Supplementary Material:
Order-Aware Generative Modeling Using the *3D-Craft* Dataset

The supplementary material is organized as follows:

- In Section A, we provide a more detailed overview of *3D-Craft*, and demonstrate the quality and diversity of our dataset in various aspects.
- We display more qualitative results of the sequential behavior of VoxelCNN in Section B. We present typical results sampled from our model, including both successful and failure cases.
- In Section C, we provide additional ablative analysis for our VoxelCNN model, studying how different module designs influence the performance of our algorithm.
- In Section D, we detail the model architectures of baseline LSTM and 3D-PixelCNN.
- We also provide some sample videos of *3D-Craft*, as well as how the order transferred on the real-world ShapeNet benchmark.

A. More About *3D-Craft* Dataset

A.1. Videos of Human Order

In the supplemental material, we include a few video recordings of *3D-Craft* building processes. We hope interested readers could find more intuition of how houses are crafted in "human-order".

A.2. Block Types in Minecraft

In Figure 1, we show some popular block types used frequently by human players in the game.

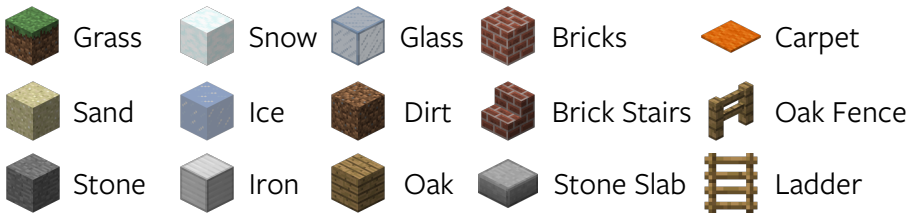


Figure 1: Some popular block types human players used in *3D-Craft*.

A.3. Houses in *3D-Craft*

A.3.1 Houses by Sizes

In Figure 2, we show some houses of different sizes.

A.3.2 Houses by Complexities

Our dataset contains houses with different complexities, as shown in Figure 3 (row 1, row 2). Notably, simple houses (row 1) are simply built from one or two block types and have much more regular shapes, while more complicated houses (row 2) can consist of more than 30 different block types with delicate interior designs (row 3).

Figure 2: Houses by sizes: small ($<$ 500 actions), medium (500-1000 actions), large ($>$ 1000 actions)

Figure 3: Houses by complexity: simple houses (row 1), complex houses (row 2) and examples of complicated indoor designs (row 3).

A.3.3 Additional Examples of Houses

Figure 4: Additional example of houses from the dataset.

B. Qualitative Results

We display more qualitative results demonstrating the sequential behavior of VoxelCNN. For a given house in the test set, we start from 50% complete and let our model predict for another 200 steps without interruption (even harder than the setup in our main paper with 50 steps). We show some results in Figure 5 and failure cases in Figure 6.

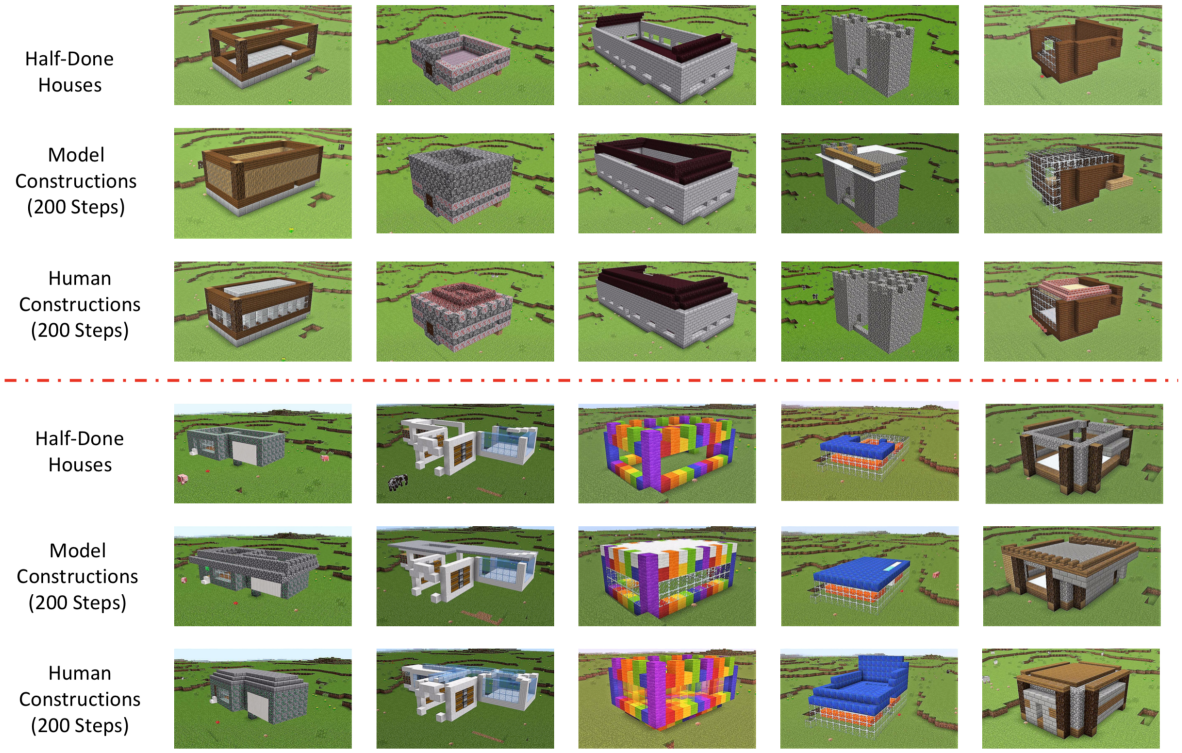


Figure 5: Sample results generated by our best model. *1st, 4th rows:* houses with 50% blocks placed; *2nd, 5th rows:* constructions from our model with 200 newly generated blocks; *3rd, 6th rows:* constructions from ground truth data with next 200 blocks.

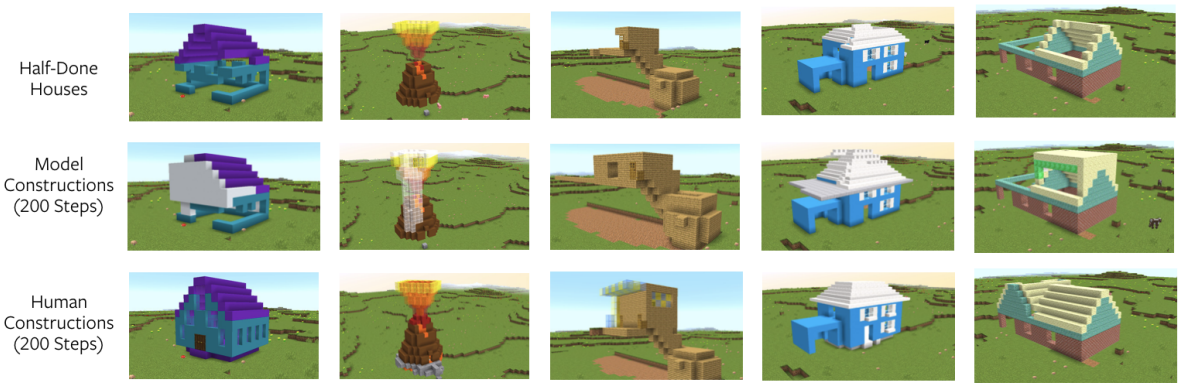


Figure 6: Sample failure cases generated by our best model: our model sometimes adds redundant repetitive structures such as a too large roof or a too long wall.

C. Additional Ablation Studies

Methods	ACC@1 (%)	ACC@5 (%)	ACC@10 (%)	CCA	MTC	Normalized MTC Rate(%)	Perplexity
VoxelCNN (human order, human input)	62.7	77.2	78.9	11.7	122.8	23.2	3.24
w/o temporal information	60.8	75.2	76.9	11.1	136.3	25.5	3.57
w/o global embedding	61.8	76.1	77.9	11.3	128.2	24.1	3.31
w/o factorized prediction	62.4	77.0	78.8	11.9	123.7	23.3	3.34

Table 1: Ablation studies of different modules in the proposed VoxelCNN.

To have a more comprehensive understanding of our VoxelCNN model, we conducted several ablation experiments.

C.1. Ablation on Separate Modules

Our first ablation study is on how each module influenced the results as displayed in Table 1.

State History. We study the influence of the temporal information in the state history by removing s_{t-1}, s_{t-2} from the input and making a Markovian prediction a_{t+1} from current state s_t. As we can see in the table, the accuracy (e.g. ACC@1) drops from 62.7% to 60.8% and the normalized MTC goes up from 23.2% to 25.5%. This is consistent with our intuition that the order does play a key role in predicting future generative behavior.

Global Encoding. We also study how the context of different resolutions helps in our model. We remove the global encoding so the model makes prediction of the next building action, only using local $7 \times 7 \times 7$ context. The performance of this single-stream network is inferior compared to our two-stream model. The accuracy ACC @ 1 drops from 62.7% to 61.8% , and the error (Normalized MTC) increases from 23.2% to 24.1%. This demonstrates that global context helps make prediction of generative actions by capturing information from a larger receptive field.

Factorized Prediction. We study how the factorized prediction module helps in our model. We remove the dependency between λ_{t+1} (the "where") and b_{t+1} (the "what") in our VoxelCNN:

$$p(\lambda_{t+1}, b_{t+1}|a_{1:t}) = p(\lambda_{t+1}|a_{1:t})p(b_{t+1}|a_{1:t})$$

i.e., we predict the block type regardless of where we will place it. The accuracy (ACC @1, @5, @10) drops by $.1\%$ and perplexity goes up from 3.24 to 3.34. The drop in the metrics is not substantial. We attribute it to the fact that most build actions are repetitive with similar block-types in a local neighborhood.

C.2. Ablation on Hyper-parameters

Block Type Embedding. As an auxiliary experiment, we add an embedding layer for block types, since one-hot encoded vectors are high-dimensional and sparse. In Table 2, we compare our vanilla model (without embedding layer) with models

with embedding layers of different sizes. We can observe that VoxelCNN with embedding dimension 8 and 10 slightly outperform other models.

Methods	ACC@1 (%)	ACC@5 (%)	ACC@10 (%)	CCA	MTC	Normalized MTC Rate(%)	Perplexity
One-hot (Vanilla)	62.7	77.2	78.9	11.7	122.8	23.2	3.24
Embedding (6)	63.7	78.0	79.8	12.6	117.5	22.3	3.12
Embedding (8)	63.9	78.4	80.2	12.0	115.3	21.8	3.06
Embedding (10)	64.0	78.2	80.0	12.3	116.7	22.1	3.07
Embedding (12)	63.4	77.5	79.3	11.8	120.0	22.7	3.17

Table 2: Ablation studies of adding an embedding layer in VoxelCNN: we compare the performance of the proposed VoxelCNN using different embedding dimensions (6, 8, 10, 12) with the Vanilla model (no embedding).

Feature Dimension. In Table 3, we conducted ablation studies on how different CNN-feature dimensions influence the performance of VoxelCNN. Our observation is that any choice in a reasonable range will produce similar good results on all metrics.

Methods	ACC@1 (%)	ACC@5 (%)	ACC@10 (%)	CCA	MTC	Normalized MTC Rate(%)	Perplexity
16 (Vanilla)	62.7	77.2	78.9	11.7	122.8	23.2	3.24
8	61.0	75.6	77.4	10.8	130.1	24.5	3.61
32	63.0	77.5	79.2	12.5	121.4	22.9	3.14
64	63.1	77.7	79.5	12.8	120.5	22.7	3.15

Table 3: Ablation studies of different feature dimension in CNN Module in our VoxelCNN

Optimization Hyper-Parameter. In Table 4 and 5, we analyzed how different batch sizes and learning rates affect the performance of our model, and found the effect of these hyper-parameters are minimal.

Methods	ACC@1 (%)	ACC@5 (%)	ACC@10 (%)	CCA	MTC	Normalized MTC Rate(%)	Perplexity
64 (Vanilla)	62.7	77.2	78.9	11.7	122.8	23.2	3.24
16	62.7	77.3	79.1	11.8	122.0	22.9	3.29
32	62.8	77.2	79.0	12.0	122.2	23.0	3.21
128	62.5	77.2	79.0	11.8	122.1	23.0	3.27

Table 4: Ablation studies of different batch-size.

Methods	ACC@1 (%)	ACC@5 (%)	ACC@10 (%)	CCA	MTC	Normalized MTC Rate(%)	Perplexity
1e-2 (Vanilla)	62.7	77.2	78.9	11.7	122.8	23.2	3.24
1e-1	62.4	77.0	78.9	11.7	124.2	23.4	3.27
1e-3	62.5	77.0	78.8	11.5	123.1	23.2	3.29

Table 5: Ablation studies of different learning rate.

All the results in Table 3, 4, 5 suggest that the performance of our VoxelCNN model is resilient to both feature dimensions

and other hyper-parameters. In conjunction with Table 1 from our main paper, it further enhanced our belief that it is the modeling of human-ordering that plays the key role of making the generative process easier and more reliable.

D. Baseline Architectures

D.1. LSTM

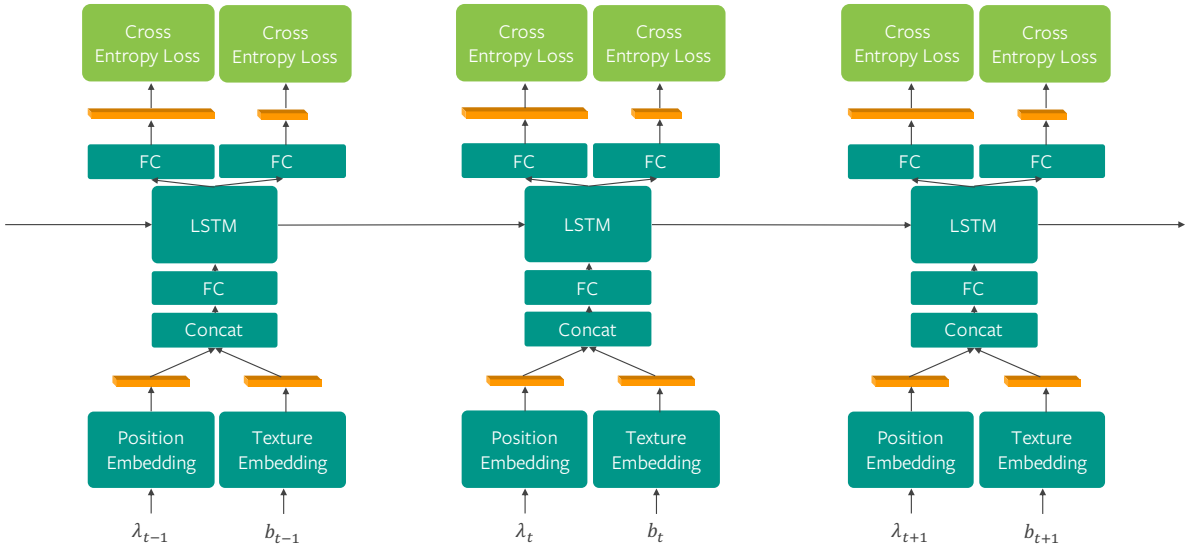
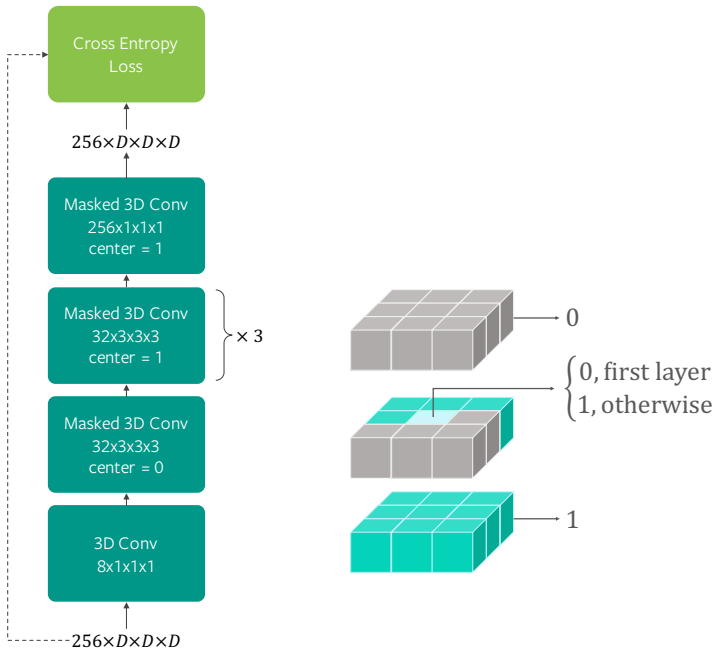


Figure 7: Baseline LSTM architecture. We normalize all the houses into a $32 \times 32 \times 32$ space. Considering the starting and the end as special actions, the input at each frame consists of position ID $\lambda_t \in \{0, 1, \ldots, 32^3 + 1\}$, and texture ID $b_t \in \{0, 1, \ldots, 257\}$. Each ID is embedded into a vector space of 256-d, then concatenated, passing through a fully connected (FC) layer, and fed into an one-layer LSTM unit, which also has 256-d hidden units. The output feature from LSTM is fed into a $(32^3 + 2)$-way classifier for position, and a 258-way classifier for texture. During training, cross-entropy losses are computed baesd on the classifier outputs.

D.2. 3D-PixelCNN



(a) 3D-PixelCNN. (b) Masked 3D Convolution Layer

Figure 8: (a) We normalize all the houses into a $D \times D \times D$ space, where $D = 32$ in our case. Considering 256 different materials, the input tensor to 3D-PixelCNN is of size $256 \times D \times D \times D$. It is embedded into lower dimensional (8-d) space by a 3D convolution layer, then followed by 5 masked 3D convolution layers. The center of the filter kernel of the first masked 3D convolution layer is zero, to ensure the output will only depend on the previous voxels in raster-scan order. The last masked 3D convolution layer has 256 output channels to predict the materials for each voxel. Cross-entropy loss is used during training. (b) A demonstration of a masked 3D convolution layer, which has kernel size $3 \times 3 \times 3$. The weights of the gray kernel elements are fixed to be zero. Therefore, the output voxel will only condition on the previous voxels in raster-scan order.