Appendix

A. BrickGPT Implementation Details

Captioning Details. The complete prompt template used for GPT-4o caption generation is as follows:

"This is a rendering of a 3D object built with LEGO bricks with 24 different views. The object belongs to the category of {CATEGORY_NAME}. You will generate five different captions for this {CATEGORY_NAME} that:

1. Describes the core object/subject and its key geometric features
2. Focuses on structure, geometry, and layout information
3. Uses confident, concrete, and declarative language
4. Omits color and texture information
5. Excludes medium-related terms (model, render, design)
6. Do not describe or reference each view individually.
7. Focus on form over function. Describe the physical appearance of components rather than their purpose.
8. Describe components in detail, including size, shape, and position relative to other components.
9. The five captions should be from coarse to fine, with the first one being the most coarse-grained (e.g., a general description of the object, within 10 words) and the last one being the most fine-grained (e.g., a detailed description of the object, within 50 words). The five captions should be different from each other. Do not include any ordering numbers (e.g., 1, a, etc.).
10. Describe the object using the category name "{CATEGORY_NAME}" or synonyms of the category name "{CATEGORY_NAME}"."

StableText2Brick Details. We generate StableText2Brick starting from the objects in ShapeNetCore [2]. While ShapeNetCore provides both voxel and mesh representations, we find that working directly with mesh data better preserves geometric details. We voxelize the 3D mesh into a $20 \times 20 \times 20$ grid representation and generate its brick structure using a delete-and-rebuild algorithm. This algorithm is similar to that studied in prior work [46]. However, instead of initializing the structure by filling its voxels with 1×1 unit bricks and randomly merging them into larger bricks, we greedily place bricks to fill the voxels layer-by-layer from bottom to top. We use eight commonly available standard bricks: 1×1, 1×2, 1×4, 1×6, 1×8, 2×2, 2×4, and 2×6. We prioritize placing 1) bricks that are only partially supported by bricks on the layer below, 2) bricks that touch multiple bricks on the layer below, 3) large bricks, and 4) bricks of the opposite orientation from bricks on the layer below.

After initializing the structure, we apply the new stability analysis algorithm [38] to iteratively identify weak regions, delete their bricks, and rebuild them by greedily placing bricks, prioritizing 1) bricks that connect multiple disconnected components and 2) large bricks. This process is stochastic; we choose weak regions randomly as per [46], and we break ties between two bricks of the same heuristic value randomly. Using this delete-and-rebuild algorithm, we generate two different structures for each object.

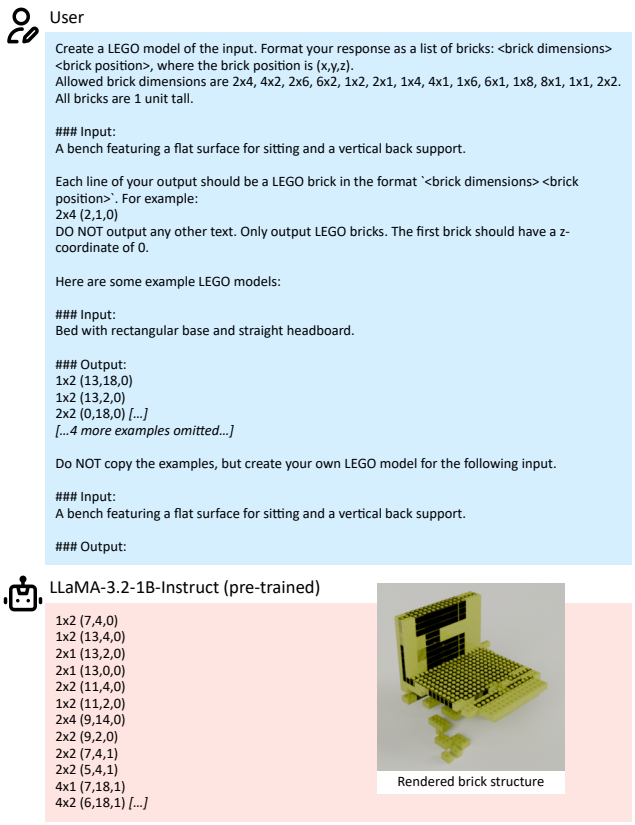

Figure 8. **Few-shot Learning.** Given a prompt and several example structures in text format, a pre-trained LLaMA model can generate brick designs with some structure.

With our delete-and-rebuild shape-to-brick algorithm, we generate $62{,}000+$ brick structures covering the 21 categories with $31{,}218$ unique 3D objects from ShapeNetCore. Among the selected 3D objects, $\sim$92% (i.e., $28{,}822$) of them have at least one stable brick design, which offer $47{,}000+$ stable layouts. Our StableText2Brick dataset significantly expands upon the previous StableLego dataset [38] in several key aspects: it contains $> 3\times$ more stable unique 3D objects with $> 5\times$ more stable structures compared to $8{,}000+$ in StableLego, spans a diverse set of 21 object categories, and provides detailed geometric descriptions for each shape.

Training Details. The full prompt that we use to construct our instruction fine-tuning dataset is as follows:

"*[SYSTEM]*You are a helpful assistant.

*[USER]*Create a LEGO model of the input. Format your response as a list of bricks: <brick dimensions> <brick position>, where the brick position is (x,y,z).

Allowed brick dimensions are 1×1, 1×2, 2×1, 1×4,

4×1, 1×6, 6×1, 1×8, 8×1, 2×2, 2×4, 4×2, 2×6, 6×2. All bricks are 1 unit tall.

Input:

[INSERT CAPTION]"

We fine-tune using low-rank adaptation (LoRA) [23] with a rank of 32, alpha of 16, and dropout rate of 0.05. To prevent catastrophic forgetting and training instability, we apply LoRA to only the query and value matrices for a total of 3.4M tunable parameters. We train for three epochs on eight NVIDIA RTX A6000 GPUs. We use the AdamW optimizer [44] with a learning rate of 0.002, using a cosine scheduler with 100 warmup steps and a global batch size of 64. The total training time is 12 hours.

Inference Details. The sampling temperature is set to 0.6 for all experiments. During brick-by-brick rejection sampling, to mitigate repeated generation of rejected bricks, we increase the temperature by 0.01 each time the model generates a brick that has already been rejected.

To force each output brick to be in the format "{h}×{w} ({x},{y},{z})", we sample only from the set of valid tokens during each step. For example, the first token must be a digit, the second an '×', and so on. This has little to no effect on the output of BRICKGPT, but helps force the baseline pre-trained models to output bricks in the specified format.

In our experiments, only 1.2% of the generated designs exceed the maximum number of rollbacks and fail to produce a stable final structure.

Novelty Analysis. For each generated structure, we find its closest structure in the training dataset, measured by computing the Chamfer distance in voxel space. As seen in Figure 9, the generated structures are distinct from their nearest neighbors in the dataset, confirming that our model can create novel designs rather than simply memorizing the training data.

In-context Learning. We use LLaMA-3.2-1B-Instruct [11] as our base model, chosen for its computational efficiency. The in-context learning pipeline is shown in Figure 8, where the base model can generate brick structures through in-context learning, while suffering from collisions and disconnectivity. We do not use rejection sampling or rollback when evaluating zero-shot or few-shot generation, as doing so results in an excessive number of rejections and a sharp increase in generation time.

B. Robotic Assembly

We demonstrate automated assembly using a dual-robot-arm system as shown in Figure 10. The system consists of two Yaskawa GP4 robots, each equipped with an ATI force-torque sensor. A calibrated baseplate is placed between them, and the robots use the bricks initially placed on the plate to construct the brick structure. Given a brick structure $B = [b_1, b_2, \ldots, b_N]$, we employ the action mask in [40] with assembly-by-disassembly search [73] to generate a physically executable assembly sequence for the robots, i.e., reordering the brick sequence so that 1) each intermediate structure is physically stable by itself, 2) each intermediate structure is stable under the robot operation, and 3) each assembly step is executable within the system's dexterity. With the executable assembly sequence, an asynchronous planner [24] distributes the assembly tasks to the robots, plans the robots' movements, and coordinates the bimanual system to construct the brick structure. The robots use the end-of-arm tool and the manipulation policy presented in [24, 39] with closed-loop force control to robustly manipulate bricks and construct the structure.

C. Manual Assembly

As shown in Figure 11, human users can assemble our generated structures, demonstrating their physical stability and buildability. Notably, since our method outputs a sequence of intermediate steps, it naturally serves as an intuitive assembly guide.

D. Limitations

Though our method outperforms existing methods, it still has several limitations. First, due to limited computational resources, we have not explored the largest 3D dataset. As a result, our method is restricted to producing designs within a $20 \times 20 \times 20$ grid across 21 categories, while recent 3D generation methods can create a wider variety of objects. Future work includes scaling up model training at higher grid resolutions on larger, more diverse datasets, such as Objaverse-XL [7]. Training on large-scale datasets can also improve generalization to out-of-distribution text prompts.

Second, our method currently supports a fixed set of commonly used toy bricks. In future work, we plan to expand the brick library to include a broader range of dimensions and brick types, such as slopes and tiles, allowing for more diverse and intricate designs.

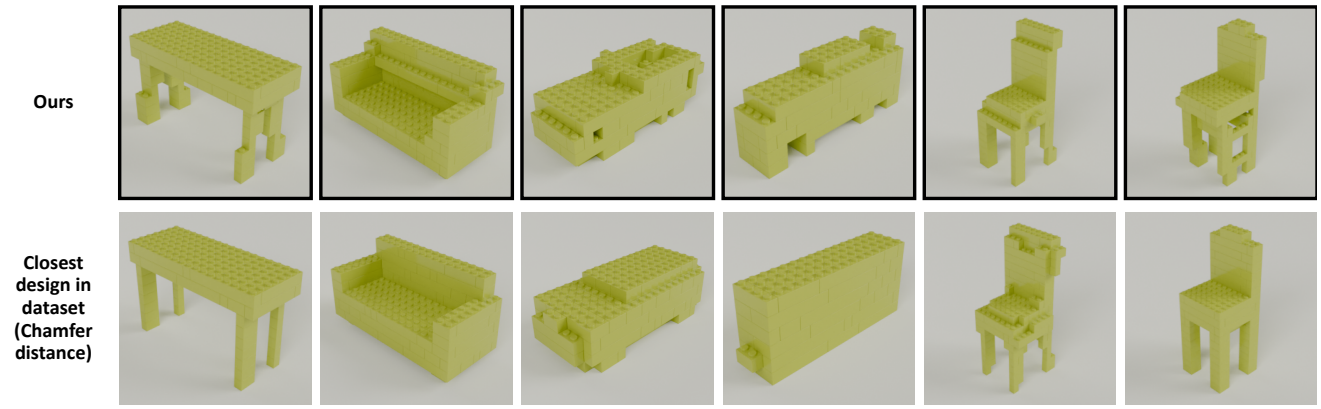

Figure 9. **Novelty Analysis.** For each structure generated by BRICKGPT, we find the closest structure in the training dataset as measured by Chamfer distance in voxel space. The generated structures are distinct from their nearest neighbors, indicating low memorization.

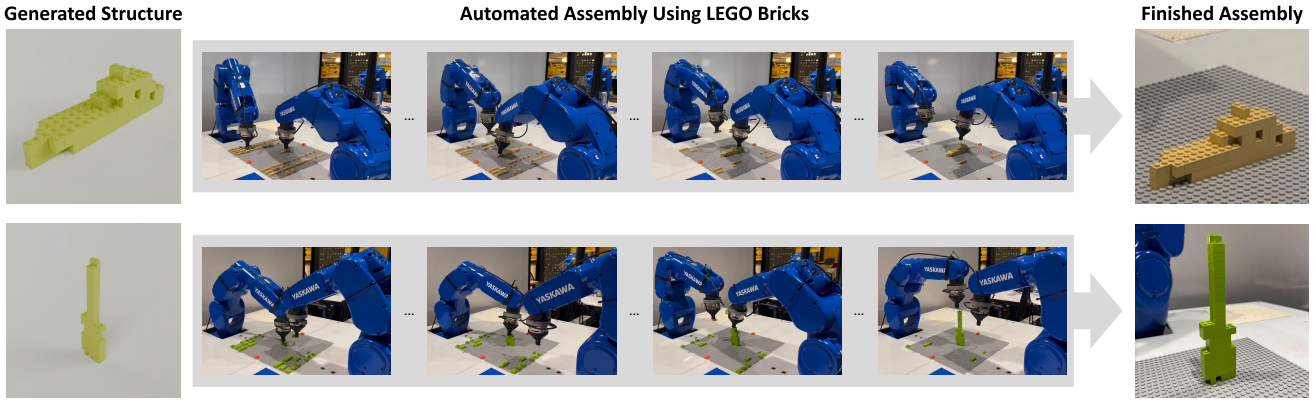

Figure 10. **Automated Assembly.** We demonstrate robotic assembly of generated structures using LEGO bricks.

Figure 11. **Manual Assembly.** We demonstrate manual assembly of generated structures using LEGO bricks.