UNSUPERVISED PROGRAM SYNTHESIS FOR IMAGES BY SAMPLING WITHOUT REPLACEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Program synthesis has recently emerged as a promising approach to the image parsing task. Most prior work has followed a two-step pipeline that involves supervised pretraining of a sequence-to-sequence model on synthetic programs, followed by reinforcement learning to fine-tune the model on real reference images. A purely RL-driven approach without supervised pretraining has never proven successful, since useful programs are too sparse in the program space to sample from. In this paper, we present the first purely unsupervised learning algorithm that parses constructive solid geometry (CSG) images into context-free grammar (CFG) without pretraining. The key ingredients of our approach include entropy regularization and sampling without replacement from the CFG syntax tree, both of which encourage exploration of the search space, and a grammar-encoded tree LSTM that enforces valid outputs. We demonstrate that our unsupervised method can generalize better than supervised training on a synthetic 2D CSG dataset. Additionally, we find that training in this way naturally optimizes the quality of the top-k programs, leading our model to outperform existing method on a 2D computer aided design (CAD) dataset under beam search.

1 INTRODUCTION

Image generation has been an extensively studied topic in machine learning and computer vision. Vast numbers of papers have explored generating images through low-dimensional latent representations (Goodfellow et al., 2014; Arjovsky et al., 2017; Li et al., 2017; Kingma & Welling, 2013; van den Oord et al., 2017; Oord et al., 2016). However, it is challenging to learn disentangled representations which allows us to better control the generative models (Higgins et al., 2017; Kim & Mnih, 2018; Locatello et al., 2018; Chen et al., 2016). In this paper, we will explore generating CFG programs from constructive solid geometry (CSG) images (Hubbard, 1990). We can consider these programs as an alternative to the low-dimensional representation of the image. We can view the model for extracting the programs as an encoder and the renderer that reconstructs the image as a decoder. Parsing an image of geometric shapes into programs enables us to manipulate only the desired components of the image while reconstruct the rest. In this paper, we assume access to a non-differentiable renderer. They are more common than differentiable ones, but more challenging to work with neural network because the gradient w.r.t. the input is inaccessible.

The standard pipeline to parse an image (e.g. CSG image) into programs (e.g. CFG programs) with non-differentiable renderer is supervised pretraining followed by REINFORCE fine-tuning (Sharma et al., 2018; Ellis et al., 2019). Sampling programs directly from a specified grammar provides data sufficient for supervision. However, Bunel et al. (2018) points out a limitation of supervised training with MLE – it maximizes the likelihood of a single reference program while penalizing many other valid programs. This observation is summarized as *program aliasing*. Ellis et al. (2019) incorporated REINFORCE to learn a value function in order to prune unpromising partial programs evaluated in the image space. Sharma et al. (2018) utilizes REINFORCE fine-tuning to accustom the program synthesizer to *approximate* CAD images not generated by grammar. While supervised methods optimize the objective in the string space, RL methods directly optimizes in the image space. Despite these benefits, a pure RL approach was not preferred because correct programs are too sparse in the program space to be sampled frequently enough to learn. We focus on improving the sample efficiency of the REINFORCE algorithm and show that a pure RL approach can achieve comparable result as a two-step model. We further demonstrate that our method generalizes better on a synthetic

2D CSG dataset than a supervised method and yields superior results under beam search on a CAD dataset than a supervised pretrained RL fine-tuned model.

Here are the key components to successfully learn to parse an image *without program supervision*:

- We use **REINFORCE** (Williams, 1992) as our main building component because direct gradient information from the *non-differentiable* renderer is inaccessible.
- We incorporate a **grammar-encoded tree LSTM** to impose a structure on the search space such that the algorithm is essentially sampling a path in the CFG syntax tree top-down. This guarantees the validity of the output program.
- We propose an **entropy estimator** suitable for sampling top-down from a syntax tree for entropy regularization to encourage exploration of the search space.
- Instead of using simple Monte Carlo sampling, we perform **sampling without replacement** from the syntax tree to optimize over top-k programs simultaneously. This leads to faster convergence during training and more effective beam search during testing.

2 Related Work

Our work is related to program synthesis, vision-as-inverse-graphics, as well as generating images with stroke-based rendering (SBR) systems.

Program synthesis has been a growing interest to researchers in machine learning. Supervised training is a natural choice when it comes to input/output program synthesis problems (Parisotto et al., 2016; Chen et al., 2018a; Devlin et al., 2017; Yin & Neubig, 2017; Balog et al., 2017; Zohar & Wolf, 2018). Shin et al. (2018) uses the input/output pairs to learn the execution traces. (Bunel et al., 2018) uses RL to address program aliasing, however supervised pretraining is still necessary to reap its benefits. Approaches to ensure valid outputs involve syntax checkers (Bunel et al., 2018) or constructions of abstract syntactic trees (AST) (Parisotto et al., 2016; Yin & Neubig, 2017; Kusner et al., 2017; Chen et al., 2018b). A graph can also model the information flow in a program (Brockschmidt et al., 2018).

Research on converting images to programs are more directly related to our work (Sharma et al., 2018; Ellis et al., 2019; Tian et al., 2019; Ellis et al., 2018; Liu et al., 2018). The setup of our work closely follows Sharma et al. (2018)'s on program synthesis by supervised pretraining before RL fine-tuning to generalize to a CAD dataset. Ellis et al. (2019) pretrains a policy with supervision from generated data and learns a value function by REINFORCE. Both are deployed to prune out unpromising candidates during test time. Their reward function is binary and is not designed for data that is not an exact match of the action space. Our model can *approximate* images not generated with the specified grammar. Tian et al. (2019) incorporates a differentiable renderer into the learning pipeline while we treat our renderer as an external procedure independent from the learning process. We cannot propagate gradient from the reward to the model. Ellis et al. (2018) uses neural network to extract the shapes from hand-drawn sketches, formulates the grammatical rules as constraints and obtains the final program by solving a constraint satisfaction problem. This process can be computationally expensive compared to neural network's performance in test time.

Vision-as-inverse-graphics concerns parsing a scene into a collection of shapes or 3D primitives, e.g. cars or trees, with parameters, e.g. colors or locations, that imitates the original scene (Tulsiani et al., 2017; Romaszko et al., 2017; Wu et al., 2017). Yao et al. (2018) further manipulates the objects de-rendered, such as color changes. Stroke-based rendering creates an image in a way natural to human. Some of the examples are recreating paintings imitating a painter's brush stroke by Huang et al. (2019), drawing sketches of objects by Ha & Eck (2017). SPIRAL by Ganin et al. (2018) is an adversarially trained deep reinforcement learning agent that can recreate MNIST digits and Omniglot characters. Stroke-based rendering behaves in an additive way. The action is usually a line with continuous parameters such as width and length. A grammar structure is unnecessary to both vision-as-inverse-graphics and stroke-based rendering.

3 Problem Definition

We use CSG (Hubbard, 1990) to form an image. The input of our model are images constructed from basic shapes such as square, circle or triangle, each with a designated size and location (see Figure 7). The outputs of the model are CFG programs.

CFG contains terminals and non-terminals. S, T, and P are non-terminals for the start, operations, and shapes. The rest are terminals, e.g. $+$ (union), $*$ (intersection), $-$ (subtraction), and $c(48, 16, 8)$

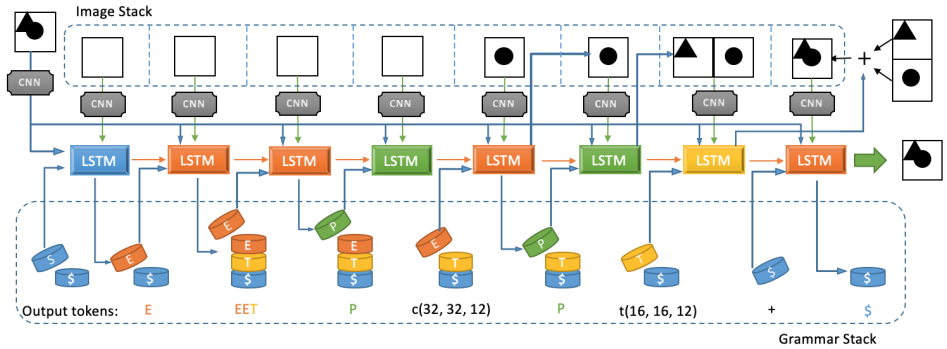


Figure 1: This is an example of grammar encoded tree LSTM at work. The top layer of canvases demonstrates the image stack and the bottom layer demonstrates the grammar stack. The blue, orange, yellow and green colored LSTM cell generates grammatical tokens according to the CFG rule 1, 2, 3 and 4 respectively. In implementation, we can constrain the output space by adding a mask to the output of the LSTM and render the invalid options with close to zero probability of being sampled.

stands for a circle with radius 8 at location $(48, 16)$ in the canvas. Please refer to Figure 3 for some examples of CSG images with their corresponding programs. Each line below is a *production rule* or just *rule* for simplicity:

$$S \to E \tag{1}$$
$$E \to EET|P \tag{2}$$
$$T \to +|-|* \tag{3}$$
$$P \to SHAPE_1|SHAPE_2|\cdots|SHAPE_n. \tag{4}$$

4 Proposed Algorithm

Our model consists of a CNN encoder for canvases, an embedding layer for the actions, and an RNN for generating the program sequences (see Figure 1 for demonstration). The model is trained with entropy regularized REINFORCE (Williams, 1992). Let $\mathcal{H}(s)$ and $f(s)$ be the entropy (we will define this later) and reward function of the sequence s respectively. θ is the parameters of our model. The objective is optimized as follows:

$$\Delta\theta \propto \mathbb{E}_{s\sim P_\theta(s)}[\nabla_\theta \log P(s) f(s)] + \alpha \nabla_\theta \mathcal{H}(s), \tag{5}$$

The output program s is converted to an image $\mathbf{y}$ by a non-differentiable renderer. The image is compared to the target image $\mathbf{x}$ and receives a reward $R(\mathbf{x}, \mathbf{y})$, which is our definition of $f(s)$. We adopt Chamfer distance as part of the reward function as in Sharma et al. (2018). Chamfer distance calculates the average matching distance to the nearest feature. Let $x \in \mathbf{x}$ and $y \in \mathbf{y}$ be pixels in each image respectively. Chamfer distance is described formally as follows:

$$Ch(\mathbf{x}, \mathbf{y}) = \frac{1}{|\mathbf{x}|}\sum_{x\in\mathbf{x}} \min_{y\in\mathbf{y}} ||x - y||_2 + \frac{1}{\|\mathbf{y}\|}\sum_{y\in\mathbf{y}} \min_{x\in\mathbf{x}} ||x - y||_2 \tag{6}$$

The Chamfer distance is scaled by ρ, which is the length of the image diagonal, such that the final value is between 0 and 1. Under the setting of this problem, the reward $1 - Ch(\mathbf{x}, \mathbf{y})$ mostly falls between 0.9 and 1. We exponentiate $1 - Ch(\mathbf{x}, \mathbf{y})$ to the power of $\gamma = 20$ to achieve smoother gradients (Laud, 2004). We add another pixel intersection based component to differentiate shapes with similar sizes and locations. The final reward function is:

$$R(\mathbf{x}, \mathbf{y}) = \max(\delta, (1 - \frac{Ch(\mathbf{x}, \mathbf{y})}{\rho})^\gamma + \frac{\sum_{x\in\mathbf{x}\cap\mathbf{y}} 1}{\sum_{x\in\mathbf{x}} 1}) \tag{7}$$

The first and second part of the reward function provide feedback on the physical distance and similarity between the prediction and the target respectively. We clip the reward below $\delta = 0.3$ to simplify it when the quality of the generated images are poor. A low reward value provides little insight on its performance and is largely dependent on its target image. Similar reward clipping idea was proposed in DQN (Mnih et al., 2013) and was used to unify the reward ranges across different games to $[-1, 1]$.

A model trained with REINFORCE objective (Sharma et al., 2018) only is unable to improve beyond the lowest reward (Figure 4 (green)). We introduce three crucial designs to improve learning. Firstly,

we propose an entropy estimator for a tree model to encourage exploration; Secondly, we perform sampling without replacement in the program space to facilitate optimization and further encourage exploration; and lastly, we adopt a grammar-encoded tree LSTM to ensure valid output sequences with an image stack to provide intermediate feedbacks.

Algorithm 1 Sampling w/o Replacement Tree LSTM

Input: Target Image $\mathbf{x}$, Number of beams k
Initialize: Grammar stack S, Image stack I, Beam set $\mathbb{B}$

Encode the target image $\tilde{T} \leftarrow \text{Encode}(T)$
$\mathbb{B} = \{\mathbf{s}^i | \mathbf{s}^i = \emptyset\}$ for $i \in 1, 2, \cdots k+1$ and $\mathcal{H}(v) = 0$
for $j := 1$ to n **do**
  $\mathbf{\Phi}_{:,j}, \mathcal{H}_{i,j} \leftarrow \text{TreeLSTM}(S, I, \mathbb{B}, \mathbf{x})$ (Algorithm 2)
  $\mathbb{B} \leftarrow \text{Sample\_w/o\_Replacement}(\mathbf{\Phi}_{:,j}, \mathbb{B}, k)$ (See Kool et al. (2019b) and Algorithm 3)
  $\hat{\mathcal{H}}_D \leftarrow \hat{\mathcal{H}}_D + \sum_{j=1}^{n} \frac{1}{W_j(S)} \sum_{s^i \in S} \frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)} \mathcal{H}_{i,j}$ (Equation 14 and Appendix B)
  if $s_{j+1}^i \in \mathcal{G}$ **then** $S_i.\text{push}(s_{j+1}^i)$ **else** $I_i.\text{push}(s_{j+1}^i), \forall s^i \in \mathbb{B}$
end for
$\mathbf{y}_i = \text{Render}(s^i)$ for $i \in 1, 2, \cdots k$
Maximize $\mathbb{E}[\sum_{i=1}^{k} R(\mathbf{x}, \mathbf{y}_i)] + \alpha \hat{\mathcal{H}}_D$

4.1 Exploration with Entropy Regularization

Entropy regularization in reinforcement learning is a standard practice for encouraging exploration. We present an entropy estimation suitable for sampling top-down from a syntax tree.

Let S be the random variable of possible programs. The entropy is denoted as $\mathcal{H}(S) = \mathbb{E}[-\log P(S)]$ [1]. Because the possible outcomes of S can be exponentially large and we cannot enumerate all of them to estimate the expectation. The entropy can be estimated naively via

$$\hat{\mathcal{H}} = -\frac{1}{K} \sum_{i=1}^{K} \log P(s^i), \tag{8}$$

with finite samples $\{s^i\}_{i=1}^K$. Without further assumption, we are not able to improve $\hat{\mathcal{H}}$. However, we can decompose program into $S = X_1 \dots X_n$, where each X_j is the random variable for the token at position j in the program. Under autoregressive models (e.g. RNN), we can further access the conditional probability. Therefore, we propose a decomposed entropy estimator $\hat{\mathcal{H}}_D$ as

$$\hat{\mathcal{H}}_D = \frac{1}{K} \sum_{i=1}^{K} \sum_{j=1}^{n} \mathcal{H}(X_j | X_1 = x_1^i, \cdots, X_{j-1} = x_{j-1}^i), \tag{9}$$

where $s^i = x_1^i, \dots, x_n^i$, and $\mathcal{H}(X_j | X_1 = x_1^i, \cdots, X_{j-1} = x_{j-1}^i)$ is the conditional entropy.

Lemma 4.1. *The proposed decomposed entropy estimator $\hat{\mathcal{H}}_D$ is unbiased with lower variance, that is $\mathbb{E}[\hat{\mathcal{H}}_D] = \mathcal{H}(S)$ and Var$(\hat{\mathcal{H}}_D) \leq$ Var$(\hat{\mathcal{H}})$.*

The proof is simple by following Cover & Thomas (2012) and we leave it in Appendix C and D.

4.2 Effective Optimization By Sampling Without Replacement

After establishing our REINFORCE with entropy regularization objective, now we show the intuition behind choosing sampling without replacement (SWOR) over sampling with replacement (SWR) with a synthetic example (Figure 2). We initialize a distribution of $m = 100$ variables with three of them having significantly higher probability than the others (Figure 2 (2)). The loss function is entropy. Its estimator is $\frac{1}{m} \sum_{i=1}^{m} \log p_i$ for SWR and $\sum_{i=1}^{m} \frac{p_i}{q_i} \log p_i$ for SWOR. In both cases, p_i is the i-th variable's probability. q_i is the re-normalized probability after SWOR. $\frac{p_i}{q_i}$ is the importance weighting. The increase in entropy by sampling 20 variables without replacement is more rapid than 40 variables with replacement. At the end of the 700 iterations, the distribution under SWOR is visibly more uniform than the other. SWOR would achieve better exploration than SWR.

[1] Here we overload S and P as used in equation 1 and equation 2 to follow the convention.

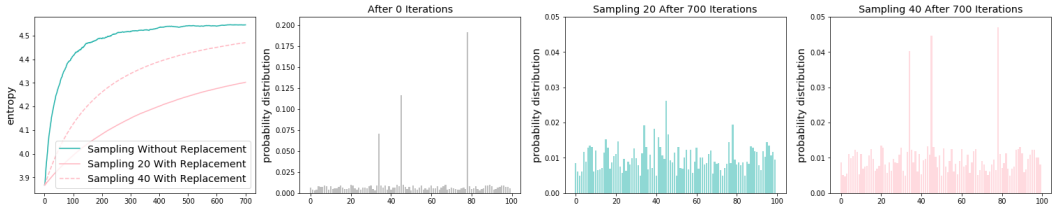


Figure 2: The left most image demonstrates the entropy value increases over 700 iterations by sampling 20 distinct samples with and without replacement as well as sampling 40 samples with replacement. The second image shows the initial distribution. The third and fourth images show the final distributions.

To apply SWOR to our objective, both of the REINFORCE objective and the entropy estimator require importance weightings. Let s^i_j denotes the first j elements of the sequence s^i:

$$\nabla_\theta \mathbb{E}_{s\sim p_\theta(s)}[f(s)] \approx \sum_{s^i \in S} \frac{\nabla_\theta p_\theta(s^i)}{q_\theta(s^i)} f(s^i) \quad \text{and,} \tag{10}$$

$$\hat{\mathcal{H}}_D \approx \sum_{j=1}^{n} \sum_{s^i \in S} \frac{p_\theta(s^i_j)}{q_\theta(s^i_j)} \mathcal{H}(X_j | X_1 = x^i_1, \cdots, X_{j-1} = x^i_{j-1}) \tag{11}$$

Implementing SWOR on a tree structure to obtain the appropriate set of programs S is challenging. It is not practical to instantiate all paths and perform SWOR bottom-up. Instead, we adopt a form of stochastic beam search by combining top-down SWOR with Gumbel trick that is equivalent to SWOR bottom-up (Kool et al., 2019b). The exact sampling process is described in Algorithm 3. In Appendix B, we will discuss the implementation of the re-normalized probability $q_\theta(s^i)$ as well as some additional tricks of variance reduction for the objective function.

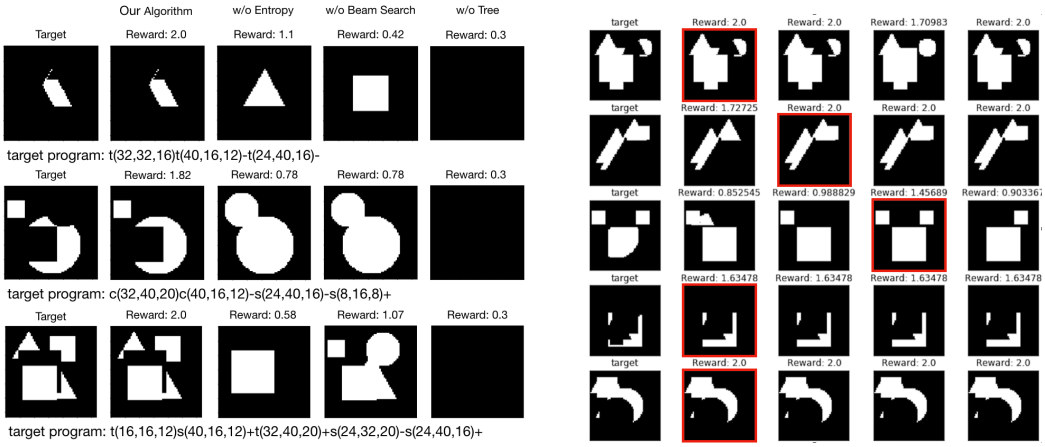
4.3 Grammar Encoded Tree LSTM

We introduce a *grammar-encoded tree LSTM* which encodes the production rules into the model, thus guaranteed to generate correct programs, and significantly reduce the search space during the training (Kusner et al., 2017; Alvarez-Melis & Jaakkola, 2016; Parisotto et al., 2016; Yin & Neubig, 2017). There are 3 types of production rules in the grammatical program generation – shape selection (P), operation selection (T), and grammar selection (E). Grammar selection in this problem setting includes $E \to EET$, and $E \to P$ and they decide whether the program would expand. We denote the set of shape, operations and non-terminal outcomes (e.g. EET in equation 2) to be $\mathcal{P}$, $\mathcal{T}$ and $\mathcal{G}$ respectively. A naive parameterization is to let the candidate set of the LSTM output to be $\{S, \$\} \cup \mathcal{T} \cup \mathcal{P}$, where \$ is the end token, and treat it as a standard language model to generate the program (Sharma et al., 2018). The model does not explicitly encode grammar structures, and expect the model to capture it implicitly during the learning process. The drawback is that the occurrence of valid programs is sparse during sampling and it can prolongs training significantly.

The proposed model can be described as an RNN model with a *masking mechanism* by maintaining a grammar stack to rule out invalid outputs. We increase the size of the total output space from $2+|\mathcal{P}|+|\mathcal{T}|$ of the previous approach (e.g. Sharma et al. (2018)) to $2+|\mathcal{P}|+|\mathcal{T}|+|\mathcal{G}|$ by including the non-terminals. During the generation, we maintain a stack to trace the current production rule. Based on the current non-terminal and its corresponding expansion rules, we use the masking mechanism to weed out the invalid output candidates. Take the non-terminal T for example, we mask the invalid outputs to reduce the candidate size from $2 + |\mathcal{P}| + |\mathcal{T}| + |\mathcal{G}|$ to $|\mathcal{T}|$ only. In this process, the model will produce a sequence of tokens, including grammatical, shape and operation tokens. We only keep the terminals as the final output program and discard the rest. The resulting programs are ensured to be grammatically correct. During the generation process, grammatical tokens are pushed onto the grammar stack while intermediate images and operations are pushed onto an *image stack*. Images in the image stack are considered observations and are part of the input to the LSTM to aid the inference in the search space. A step-by-step guide through the tree LSTM for better understanding is in Appendix A and Figure 1 is a visual representation of the process.

5 Experiments

We are going to investigate how each design feature affects learning and how our algorithm compares to a supervision based method. Firstly, we utilize a synthetic dataset generated by the CFG specified



(a) Example output of each method **(b)** Example output of each beam

Figure 3: (a) We show a target image from each dataset and attach its correct program below. To the right of the target program are the reconstructed output programs from our algorithm and three variants each missing one design feature. The reward is on top of the reconstructed images. (b) Some reconstructed example output programs of our algorithm. Each row represents one data point. The leftmost images of the five columns are the target images and the four columns to their right are the reconstructed outputs of four samples. The final output is the program of the reconstruction with the highest reward (highlighted in red).

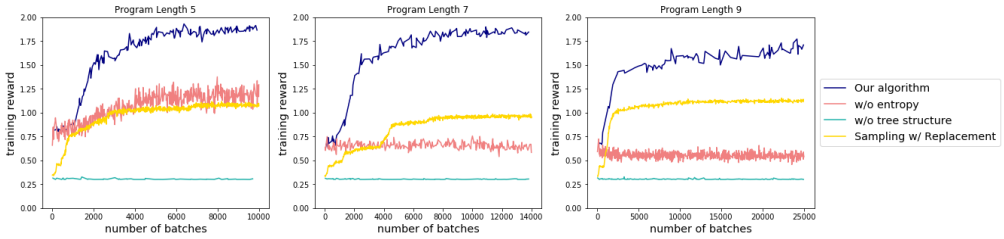


Figure 4: From left to right, we have reward per batch for programs of length 5, 7 and 9. It demonstrates the performance of our algorithm and controlled comparison in performance with alternative algorithms by removing one component at a time.

in Section 3 to perform ablation study. Next, we use a 2D CAD furniture dataset as input and show that the programs our algorithm generate can *approximate* them despite not having an exact match. We train supervised models on both datasets and our method is able to achieve comparable, and in some case – superior, result. Lastly, we show empirically that the stepwise entropy estimator (Equation 9) has smaller variance than the naive estimator (Equation 8).

5.1 Ablation Study of Design Features

We use three synthetic datasets to test our algorithm. The action space includes 27 shape (Figure 7), 3 operation and 2 grammar to create images on a 64 by 64 canvas. The search space for an image up to 3 shapes (or program length 5) is around 1.8×10^5 and it gets up to 1.1×10^9 for 5 shapes (or program length 9). We separate our dataset by the length of the program to study the effect of image complexity. Our synthetic dataset is created by generating all combinations of shape and operation actions in text for each length of program and filtering out the duplicates and empty images. Images are considered duplicates if only 120 pixels are different between the two and are considered empty if there are no more than 120 pixels on the canvas. Table 1 contains the dataset size information. For these 3 datasets, we sampled 19 programs without replacement for each target image. The negative entropy coefficient is 0.05 and the learning rate is 0.01. We use SGD with 0.9 momentum.

Type	Length 5	Length 7	Length 9	2D CAD
Training set size	3600	4800	12000	10000
Testing set size	586	789	4630	3000

Table 1: Dataset statistics.

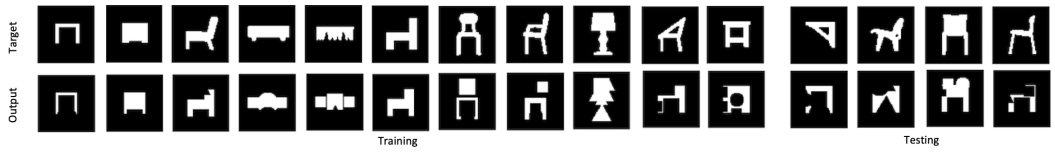


Figure 5: The four examples on the right are from the test set, and the rest on the left are from the training set. The target images are on top and the reconstruction from the output programs are at the bottom.

Test Metric	Length 5	Length 7	Length 9
Chamfer	0.985	0.960	0.969
IoU	0.996	0.964	0.969

Chamfer	length 5	length 7	length 9
Training	0.997	0.996	0.994
Testing	0.987	0.906	0.833

Table 2: (Left) The performance of the converged model of our algorithm on the test set measured with Chamfer distance to the power of 20 and IoU. (Right) Supervised training results.

Removing either one of the three design features has reduced the performance of our algorithm. Under sampling with replacement setting, the model is quickly stuck at a local optimum (Figure 4 (yellow)). The starting reward is lower comparing to sampling without replacement because its final reward is the maximum reward of all distinct programs. Without the entropy term in the objective function, the reward function is able to improve on the length 5 dataset but fails to do so on longer programs. With the tree structure removed, the reward stays around the lowest reward (Figure 4 (green)) because the program is unable to generate valid programs to render. All these design features are crucial to learning increasingly complex images.

We allow variations in the generated programs as long as the target images can be recovered, thus we evaluate the program quality in terms of the reconstructed image's similarity to the target image. We measure our converged algorithm's performance (Table 2 (Left)) on the three test sets with Chamfer and IoU reward metrics (Equation 7 first and second term). The perfect match receives 1 in both metrics. Figure 3 provides some qualitative examples on the algorithms in Figure 4.

5.2 Experiment on 2D CAD Furniture Dataset

The dataset used in this experiment is a 2D CAD dataset (Sharma et al., 2018) that contains binary 64×64 images of various furniture items. We apply our algorithm to this problem with an action space of 396 basic shapes plus the operations and grammatical terminals described in Section 3. We limit the number of LSTM iterations to 24 steps, which corresponds to a maximum of 6 shapes. For an image up to 6 shapes the search space is 9.4×10^{17}. If we remove the grammar-encoded tree structure, the search space is 3.8×10^{28}. The learning rate and entropy values used are 0.01 and 0.7 respectively. We train the model with only the Chamfer reward (first part of the Equation 7) because these images are not generated by CFG and exact match solutions do not exist. During training, the reward converges to 0.7. Qualitative results are reported from the training and test set (Figure 5). The program reconstructions are able to capture the overall profile of the target images. However, the cutouts and angles deviate from the original because the shape actions consist solely of unrotated squares, perfect circles and equilateral triangles.

5.3 Comparison with Supervised Learning Method

We compare the training and testing results using a supervised learning method with the same neural network model to the unsupervised method on the synthetic dataset. The input at each step is the concatenation of the embedded ground truth program and the encoded final and intermediate images. We use the same synthetic dataset and the Chamfer reward (Equation 7 first term) metric in Table 2 (Left) to measure the quality of the output programs. The testing results of the supervised method worsen with the increasing complexity (program length) while the training results are almost perfect across all three datasets. The unsupervised method receives consistently high scores. This shows that the supervised training method does not generalize well to new data in comparison to the unsupervised method (Table 2 (Right)). This may be explained by program aliasing described by Bunel et al. (2018). Our conjecture is that the supervised learning optimizes over the loss function in the program space while the unsupervised learning directly optimizes over the reward function in the image space.

We also took the model pretrained on 1.2M ground truth programs from Sharma et al. (2018) to compare with our performance on the CAD dataset. We allow our models to generate up to 6 shapes and measure the image similarity in Chamfer Distance (Equation 6). We further fine-tuned the

CD	k = 1	k = 10	k = 50	k = 100
Pretrained Model	4.62	1.89	1.39	1.31
Fine-tuned Model	1.43	1.27	1.20	1.18
Our Model	1.57	1.07	0.83	**0.82**

Table 3: Empirical comparison of pretrained model, fine-tuned model and our model on 2D CAD dataset with Chamfer distance

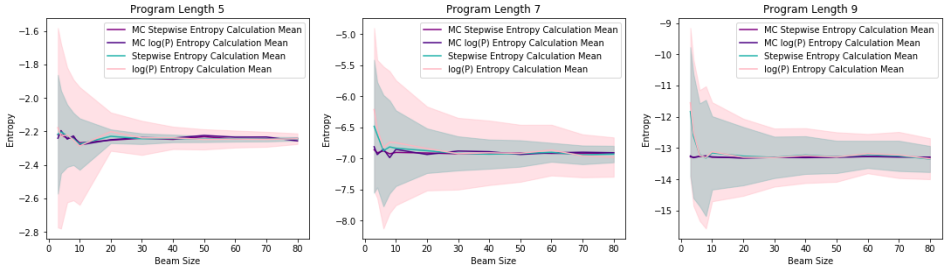


Figure 6: Compare the entropy estimation following the Equation 14 as a weighted sum of stepwise entropy versus taking the average of the sequence log probability. The number of samples, or beam size, varies from 2 to 80 on the x-axis. The shaded area represents the standard deviation of each estimator. From left to right, we demonstrate the result on datasets of three program lengths.

pretrained model on CAD dataset. Our model was trained directly on the CAD dataset without supervision. We report the result of the pretrained model, fine-tuned model and our model by beam search with $k = 1, 10, 50, 100$ in Table 3. The pretrained model is not able to directly generalize to the novel dataset. The fine-tuned model perform better than our model at $k = 1$. But beam search benefits our model significantly more than a model fine-tuned by vanilla REINFORCE. Our hypothesis is sampling without replacement, or stochastic beam search, during training also teaches the model to use beam search more effectively in test time (Negrinho et al., 2018). The least Chamfer distance is achieved at 0.82 with $k = 100$ by our model.

5.4 Variance Study of Entropy Estimation

This study (Figure 6) shows that the estimator $\hat{\mathcal{H}}_D$ achieves a lower variance than $\hat{\mathcal{H}}$ (Section 4.1).

We take a single model saved at epoch 40 during the training time of the length 5, 7, and 9 dataset and estimate the entropy with $\hat{\mathcal{H}}_D$ (Equation 9) and $\hat{\mathcal{H}}$ (Equation 8). We consider two sampling schemes: with and without replacement. We combine both entropy estimation methods with the two sampling schemes creating four instances for comparisons. The x-axis of the plot documents the number of samples to obtain a single estimation of the entropy. We further repeat the estimation 100 times to get the mean and variance. The means of SWR method act as a baseline for the means of the SWOR while we compare the standard deviations (the shaded area) of the two entropy estimation methods.

Across all three datasets, $\hat{\mathcal{H}}_D$ (green) shows significantly smaller variance with the number of samples ranges from 2 to 80. But we notice that longer programs, or more complex images, require much more samples to reduce the variance. This makes sense because the search space increases exponentially with longer program length. The initial bias in the SWOR estimation dissipates after the number of samples grows over 10. The initial bias is greater in dataset with longer program length.

6 Discussion

Our algorithm is the first to parses a CSG image, created by a non-differentiable renderer, into CFG programs without supervision. Instead of relying on supervised pretrainig, we work on improving the sample efficiency of the REINFORCE-based algorithm applied in this task. To simplify the parsing pipeline, we first augment the REINFORCE objective with an unbiased and low-variance entropy estimator of the model to encourage exploration, then we optimize the new objective by sampling without replacement from the CFG syntax tree enforced by a grammar-encoded tree LSTM. Our experiments have demonstrated the importance of each design feature qualitatively and quantitatively. We also show that our unsupervised model generalizes better given the same amount of synthetic data and it outperforms the supervise-pretrained RL fine-tuned model under beam search on the 2D CAD dataset.

REFERENCES

David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. 2016.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *International Conference on Representation Learning (ICLR)*, 2017.

Marc Brockschmidt, Miltiadis Allamanis, Alexander L Gaunt, and Oleksandr Polozov. Generative code modeling with graphs. *arXiv preprint arXiv:1805.08490*, 2018.

Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, 2016.

Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided neural program synthesis. 2018a.

Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In *Advances in neural information processing systems*, pp. 2547–2557, 2018b.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 990–998. JMLR. org, 2017.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in neural information processing systems*, pp. 6059–6068, 2018.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl, 2019.

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

David Ha and Douglas Eck. A Neural Representation of Sketch Drawings. *ArXiv e-prints*, April 2017.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2017.

Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning. *arXiv preprint arXiv:1903.04411*, 2019.

Philip M Hubbard. *Constructive solid geometry for triangulated polyhedra*. 1990.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019a.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pp. 3499–3508, 2019b.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1945–1954. JMLR. org, 2017.

Adam Daniel Laud. Theory and application of reward shaping in reinforcement learning. Technical report, 2004.

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pp. 2203–2213, 2017.

Yunchao Liu, Zheng Wu, Daniel Ritchie, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to describe scenes with programs. 2018.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Renato Negrinho, Matthew Gormley, and Geoffrey J Gordon. Learning beam search policies via imitation learning. In *Advances in Neural Information Processing Systems*, pp. 10652–10661, 2018.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

Lukasz Romaszko, Christopher KI Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 851–859, 2017.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5515–5523, 2018.

Richard Shin, Illia Polosukhin, and Dawn Song. Improving neural program synthesis with inferred execution traces. In *Advances in Neural Information Processing Systems*, pp. 8917–8926, 2018.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv preprint arXiv:1901.02875*, 2019.

Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2635–2643, 2017.

Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pp. 6306–6315, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 699–707, 2017.

Shunyu Yao, Tzu Ming Hsu, Jun-Yan Zhu, Jiajun Wu, Antonio Torralba, Bill Freeman, and Josh Tenenbaum. 3d-aware scene manipulation via inverse graphics. In *Advances in Neural Information Processing Systems*, pp. 1887–1898, 2018.

Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

Amit Zohar and Lior Wolf. Automatic program synthesis of long programs with a learned garbage collector. In *Advances in Neural Information Processing Systems*, pp. 2094–2103, 2018.

A GRAMMAR TREE LSTM GUIDE

This section provides a guide for the tree-LSTM illustration in Figure 1. This guide follows the arrows in the illustration (Figure 1) from left to right:

Algorithm 2 TreeLSTM Model

Input: Grammar Stack S, Image Stack I, Target Image $\mathbf{x}$, Sample Set $\mathbb{B}$

function TREELSTM($S, I, \mathbb{B}, \mathbf{x}$)
  $\tilde{\mathbf{x}} \leftarrow \text{Encode}(\mathbf{x})$
  for $s^i \in \mathbb{B}$ **do**
    $g_i \leftarrow S_i.\text{pop}()$
    $\tilde{g}_i \leftarrow \text{Embed}(g_i)$
    $\tilde{I}_i \leftarrow \text{Encode}(I_i)$
    $H_{i,j} \leftarrow \text{LSTM}(\tilde{g}_i, \tilde{I}_i, \tilde{\mathbf{x}}, H_{i,j-1})$
    $\mathbf{p_{i,j}} \leftarrow \text{softmax}(f(H_{i,j}) + \text{Mask}(g_i))$
    Estimate entropy at this node $v_{i,j}$:
    $\mathcal{H}_{i,j} = \tilde{\mathbf{p}}_{\mathbf{i,j}} \cdot \log \tilde{\mathbf{p}}_{\mathbf{i,j}}$
    Update the log probabilities of partial sequences
    $\vec{\phi}_{i,j} = \mathbb{1} \cdot \phi_{s^i_{j-1}} + \log \mathbf{p_{i,j}}$
  end for
  return $\mathbf{\Phi}_{:,j+1}, \mathcal{H}_{i,j}$
end function

- A grammar stack with a start token S and an end token \$ as well as an empty image stack is initialized.
- In the first iteration, the token S is popped out. Following Rule equation 1, all other options will be masked except E, the only possible output. E token is added to the stack.
- In the second iteration, or any iteration where the token E is popped, the input for all examples and all softmax outputs are masked except the entries representing EET and P according to Rule equation 2. If EET is sampled, T, E and E tokens will be added to the stack separately in that exact order to expand the program further. If P is sampled, it will be added to the stack and the program cannot expand further.
- If T is popped out of the stack, the output space for that iteration will be limited to all the operations (Rule equation 3). Similarly, if P is popped out, the output space is limited to all the geometric shapes (Rule equation 4).
- When a shape token is sampled, it will not be added to the grammar stack as they do not contribute to the program structure. Instead, the image of the shape will be pushed onto the corresponding image stack.
- When an operation token is sampled, it also will not be added to the grammar stack. Instead, we pop out the top two images to apply the operation on them and push the final image onto the image stack again.
- When the stack has popped out all the added tokens, the end token \$ will be popped out in the last iteration. We then finish the sampling as standard RNN language models.

In practice, we implement the masking mechanism by adding a vector to the output before passing into softmax layer to get the probability. The vector contains 0 for valid output and large negative numbers for invalid ones. This makes sure that invalid options will have almost zero probability of being sampled. The input of the RNN cell includes encoded target image and intermediate images from the image stack, embedded pop-out token from grammar stack and the hidden state from the RNN's last iteration. The exact algorithm is in Algorithm 2.

B SAMPLING WITHOUT REPLACEMENT

This section describes how we achieve sampling without replacement with the help of stochastic beam search (Kool et al., 2019b).

At each step of generation, the algorithm chooses the top-k beams to expand based on the $\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}}$ score at time step j. Let A be all possible actions at time step j, $\vec{\phi}_{i,j} \in \mathbb{R}^A$ is the log probability of each outcomes of sequence i at time j plus the log probability of the previous $j-1$ actions.

$$\vec{\phi}_{i,j} = \big[\log P(a_1), \log P(a_2), \ldots, \log P(a_A)\big] + \log P(a_{t_1}, a_{t_2}, a_{t_j}) \cdot \mathbb{1}$$

For each beam, we sample a Gumbel random variable $G_{\phi_{i,j,a}} = \text{Gumbel}(\phi_{i,j,a})$ for each of the element a of the vector $\vec{\boldsymbol{\phi}}_{i,j}$. Then we need to adjust the Gumbel random variable by conditioning on its parent's stochastic score $G_{s^i_{j-1}} = \text{Gumbel}(\log P(a_{t_1}, a_{t_2}, a_{t_{j-1}}))$ being the largest (Equation 12) in comparison to all elemtns in $\vec{\mathbf{G}}_{\phi_{i,j}}$, the resulting value $\vec{\tilde{\mathbf{G}}}_{\phi_{i,j}} \in \mathbb{R}^A$ is the adjusted stochastic score for each of the potential expansions.

$$\vec{\tilde{\mathbf{G}}}_{\phi_{i,j}} = -\log(\exp(-\mathbb{1} \cdot G_{s^i_{j-1}})) - \exp(-\mathbb{1} \cdot Z_{i,j}) + \exp(-\mathbf{G}_{\boldsymbol{\phi}_{i,j}}) \tag{12}$$

Here $Z_{i,j}$ is the largest value in the vector $\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}}$ and $G_{s^i_{j-1}}$ is the stochastic score of i-th beam at step $j-1$. Conditioning on the parent stochastic score being the largest in this top-down sampling scheme makes sure that each leaf's stochastic score $G_{\phi_{i,j}} \sim \text{Gumbel}(\phi_{i,j})$ is independent, equivalent to sampling the sequences bottom up (Kool et al., 2019b). Once we have aggregated all the stochastic scores in all $k+1$ beams, we select the top-$k+1$ scored beams from $(k+1) * A$ scores for expansions. Note that the reason that we maintain one more beam than we intended to expand because we need the $k+1$ largest stochastic score to be the threshold during estimation of the entropy and REINFORCE objective. This is explained next. Please refer to Algorithm 3 for details in the branching process.

Algorithm 3 Sampling_w/o_Replacement

Input: Log probability at time j $\boldsymbol{\phi}_{:,j}$ Beam Set $\mathbb{B}$, Number of beams k

function SAMPLING_W/O_REPLACEMENT($\boldsymbol{\Phi}_{:,j}, \mathbb{B}, k$)
  $\tilde{\boldsymbol{G}} \leftarrow \emptyset$
  for $s^i \in \mathbb{B}$ **do**
    $\vec{\mathbf{G}}_{\vec{\boldsymbol{\phi}}_{i,j}} \sim \text{Gumbel}(\vec{\boldsymbol{\phi}}_{i,j})$
    $Z_{i,j} \leftarrow \max(\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}})$
    Calculate $\vec{\tilde{\mathbf{G}}}_{\boldsymbol{\phi}_{i,j}}$ (Equation 12)
    Aggregate the values in the vector $\tilde{\mathbf{G}}_{\phi_{i,j}}$
    $\tilde{\boldsymbol{G}} \leftarrow \tilde{\boldsymbol{G}} \cup \tilde{\mathbf{G}}_{\phi_{i,j}}$
  end for
  Choose top $k+1$ values in $\tilde{\boldsymbol{G}} \in \mathbb{R}^{(k+1)\cdot A}$ and form the new beam set
  $\tilde{\mathbb{B}} = \{\tilde{\mathbf{s}}^i | \tilde{\mathbf{s}}^i \cup \tilde{s}^i_j, \}$where $i \in 1, 2, \cdots, k+1$
  return $\tilde{\mathbb{B}}$
end function

The sampling without replacement algorithm requires importance weighting of the objective functions to ensure unbiasness. The weighting term is $\frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)}$. $p_\theta(s^i)$ represents the probability of the sequence s^i and S represents the set of all sampled sequences s^i for $i = 1, 2, \cdots, k$. $q_{\theta,\kappa}(s^i_j) = P(G_{s^i_j} > \kappa)$ where κ is the $(k+1)$-th largest $G_{s^i_j}$ score for all i. It acts as a threshold for branching selection. During implementation, we need to keep an extra beam, thus $k+1$ beams in total, to accurately estimate κ in order to ensure the unbiasness of the estimator.

To reduce variance of our objective function, we introduce additional normalization terms as well as a baseline. However, the objective function is biased with these terms. The normalization terms are $W(S) = \sum_{s^i \in S} \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)}$ and $W^i(S) = W(S) - \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} + p_\theta(s^i)$.

Incorporating a baseline into the REINFORCE objective is a standard practice. A baseline term is defined as $B(S) = \sum_{s^i \in S} \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} f(s^i)$.

To put everything together, the exact objective is as follows (Kool et al., 2019a):

$$\nabla_\theta \mathbb{E}_{s \sim p_\theta(s)}[f(s)] \approx \sum_{s^i \in S} \frac{1}{W^i(S)} \cdot \frac{\nabla_\theta p_\theta(s^i)}{q_{\theta,\kappa}(s^i_n)} (f(s^i) - \frac{B(S)}{W(S)}) \tag{13}$$

Entropy estimation uses a similar scaling scheme as the REINFORCE objective:

$$\hat{\mathcal{H}}_D(X_1, X_2, X_3, \cdots, X_n) \approx \sum_{j=1}^{n} \frac{1}{W_j(S)} \sum_{s^i \in S} \frac{p_\theta(s^i_j)}{q_{\theta,\kappa}(s^i_j)} \mathcal{H}(X_j | X_1 = x^i_1, \cdots, X_{j-1} = x^i_{j-1}) \tag{14}$$

where $W_j(S) = \sum_{s^i \in S} \frac{p_\theta(s^i_j)}{q_{\theta,\kappa}(s^i_j)}$ and s^i_j denotes the first j elements of the sequence s^i. The estimator is unbiased excluding the $\frac{1}{W_j(S)}$ term.

C Proof of Stepwise Entropy Estimation's Unbiasness

Entropy of a sequence can be decomposed into the sum of the conditional entropy at each step conditioned on the previous values. This is also called the chain rule for entropy calculation. Let $X_1, X_2, \cdots, X_n$ be drawn from $P(X_1, X_2, \cdots, X_n)$ Cover & Thomas (2012):

$$\mathcal{H}(X_1, X_2, \cdots, X_n) = \sum_{j=1}^{n} \mathcal{H}(X_j | X_1, \cdots, X_{j-1}) \tag{15}$$

If we sum up the empirical entropy at each step after the softmax output, we can obtain an unbiased estimator of the entropy. Let S be the set of sequences that we sampled and each sampled sequence s^i consists of $X_1, X_2, \cdots, X_n$:

$$\begin{aligned}
\mathbb{E}_{X_1, X_2, \ldots, X_{j-1}}(\hat{\mathcal{H}}_D) &= \mathbb{E}_{X_1, X_2, \ldots, X_{j-1}}(\frac{1}{|S|} \sum_{i \in |S|} \sum_{j=1}^{n} \mathcal{H}(X_j | X_1 = x^i_1, \ldots, \quad X_{j-1} = x^i_{j-1})) \\
&= \frac{1}{|S|} \cdot |S| \sum_{j=1}^{n} \mathcal{H}(X_j | X_1, \cdots, \quad X_{j-1}) \\
&= \mathcal{H}(X_1, X_2, \cdots, X_n)
\end{aligned}$$

In order to incorporate the stepwise estimation of the entropy into the beam search, we use the similar reweighting scheme as the REINFORCE objective. The difference is that the REINFORCE objective is reweighted after obtaining the full sequence because we only receive the reward at the end and here we reweight the entropy at each step. We denote each time step by j and each sequence by i, the set of sequences selected at time step j is S_j and the complete set of all possible sequences of length j is T_j and $S_j \in T_j$. We are taking the expectation of the estimator over the $G_{\phi_{i,j}}$ scores. As we discussed before, at each step, each potential beam receives a stochastic score $G_{\phi_{i,j}}$. The beams associated with the top-$k+1$ stochastic scores are chosen to be expanded further and κ is the $k+1$-th largest $G_{\phi_{i,j}}$. κ can also be seen as a threshold in the branching selection process

and $q_{\theta,\kappa}(s_j^i) = P(G_{s_j^i} > \kappa) = 1 - \exp(-\exp(\phi_{i,j} - \kappa))$. For details on the numerical stable implementation of $q_{\theta,\kappa}(s_j^i)$, please refer to Kool et al. (2019b).

$$
\begin{aligned}
&\mathbb{E}_{G_\phi}(\sum_{j=1}^{n}\sum_{s_j^i \in S_j}\frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)}\mathcal{H}(X_j|X_1 = x_1^i, X_2 = x_2^i, \cdots, \quad X_{j-1} = x_{j-1}^i)) \\
&= \sum_{j=1}^{n}\mathbb{E}_{G_\phi}(\sum_{i\in|T_j|}\frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)}\mathcal{H}(X_j|X_1 = x_1^i, X_2 = x_2^i, \quad \cdots, X_{j-1} = x_{j-1}^i))\mathbb{1}_{\{x_1^i,\cdots,x_j^i\}\in S_j}) \\
&= \sum_{j=1}^{n}\sum_{i\in|T_j|}p_\theta(s_j^i)\mathcal{H}(X_j|X_1 = x_1^i, X_2 = x_2^i, \cdots, \quad X_{j-1} = x_{j-1}^i)\mathbb{E}_{G_\phi}(\frac{\mathbb{1}_{\{s_j^i = x_1^i,\cdots,x_j^i\}\in S_j}}{q_{\theta,\kappa}(s_j^i)}) \\
&= \sum_{j=1}^{n}\mathcal{H}(X_j|X_1, X_2, \cdots, X_{j-1})\cdot 1 \\
&= \mathcal{H}(X_1, X_2, \cdots, X_n)
\end{aligned}
$$

For the proof of $\mathbb{E}_{G_\phi}(\frac{\mathbb{1}_{\{s_j^i\in S_j}}{q_{\theta,\kappa}(s_j^i)}) = 1$, please refer to Kool et al. (2019b), appendix D.

D Proof of Lower Variance of the Stepwise Entropy Estimator

We will continue using the notations from above. We want to compare the variance of the two entropy estimator $\hat{\mathcal{H}}$ and the stepwise entropy estimator $\hat{\mathcal{H}}_D$ and show that the second estimator has lower variance.

Proof. We abuse $\mathbb{E}_{X_j}$ to be $\mathbb{E}_{X_j|X_1,\ldots,X_{j-1}}$ and Var_{X_j} to be $\mathrm{Var}_{X_j|X_1,\ldots,X_{j-1}}$ to simplify the notations.

$$
\begin{aligned}
&\mathrm{Var}_{X_1,X_2,\cdots,X_n}(\frac{1}{|S|}\sum_{i\in|S|}\sum_{j=1}^{n}\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}\mathrm{Var}_{X_1,X_2,\cdots,X_n}(\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}(\mathbb{E}(\mathcal{H}^2(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\quad - \mathbb{E}^2(\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i))) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}(\mathbb{E}_{X_1,\cdots,X_{j-1}}\mathbb{E}_{X_j}^2(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\quad - \mathbb{E}_{X_1,\cdots,X_{j-1}}^2\mathbb{E}_{X_j}(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i))) \\
&= \frac{1}{|S|^2}\sum_{s^i\in S}\sum_{j=1}^{n}(\mathbb{E}_{X_1,\cdots,X_{j-1}}(\mathbb{E}_{X_j}(\log^2 P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\quad - \mathrm{Var}_{X_j}(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i))) \\
&\quad - \mathbb{E}_{X_1,\cdots,X_{j-1}}^2\mathbb{E}_{X_j}(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i))) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}(\mathbb{E}_{X_1,\cdots,X_j}(\log^2 P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\quad - \mathbb{E}_{X_1,\cdots,X_j}^2(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)))
\end{aligned}
$$

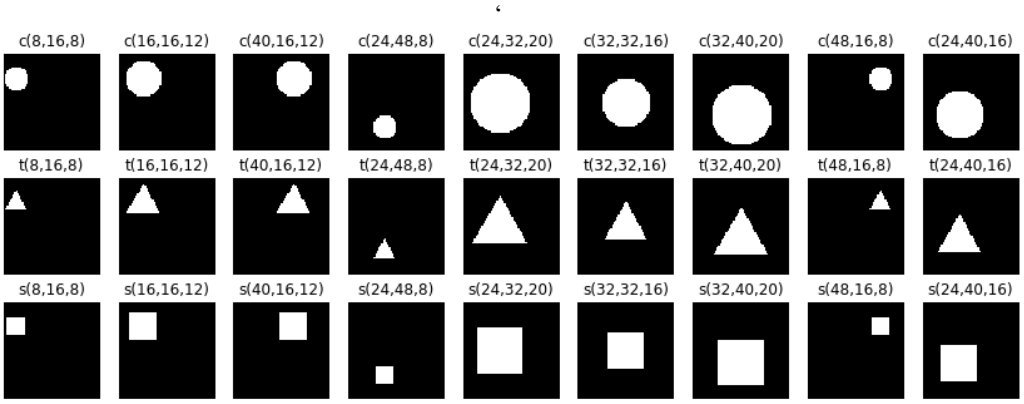


Figure 7: Each shape encoding is on top of the image it represents.

$$
\begin{aligned}
&- \mathbb{E}_{X_1,\cdots,X_{j-1}} \mathrm{Var}_{X_j}(\log P(X_j | X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \frac{1}{|S|^2} \sum_{i \in |S|} \sum_{j=1}^{n} (\mathrm{Var}_{X_1,\cdots,X_j}(\log P(X_j | X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&- \mathbb{E}_{X_1,\cdots,X_{j-1}} \mathrm{Var}_{X_j}(\log P(X_j | X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\leq \frac{1}{|S|^2} \sum_{i \in |S|} \sum_{j=1}^{n} \mathrm{Var}_{X_1,\cdots,X_j}(\log P(X_j | X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \mathrm{Var}_{X_1,X_2,\cdots,X_n}(\frac{1}{|S|} \sum_{i \in |S|} \log P(s^i))
\end{aligned}
$$

□

The fifth equation holds from the fact that $\mathbb{E}_X^2 \mathbb{E}_{Y|X}[f(X,Y)] = \mathbb{E}_{X,Y}^2[f(X,Y)]$. The result still stands after applying reweighting for the beam search.

E Shape Encoding Demonstration

In Figure 7, we show the code name on top of the image that it represents. c, s, and t represent circle, square and triangle respectively. The first two numbers represent the position of the shape in the canvas and the last number represents the size.

F Additional Test Output Examples for the 2D CAD Dataset

In this section, we include additional enlarged test outputs (Figure 8). We add the corresponding output program below each target/output pair. We observe that the algorithm approximates thin lines with triangles in some cases. Our hypothesis for the cause is the Chamfer distance reward function, which is a greedy algorithm and finds the matching distance based on the nearest features.

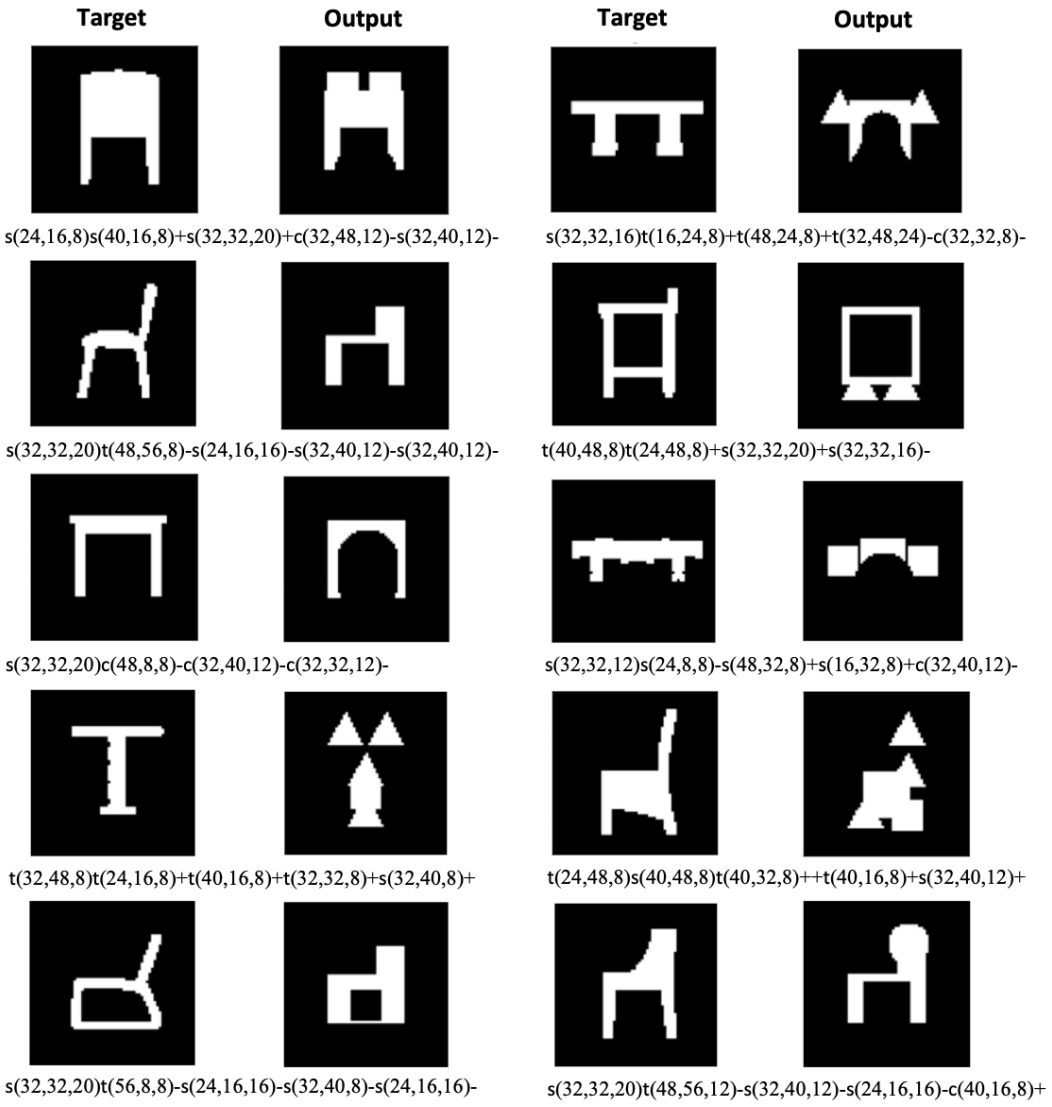


Figure 8: Additional test output with corresponding programs. The odd-numbered columns contain the target images and the images to their right are example outputs.