model	P_{inv}	VQAScore	PE	SigLIP 2
BrickGPT [27]	0.063	0.050	0.157	0.052
0.6b	0.256	0.557	0.279	0.603
1.7b	0.260	0.593	0.282	0.631
4b	0.239	0.615	0.283	0.639
8b	0.233	0.608	0.284	0.647
14b	0.231	0.602	0.283	0.625

contrast, the graph-backed parametrization does not exhibit this steep falloff, as connectivity is encoded directly into the representation. However, when collision avoidance is included, the approaches are comparable, with approximately 20 steps (Tab. 2). Across metrics the 0.6b model performs worst, but there appear to be diminishing returns in how greater size translate to higher validity.

We show sample generations from the graph-backed 14b in Fig. 8 against those of BrECS [1], a recent generative model supervised with ModelNet-40 [36] voxel structures.

5.2. Text-Conditioned Generation

Next, we finetune our graph-backed PT models on BrickNet-SFT for the task of text-conditioned structure generation. We follow the same training objective as in the PT eval.

We evaluate on 512 held-out test prompts and score renders of our generations perceptually against the text prompts using 1) VQAScore [21] with Qwen2.5-VL 7b [30]; 2) Perception Encoder (PE) [4]; and 3) SigLIP 2 [32]. If a sequence is not parseable, we regenerate it, which was necessary to evaluate all model-size variants. We compare against prior work BrickGPT [27] and provide samples for both in Fig. 9.

While we observed fairly consistent quantitative trends in the unconditional evaluation across model sizes (Tab. 3), the pattern in perceptual quality is less clear. The 0.6b model underperforms all others, but 1.7b outperforms 14b on one metric and 14b is the best only in avoiding invalid placements. This suggests that model capacity may not be a key limiting factor holding back quantitative performance. However, with the exception of the ordering between 0.6b and 1.7b, the proportion of invalid placements appears to trend with size, in line with findings from the unconditional stage.

We additionally evaluate model perplexity on the ground-truth held-out sequences in Tab. 4 for both the PT and SFT graph models, and for varying amounts of PT or SFT data. In contrast to the perceptual evaluation, we find a consistent ranking as model and dataset size are increased. We suggest that this may indicate misalignment between the training objective – which is being correctly minimized – and the sampling task. We note also that perplexity is parametrization-variant and cannot be used to directly evaluate between pose and graph representations.

Table 4. **Perplexity.** We compute perplexity for each of the graph-backed model sizes on our held-out 512 samples. PT models are evaluated without a prompt. We ablate both the effect of our training stages (Tab. 4a) and the amount of PT/SFT training data (Tab. 4b).

(a) **Training-Stage Ablation**

Model	PT (uncond.)	PT + SFT (cond.)	No-PT + SFT
0.6b	1.331	1.298	1.343
1.7b	1.324	1.290	1.326
4b	1.307	1.274	1.311
8b	1.305	1.273	1.303
14b	1.300	1.266	1.298

(b) **Dataset-Size Ablation (14b)**

PT\SFT	Full	Half	Quarter	None
Full	1.266	1.279	1.288	1.300
Half	1.273	1.284	1.296	1.318
Quarter	1.276	1.292	1.305	1.361

6. Discussion and Limitations

As the result of our proposed large-scale object dataset and graph-backed parametrization, our model is able to learn to model sequences and connectivity fairly well. However, it struggles to produce long sequences without introducing inter-part collisions. Future work should consider both how inference-time approaches might be leveraged to guide decoding and how to improve the spatial understanding of the model itself.

While our graph representation is compact and fairly general, and may have broader application, it requires the model to have knowledge of the domain. If not for the pretraining, the model would not know the positions of the connectors on the parts, and therefore would then not have the ability to use them. In a closed setting such as this where the part vocabulary is static, this can be learned, but it poses challenges for generalization to new domains.

In order to maintain our focus on data and representation, our supervised fine-tuning evaluation was made deliberately straightforward. There is likely strong potential for improvement through post-training techniques.

As a simplification, and due partially to computational constraints, we limited build-sequence length to 100 parts. However, real-world sets carry no such limitation. Future work should consider how scaling might be achieved.

While we demonstrated that unconditional pretraining improves model generalization and sample efficiency for the task of text-conditioned generation, we anticipate that the learned prior may be adapted to additional downstream tasks,

1 This is a LEGO model of a stylized, light-green bamboo stalk with leaves, constructed from cylindrical bricks and leaf elements, mounted on a circular white base. **2** This LEGO model depicts a small, gray stone shrine or altar, featuring an archway topped with two torches, a central pedestal holding a red gem, and various accessories including a green plant, a white urn, and a black fixture. **3** This is a LEGO model of a white and grey handheld device, possibly a scanner or medical tool, featuring a blue transparent element and a silver cylindrical piece at its front. **4** This is a LEGO model of a traditional black and gold rickshaw, featuring two large spoked wheels, a covered passenger seat with beige upholstery, and long pulling handles at the front. **5** This is a LEGO model of a rectangular, dark red gift box, decorated with a lighter red ribbon and bow on top, and featuring a black winding key on its side, indicating it is a music box. **6** This LEGO minifigure features vibrant red hair, a white torso printed with a silver scale-mail vest, and solid blue legs. **7** This is a blue LEGO sports car model, featuring a prominent rear spoiler, grey wheels, and transparent windows, shown from multiple angles. **8** This is a detailed LEGO model of a black espresso machine, featuring a transparent bean hopper on top, a side-mounted water reservoir, a front-facing portafilter and steam wand, and a pressure gauge. **9** This LEGO model features a minifigure in a red torso and Santa hat, holding a large, stylized green Christmas tree made of stacked, angled pieces. **10** This LEGO model is a rectangular patch of dark green grass, constructed from numerous plant stem pieces on a black plate with a brown base, featuring a black Technic axle protruding from one end for connection. **11** This is a LEGO model of a large, cylindrical purple container or barrel, reinforced with tan bands and featuring various grey mechanical attachments, including handles, levers, and what appear to be nozzles or connectors. **12** This is a simple, abstract LEGO vehicle constructed from primary-colored bricks—blue, red, yellow, and black—and equipped with four red wheels. **13** This LEGO model is a colorful, low-profile bed constructed from a yellow base and a top layer of multicolored rectangular bricks, featuring a light blue wedge piece at one end to serve as a headboard. **14** This is a 3D digital model of a stylized, blocky LEGO vehicle, resembling a futuristic truck or armored van, shown from multiple angles to display its gray, white, and light blue construction with prominent black wheels. **15** This is a LEGO model of a black squirrel, constructed with dark grey and light grey bricks, featuring a large bushy tail, prominent white eyes, and holding two small, round objects in its paws. **16** This LEGO model depicts a dilapidated, single-room shelter or bunker with crumbling brown and tan walls, containing a simple bed, a rifle, a sack, and a small cache of money.

Figure 9. **Text-Conditioned Samples.** Samples produced using prompts from the evaluation set. Outputs are arranged as pairs with a prompt number. Within pairs, our outputs are above and those of BrickGPT [27] are beneath. Match the number to the full prompt below.

including reconstruction or editing. Additionally, our model, in autoregressively sampling full sequences, was effectively allowed an infinite library of parts. Future work should also explore generation or assembly conditioned on a specified part set or under additional constraints.

7. Conclusion

Our investigation represents the first attempt to model and autoregressively generate LEGO-brick structures using pieces with arbitrary connectivity.

To achieve this, we first proposed BrickNet, a large-scale dataset of human-designed digital LDraw structures consisting of millions of placed pieces. Second, we introduced a low-dimensional parametrization of brick structure and demonstrated improved parsing validity across sequence lengths and model-size variants. To make this possible, we annotated the LDraw part library with a typed connector system with precise positions. Finally, in adapting our BrickNet-PT model to the task of text-conditioned generation, we demonstrated that our learned prior is adaptable. We hope our dataset will enable broader task application.

Acknowledgments. This work was funded in part by the French government under management of Agence Nationale de la Recherche as part of the "France 2030" program, reference ANR-23-IACL-0008 (PR[AI]RIE-PSAI project) and the ANR project VideoPredict ANR-21-FAI1-0002-01. Cordelia Schmid would like to acknowledge the support by the Körber European Science Prize. We are grateful to Michael Black for his encouragement and support.

References

[1] Seokjun Ahn, Jungtaek Kim, Minsu Cho, and Jaesik Park. Budget-aware sequential brick assembly with efficient constraint satisfaction. *TMLR*, 2024. 2, 3, 6, 7

[2] Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, Jakob Engel, Edward Miller, Richard Newcombe, and Vasileios Balntas. SceneScript: Reconstructing scenes with an autoregressive structured language model. In *ECCV*, 2024. 2

[3] Jamie Berard. Stressing the elements, 2006. 3

[4] Daniel Bolya, Po-Yao Huang, Peize Sun, Jang Hyun Cho, Andrea Madotto, Chen Wei, Tengyu Ma, Jiale Zhi, Jathushan Rajasegaran, Hanoona Abdul Rasheed, Junke Wang, Marco Monteiro, Hu Xu, Shiyu Dong, Nikhila Ravi, Shang-Wen Li, Piotr Dollar, and Christoph Feichtenhofer. Perception encoder: The best visual embeddings are not at the output of the network. In *NeurIPS*, 2025. 7

[5] Hongyi Cao, Gang Xu, Renshu Gu, Jinlan Xu, Xiaoyu Zhang, and Timon Rabczuk. A parallel feature-preserving mesh variable offsetting method with dynamic programming, 2023. 4

[6] Hyunsoo Chung, Jungtaek Kim, Boris Knyazev, Jinhwi Lee, Graham W Taylor, Jaesik Park, and Minsu Cho. Brick-by-brick: Combinatorial construction with deep reinforcement learning. In *NeurIPS*, pages 5745–5757. Curran Associates, Inc., 2021. 2

[7] Gheorghe Comanici et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. 4

[8] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. InverseCSG: automatic conversion of 3d models to CSG trees. *ACM TOG*, 37(6), 2018. 2

[9] Mohammed Munzer Dwedari, Matthias Niessner, and Zhenyu Chen. Generating context-aware natural answers for questions in 3d scenes. In *BMVC*. BMVA, 2023. 2

[10] Jiahao Ge, Mingjun Zhou, and Chi-Wing Fu. Learn to create simple LEGO micro buildings. *ACM TOG*, 43(6), 2024. 2

[11] Seyed Kamyar Seyed Ghasemipour, Satoshi Kataoka, Byron David, Daniel Freeman, Shixiang Shane Gu, and Igor Mordatch. Blocks assemble! learning to assemble with large-scale structured reinforcement learning. In *ICML*, pages 7435–7469. PMLR, 2022. 2

[12] Mengqi Guo, Chen Li, Yuyang Zhao, and Gim Hee Lee. TreeSBA: Tree-transformer for self-supervised sequential brick assembly. In *ECCV*, pages 35–51, Cham, 2025. Springer Nature Switzerland. 2

[13] Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3D-LLM: Injecting the 3D world into large language models. In *NeurIPS*, pages 20482–20494. Curran Associates, Inc., 2023. 2

[14] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. SceneCraft: An LLM agent for synthesizing 3d scenes as blender code. In *ICML*, 2024. 2

[15] James Jessiman. LDraw, 1995–2026. 1, 3, 5

[16] R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. ShapeAssembly: learning to generate programs for 3D shape structure synthesis. *ACM TOG*, 39(6), 2020. 1, 2

[17] Jungtaek Kim, Hyunsoo Chung, Jinhwi Lee, Minsu Cho, and Jaesik Park. Combinatorial 3D shape generation via sequential assembly. In *NeurIPS Workshop on Machine Learning for Engineering Modeling, Simulation, and Design (ML4Eng)*, 2020. 2

[18] Milin Kodnongbua, Benjamin T. Jones, Maaz Bin Safeer Ahmad, Vladimir G. Kim, and Adriana Schulz. ReparamCAD: Zero-shot CAD re-parameterization for interactive manipulation. *SIGGRAPH Asia*, 2023. 2

[19] Peter Kulits, Haiwen Feng, Weiyang Liu, Victoria Fernandez Abrevaya, and Michael J. Black. Re-thinking inverse graphics with large language models. *TMLR*, 2024. 2

[20] Peter Kulits, Michael J. Black, and Silvia Zuffi. Reconstructing animals and the wild. In *CVPR*, pages 16565–16577, 2025. 2

[21] Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *ECCV*, pages 366–384, Cham, 2025. Springer Nature Switzerland. 7

[22] Jiaxi Lv, Yi Huang, Mingfu Yan, Jiancheng Huang, Jianzhuang Liu, Yifan Liu, Yafei Wen, Xiaoxin Chen, and Shifeng Chen. GPT4Motion: Scripting physical motions in text-to-video generation via blender-oriented GPT planning. In *CVPRW*, pages 1430–1440, 2024. 2

[23] Khaled Mamou, E. Lengyel, and A. Peters. Volumetric hierarchical approximate convex decomposition. *Game Engine Gems*, 3:141–158, 2016. 4

[24] Roland Melkert. LDCad: LDraw CAD editor, 2026. 3

[25] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy J. Mitra, and Leonidas J. Guibas. StructureNet: Hierarchical graph networks for 3D shape generation. *ACM TOG*, 38(6), 2019. 1, 2

[26] Maxim Peysakhov and William C. Regli. Using assembly representations to enable evolutionary design of LEGO structures. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(2):155–168, 2003. 2

[27] Ava Pun, Kangle Deng, Ruixuan Liu, Deva Ramanan, Changliu Liu, and Jun-Yan Zhu. Generating physically stable and buildable brick structures from text. In *ICCV*, pages 14798–14809, 2025. 2, 3, 5, 7, 8

[28] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural shape parser for constructive solid geometry. In *CVPR*, pages 5515–5523, 2018. 1, 2

[29] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3D-GPT: Procedural 3D modeling with large language models. In *3DV*, pages 1253–1263, 2025. 2

[30] Qwen Team. Qwen2.5-VL, 2025. 7

[31] Rylee Thompson, Ghalebi Elahe, Terrance DeVries, and Graham W. Taylor. Building LEGO using deep generative models

of graphs. *Machine Learning for Engineering Modeling, Simulation, and Design Workshop at NeurIPS*, 2020. 2, 3

[32] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. SigLIP 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features, 2025. 7

[33] Aaron Walsman, Muru Zhang, Klemen Kotar, Karthik Desingh, Ali Farhadi, and Dieter Fox. Break and make: Interactive structural understanding using LEGO bricks. In *ECCV*, pages 90–107, Cham, 2022. Springer Nature Switzerland. 2, 3, 5

[34] Aaron Walsman, Muru Zhang, Adam Fishman, Ali Farhadi, and Dieter Fox. Learning to build by building your own instructions. In *ECCV*, pages 261–278, Cham, 2025. Springer Nature Switzerland. 2, 3

[35] Ruocheng Wang, Yunzhi Zhang, Jiayuan Mao, Chin-Yi Cheng, and Jiajun Wu. Translating a visual LEGO manual to a machine-executable plan. In *ECCV*, pages 677–694. Springer, 2022. 2

[36] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 7

[37] Le Xue, Mingfei Gao, Chen Xing, Roberto Martín-Martín, Jiajun Wu, Caiming Xiong, Ran Xu, Juan Carlos Niebles, and Silvio Savarese. ULIP: Learning a unified representation of language, images, and point clouds for 3D understanding. In *CVPR*, pages 1179–1189, 2023. 2

[38] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. 6

[39] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3d embodied ai environments. In *CVPR*, pages 16227–16237, 2024. 2

[40] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. In *ICLR*, 2024. 2