Denoising Diffusion for Sampling SAT Solutions

Anonymous Author(s)
Affiliation
Address
email

Abstract

Generating diverse solutions to the Boolean Satisfiability Problem (SAT) is a hard computational problem with practical applications for testing and functional verification of software and hardware designs. We explore the way to generate such solutions using Denoising Diffusion coupled with a Graph Neural Network to implement the denoising function. We find that the obtained accuracy is similar to the currently best purely neural method and the produced SAT solutions are highly diverse, even if the system is trained with non-random solutions from a standard solver.

1 Introduction

Boolean Satisfiability Problem (SAT) is a significant NP-complete problem with numerous practical applications – product configuration, hardware verification, and software package management, to name a few. There is no single efficient algorithm that solves every SAT problem, but heuristics have been developed that are well-suited to instances arising in practice. Here we consider an even harder task of sampling solutions of a SAT instance which is the key component for functional verification of software and hardware designs. Most SAT solvers return the same solution if run repeatedly therefore special approaches are needed to obtain several solutions, especially if it is required to sample more or less uniformly from the solution space. UniGen [Chakraborty et al., 2014, Soos et al., 2020] and QuickSample [Dutra et al., 2018] are some algorithms for this purpose. UniGen guarantees approximate uniformity of the sampled solutions but is very slow in certain cases. So, we are interested if the application of neural networks and denoising diffusion can produce diverse samples in less time.

Deep learning methods can accelerate solving NP-hard problems. Purely neural SAT solvers have been developed [Selsam et al., 2019, Amizadeh et al., 2019, Ozoliņš et al., 2021] that can solve small instances. Hybrid solvers that provide neural heuristics within existing solvers [Selsam and Bjørner, 2019, Han, 2020] are more practical for real-life instances. Also, deep learning methods are used for Combinatorial Optimization [Li et al., 2018, Bengio et al., 2021, Cappart et al., 2021] including MIP [Nair et al., 2020], TSP [Joshi et al., 2020] and VRP [Nazari et al., 2018].

Denoising diffusion [Yang et al., 2022], coupled with neural networks, provide a learnable way for sampling from a given distribution. Diffusion models have achieved state-of-the-art results for image generation [Ho et al., 2022, Vahdat et al., 2021, Rombach et al., 2021]. Diffusion has been applied to discrete binary [Sohl-Dickstein et al., 2015] and categorical [Hoogeboom et al., 2021b] data, text generation [Austin et al., 2021], symbolic music generation [Mittal et al., 2021], graph generation [Niu et al., 2020], and autoregressive modelling [Hoogeboom et al., 2021a]. But we are not aware of any attempt to use it for sampling SAT solutions.

In this paper, we apply denoising diffusion to sample solutions of SAT. We use Categorical Diffusion introduced by [Hoogeboom et al., 2021b] and couple it with QuerySat [Ozoliņš et al., 2021] graph neural network to implement the denoising function. We find that this approach achieves solving

Under review at the NeurIPS 2022 Workshop on Score-Based Methods. Do not distribute.

accuracy comparable to the state-of-the-art purely neural SAT solver (QuerySat) and finds diverse solutions of similar characteristics as UniGen up to 8000 times faster, in some cases.

2 Background: Diffusion Models

Given data x_0(variable assignment in our case), a diffusion model [Sohl-Dickstein et al., 2015] consists of predefined variational distributions $q(x_t|x_{t-1})$ that gradually introduces noise over time steps $t \in \{1, ..., T\}$. The diffusion trajectory is defined such that $q(x_t|x_{t-1})$ adds a small amount of noise around x_t. This way, information is gradually destroyed and at the final time step, x_T carries almost no information about x_0. A nice property of the diffusion process is that it can be reversed if the gradient of the distribution can be estimated which is often expressed as a function that denoises the data. Usually, the Normal distribution for noise is used which is simple to work with and produces excellent results for images [Ho et al., 2022] and sound [Kong et al., 2021].

To deal with the discrete SAT solution values, here we employ diffusion for categorical data, namely the Multinomial Diffusion [Hoogeboom et al., 2021b]. Having K categories (2 in our case – True and False), x_t is encoded as one-hot vector for each variable. The multinomial diffusion process is defined using a categorical distribution that has a small probability β_t of resampling a category uniformly and a large $(1 - \beta_t)$ probability of sampling the previous value x_{t-1}:

$$q(x_t|x_{t-1}) = \mathcal{C}(x_t|(1 - \beta_t)x_{t-1} + \beta_t/K), \tag{1}$$

where $\mathcal{C}$ denotes a categorical distribution with probability parameters after $|$. For such diffusion process the probability of any x_t given x_0 is expressed as:

$$q(x_t|x_0) = \mathcal{C}(x_t|\bar{\alpha}_t x_0 + (1 - \bar{\alpha}_t)/K), \tag{2}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{\tau=1}^{t} \alpha_\tau$. For the reverse distribution step, we follow the common practice to parametrize it using x_0. According to [Hoogeboom et al., 2021b], the distribution for the previous time step $t - 1$ can be computed from the value x_t at the next step and the initial value x_0 as:

$$\begin{gathered} q(x_{t-1}|x_t, x_0) = \mathcal{C}(x_{t-1}|\theta_{post}(x_t, x_0)), \text{where } \theta_{post}(x_t, x_0) = \tilde{\theta} / \sum_{k=1}^{K} \tilde{\theta}_k, \\ \text{and } \tilde{\theta} = [\alpha_t x_t + (1 - \alpha_t)/K] \odot [\bar{\alpha}_{t-1} x_0 + (1 - \bar{\alpha}_{t-1})/K] \end{gathered} \tag{3}$$

During the reverse process, an approximation $\hat{x}_0$ is used instead of x_0 which is produced by a neural network μ: $\hat{x}_0 = \mu(x_t, \bar{\alpha}_t)$. The neural network is trained by feeding it with the cumulative noise magnitude[1] $\bar{\alpha}_t$ and noisy sample x_t which is produced by the forward diffusion and asking the network to produce a clean sample $\hat{x}_0$.The loss function for training is the KL divergence between the true distribution and the predicted one:

$$KL(\mathcal{C}(\theta_{post}(x_t, x_0))|\mathcal{C}(\theta_{post}(x_t, \hat{x}_0))) \tag{4}$$

3 Background: GNNs for Solving SAT

We consider SAT formulas in conjunctive normal form and represent them as a bipartite variables-clauses graph, likewise known as an SAT factor graph Biere et al. [2009]. In such a graph, the edge between variables and clauses graph exists whenever the variable is present in the clause. Two types of edges are used to distinguish a variable from its negation. Variables-clauses graph of SAT formula with n variables and m clauses can be represented as a sparse $n \times m$ adjacency matrix.

Graph Neural Networks(GNN), especially message-passing networks, are commonly used for solving graphs-based tasks. Such network architectures are simple, yet have strong inductive bias over the graph Zhou et al. [2018]. For boolean satisfiability solving, the currently best neural architecture is QuerySat [Ozoliņš et al., 2021] which is capable of predicting solutions to random 3-SAT formulas up to 400 variables. QuerySat is a recurrent GNN, specially tailored for SAT solving, which at each

[1] Hoogeboom et al. [2021b] parametrize the neural network with t, instead. This is equivalent once we fix the noise schedule.

step exchanges the information between variables and clauses, gives a candidate prediction, and evaluates how far it is from a valid solution. It is trained with 32 recurrent iterations but at inference time the number of iterations can be increased to yield better results. The prediction at the final step is used as the result. We use QuerySat architecture with 32 recurrent iterations as the μ network for denoising.

4 Diffusion for SAT Sampling

To train the neural network μ, we generate a dataset with SAT formulas and their solutions. The solution is one-hot encoded and treated as x_0 but the SAT instance is used to build the GNN network so the μ function gets conditioned on the given instance. Then for each example we randomly sample $\bar{\alpha}_t$ according to noising schedule $\bar{\alpha}_t = 1 - \sqrt{t/T}$ and obtain x_t by sampling $q(x_t|x_0)$ as given in Eq. 2. The neural network $\mu(x_t, \bar{\alpha}_t)$ is trained using KL divergence given by Eq. 4 to estimate x_0.

When the model is trained, we can use it for sampling a solution to the given SAT formula. We employ the standard denoising diffusion algorithm. Given the total number of steps T, the algorithm works backward from step T toward step 1. The distribution x_T is created with equal probabilities of True and False for each variable and then at each step t a sample from it is drawn which is presented to the neural network μ. The neural network returns the estimated probabilities of SAT variable assignment $\hat{x}_0$. The value of the previous timestep x_{t-1} is obtained by sampling the distribution $q(x_{t-1}|x_t, \hat{x}_0)$ given by Eq. 3. The algorithm returns the assignment $\hat{x}_0$ with argmax applied to each variable.

5 Evaluation

We evaluate the proposed SAT sampling algorithm on two datasets: random 3-SAT and 3-Clique instances. The 3-SAT instances consist of random clauses of 3 literals at the satisfiability threshold i.e with the clause/variable ratio of about 4.3. We also test on 3-SAT instances with clause/variable ratio 3. The 3-Clique instances represent SAT encoding of the task to find triangles in a graph. See Appendix A for the dataset details.

At first, let us evaluate the accuracy of the produced solutions i.e. the fraction of the instances that are fully solved. We consider an instance solved if a solution is found at any of the diffusion steps, this increases our chances to spot a solution since a great amount of randomness is involved in the diffusion process. We compare our solution, named DiffusionSat, with QuerySat. For DiffusionSat we consider two choices for the number of diffusion steps – 32 or 128. For QuerySat we use two setups of 32 or 4K recurrent iterations. Note that the amount of computing for QuerySat 4K is roughly the same as for DiffusionSat 128. Since the small instances used for training are solved almost perfectly, for accuracy evaluation, we use about 4x larger ones.

We consider two solvers for generating the solutions that are used for DiffusionSat training – Glucose4 [Audemard and Simon, 2018] and UniGen3 [Soos et al., 2020]. Glucose is a standard SAT solver that does not guarantee any particular distribution of the produced solutions but UniGen provides approximately uniformly distributed solutions. Diffusion models are intended to reproduce the same distribution of samples that was supplied for training, so we may get different results in both cases.

The accuracy results are given in Table 1. It depicts the average and the standard deviation of 5 test runs. We see that QuerySat is better for 3-SAT but DiffusionSat is better for 3-Clique. Notably, DiffusionSat 128 achieves 100% solved instances of 3-Clique where QuerySat 4K gets about 95% while both these cases use the same amount of computing. Also, we can notice that increasing the number of diffusion steps increases the accuracy.

To evaluate sample diversity, we produce 100 solutions of each instance and count how many of them are unique. The evaluation is performed on 3-SAT instances of size 5–100 variables and on 3-Clique instances obtained from graphs with 4 to 40 vertices. Our first evaluation revealed that 3-SAT at the satisfiability threshold and 3-Clique instances have only a few solutions – they have been designed to be maximally hard and are not well-suited to evaluate diversity. So, we included a dataset with 3-SAT instances with clause/variable ratio of 3 which have many solutions.

Table 2 shows the solution diversity. We see that QuerySat produce only a few different solutions, although it employs some randomness internally. The best results are produced by DiffusionSat, even

Table 1: Test accuracy (higher is better) as percent of fully solved instances.

Method	3-SAT		3-Clique
	cl/var ≈ 4.3	cl/var = 3.0	
QuerySat (32 iterations)	61.9 ± 5.2	**100** ± 0	82.0 ± 4.7
QuerySat (4K iterations)	**93.3** ± 3.2	**100** ± 0	94.7 ± 4.6
DiffusionSat Glucose (32 steps)	45.3 ± 0.9	95.2 ± 0.4	67.5 ± 2.5
DiffusionSat UniGen (32 steps)	40.8 ± 0.9	99.9 ± 0.0	99.1 ± 0.1
DiffusionSat UniGen (128 steps)	47.1 ± 0.7	**100** ± 0	**100** ± 0

Table 2: Percentage of unique solutions (higher is better) from 100 samples.

Method	3-SAT		3-Clique
	cl/var ≈ 4.3	cl/var = 3.0	
QuerySat (32 iterations)	6.7 ± 1.1	6.3 ± 0.7	–
DiffusionSat Glucose (32 steps)	56.6 ± 7.1	95.0 ± 3.1	5.9 ± 0.2
DiffusionSat UniGen (32 steps)	**61.5** ± 7.6	**95.9** ± 2.8	2.6 ± 0.2
UniGen	49.4 ± 3.0	92.6 ± 5.9	1.4 ± 0.1

surpassing UniGen. Interestingly, there is not much difference whether DiffusionSat has been trained with diverse solutions or with those coming from a standard SAT solver – both produce slightly better diversity than UniGen.

To get more insight how diverse are the solutions, we inspect how many variables are equal in two solution samples on 3-SAT instances, see Appendix C. For the clause/variable ratio of 4.3 about 85% variables are equal and for ratio 3 – about 65%. Note that completely random assignment would yield 50% but that may not be a valid solution. We find that DiffusionSat and UniGen perform pretty similarly regarding this metric.

We evaluated the time taken to produce one solution sample on 3-SAT instance with 100 variables. DiffusionSat (with 32 diffusion steps) processes one batch of about 95 instances in 5.7 seconds. That means, one instance takes 0.06s. UniGen takes about 0.22s per instance of clause ratio 4.3 but much longer – about 500s (with high variance from 70s to 1300s) for clause ratio 3. When compared, DiffusionSat produces a solution more than 8000 times faster for the latter case (although using GPU and batched instances).

6 Conclusion

We have shown how to combine denoising diffusion with GNN and apply it for generating SAT solutions. From theoretical point of view, if trained with uniformly distributed samples and the neural network being able to perfectly learn the denoising function, the samples produced by diffusion are also uniformly distributed. Both these requirements are hard to satisfy in practice, but we have shown that DiffusionSat generates highly diverse samples of similar characteristics than UniGen which is known to generate nearly uniform distribution. Moreover, there is no need to use uniform distribution for training, as training on solutions from a standard SAT solver gives equally good results. Besides, for tricky instances (3-SAT formulas with clause/variable ratio 3), DiffusionSat is much faster that UniGen.

There are some limitations of this work: (a) We have tested only two properties of diversity, more thorough tests should be done to draw conclusions about uniformity. Also, more datasets should be considered. (b) Training sets with larger instances may be infeasible to generate, QuerySat has an advantage here, not requiring to know the solutions for training. (c) DiffusionSat is an incomplete solver – it cannot prove that an instance is UNSAT. (d) Only SAT instances up to 50K clauses can be solved with the current A6000 GPU(with 48GB memory).

References

S. Amizadeh, S. Matusevych, and M. Weimer. PDP: A general neural framework for learning constraint satisfaction solvers. *arXiv preprint arXiv:1903.01969*, 2019.

G. Audemard and L. Simon. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(01):1840001, 2018.

J. Austin, D. Johnson, J. Ho, D. Tarlow, and R. van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34, 2021.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

A. Biere, M. Heule, and H. van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

S. Chakraborty, K. S. Meel, and M. Y. Vardi. Balancing scalability and uniformity in sat-witness generator. In *Proceedings of Design Automation Conference (DAC)*, pages 60:1–60:6, 6 2014.

J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial intelligence*, 81(1-2):31–57, 1996.

R. Dutra, K. Laeufer, J. Bachrach, and K. Sen. Efficient sampling of sat solutions for testing. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 549–559. IEEE, 2018.

J. M. Han. Enhancing SAT solvers with glue variable predictions. *arXiv preprint arXiv:2007.02559*, 2020.

J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.

E. Hoogeboom, A. A. Gritsenko, J. Bastings, B. Poole, R. v. d. Berg, and T. Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021a.

E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34, 2021b.

C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent. Learning TSP requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro. DiffWave: A Versatile Diffusion Model for Audio Synthesis. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=a-xFK8Ymz5J`.

M. Lauria, J. Elffers, J. Nordström, and M. Vinyals. CNFgen: A generator of crafted benchmarks. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 464–473. Springer, 2017.

Z. Li, Q. Chen, and V. Koltun. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/8d3bba7425e7c98c50f52ca1b52d3735-Paper.pdf`.

G. Mittal, J. Engel, C. Hawthorne, and I. Simon. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac. Reinforcement Learning for Solving the Vehicle Routing Problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf`.

C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 4474–4484. PMLR, 2020.

E. Ozoliņš, K. Freivalds, A. Draguns, E. Gaile, R. Zakovskis, and S. Kozlovičs. Goal-Aware Neural SAT Solver. *arXiv preprint arXiv:2106.07162*, 2021.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. *arXiv preprint arXiv:2112.10752*, 2021.

D. Selsam and N. Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 336–353. Springer, 2019.

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=HJMC_iA5tm`.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

M. Soos, S. Gocht, and K. S. Meel. Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling. In *Proceedings of International Conference on Computer-Aided Verification (CAV)*, 7 2020.

A. Vahdat, K. Kreis, and J. Kautz. Score-based generative modeling in latent space. *Advances in Neural Information Processing Systems*, 34, 2021.

L. Yang, Z. Zhang, and S. Hong. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796*, 2022.

J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods andapplications. *arXiv preprint arXiv: 1812.08434*, 2018.

J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806, 2020.

A Appendix: Datasets

3-Clique and 3-SAT tasks are generated using the CNFGen library Lauria et al. [2017], which allows encoding several popular problems as SAT instances. We generate hard 3-SAT instances at the satisfiability boundary where the relationship between the number of clauses (m) and variables (n) is $m = 4.258n + 58.26n^{-\frac{2}{3}}$ Crawford and Auton [1996]. The train and validation sets of 3-SAT tasks consist of formulas with 5 to 100 variables. Since the small instances used for training are solved almost perfectly, for accuracy evaluation, we use larger ones – up to 405 variables. Such larger test instances were also used in the QuerySat paper [Ozoliņš et al., 2021] and show the generalization ability of the methods. The test set for sample diversity evaluation has instances of the same size as used for training, i.e. with 5 to 100 variables. For diversity evaluation, we have another test set with clause-to-variable ratio $m = 3n$. These instances are much easier and have many solutions.

For the 3-Clique task, we generate Erdős–Rényi graphs with edge probability p. The goal of this task is to find all triangles in the graph, we use $p = 3^{\frac{1}{3}}/(v(2 - 3v + v^2))^{\frac{1}{3}}$, where v is the vertex count in the graph. Such generation produces graphs in which a few triangles are expected. The graphs are then encoded as SAT instances using the CNFGen library. Train and validation sets consist of SAT encodings of graphs with 4 to 40 vertices. They end up encoded into SAT formulas of sizes up to 120 variables and 7K clauses. The test set for accuracy evaluation has graphs with 4 to 100 vertices. The test set for diversity evaluation has graphs with 4 to 40 vertices.

B Appendix: Implementation and Training

We have implemented DiffusionSat in Tensorflow based on the official code of QuerySat[2]. From there, we employ the GNN neural backbone to implement the μ network, the dataset generation and accuracy evaluation code. So, the accuracy results of DiffusionSat and QuerySat are obtained in the same conditions and are directly comparable. The backbone network that is employed inside DiffusionSat has 32 recurrent iterations that is the same as used for training QuerySat.

For generating SAT solutions we use Glucose4 solver from python-sat[3] library and UniGen3[4]. Only satisfiable instances are used in training.

Training is performed for 166K steps with AdaBelief optimizer [Zhuang et al., 2020] having learning rate 3e-3. Several instances are batched together to increase training and inference speed. Training on a single Nvidia RTX A6000 GPU takes about 1 day.

To evaluate 3SAT instances with clause/variable ratio 3, we do not perform separate training, we use the same model trained for the 4.3 ratio – it works very well.

[2]https://github.com/LUMII-Syslab/QuerySAT

[3]https://pypi.org/project/python-sat/

[4]https://github.com/meelgroup/unigen

C Appendix: Additional Results

We inspected how many variables are equal in two solution samples on 3-SAT instances. The results for DiffusionSat and UniGen are quite similar.

Table 3: Percentage how many variables are equal between two solution samples (lower is better). Random variable assignment would yield 50%.

Method	3-SAT	
	cl/var ≈ 4.3	cl/var = 3.0
DiffusionSat Glucose (32 steps)	86.0 ± 1.7	62.4 ± 6.6
DiffusionSat UniGen (32 steps)	86.4 ± 2.0	65.0 ± 5.8
UniGen	83.1 ± 16.4	63.2 ± 7.4