GRAPHGUIDE: INTERPRETABLE AND CONTROLLABLE CONDITIONAL GRAPH GENERATION WITH DISCRETE BERNOULLI DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion models achieve state-of-the-art performance in generating realistic objects and have been successfully applied to images, text, and videos. Recent work has shown that diffusion can also be defined on graphs, including graph representations of drug-like molecules. Unfortunately, it remains difficult to perform conditional generation on graphs in a way which is interpretable and controllable. In this work, we propose GraphGUIDE, a novel framework for graph generation using diffusion models, where edges in the graph are flipped or set at each discrete time step. We demonstrate GraphGUIDE on several graph datasets, and show that it enables full control over the conditional generation of arbitrary structural properties without relying on predefined labels. Our framework for graph diffusion can have a large impact on the interpretable conditional generation of graphs, including the generation of drug-like molecules with desired properties in a way which is informed by experimental evidence.

1 INTRODUCTION

Diffusion models have rapidly become the state-of-the-art method for generating many kinds of data Sohl-Dickstein et al. (2015); Ho et al. (2020); Song et al. (2021), including images and videos Dhariwal & Nichol (2021); Ho et al. (2022), text Li et al. (2022), and tabular data Kotelnikov et al. (2022). In order to generate objects from some distribution $q_0(x)$, a diffusion model defines a *forward-diffusion* process which iteratively adds noise (typically continuous Gaussian noise) to an object $x_0 \sim q_0(x)$. The model then learns a *reverse-diffusion* process to iteratively denoise a diffused object back to the original x_0. This allows the model to effectively sample new objects from $q_0(x)$.

The most useful and impactful developments in diffusion models, however, arguably stem from the ability to *conditionally* generate objects—that is, generating objects x_0 which satisfy some desired property. There are a few methods for conditional generation today, which effectively supply the model with a label y (out of a predefined set) to steer the reverse-diffusion process toward generating an object x_0 which satisfies the property defined by y (Section 2). Using these methods, many of the most prominent recent works in diffusion models have centered on conditional generation; these include image generation by class Song et al. (2021); Dhariwal & Nichol (2021) and image/video synthesis from text Rombach et al. (2021); Ho et al. (2022).

Although diffusion models have been successful in generating and conditionally generating many different data types, applying these models to generating *graphs* poses several somewhat unique challenges, particularly for conditional generation. These challenges are major obstacles for many real-world problems, such as drug discovery, where it is critically important to be able to generate molecular graphs which have certain desired physiological or chemical properties. Although methods exist today for generating graphs using diffusion models Niu et al. (2020); Jo et al. (2022); Vignac et al. (2022), they are limited in several ways.

One such limitation is that many current graph-diffusion methods are *continuous* and generate reverse-diffusion intermediates which are not well-defined discrete graphs; instead, their intermediates include fractional or negative-valued edges. This severely hinders the interpretability and controllability of these methods; after all, it is not easy to decipher what -0.2 in an adjacency matrix means, or how to control generation where the probability of an edge is 0.6.

A bigger set of issues lies in the current available methods for *conditional generation*, which are rather limiting for graph diffusion (including for diffusion frameworks which are discrete). Firstly, current conditional-generation methods supply a label y to influence reverse diffusion as a soft constraint. This renders conditional generation an inscrutable process which prevents humans from interpreting or controlling the generated graph sample (this is even worse for continuous-diffusion frameworks). For many real-world applications such as drug discovery, being able to very precisely control the generated outputs for specific features (e.g. functional groups) is an important feature for a generative algorithm. Secondly, current conditional-generation methods rely on a predefined set of possible labels to condition on, and the addition of new properties necessitates the retraining of all or part of the model. For many graph applications, this is particularly limiting, as there are a huge number of specific structural properties that may be conditioned on. For example, in the fields of drug discovery and molecule design, it is not uncommon to desire a molecular graph which might contain a benzene, or toluene, or aniline, etc. Thus, for many real-world graph-generation problems such as drug design, both these limitations in current conditional-generation methods can pose a serious obstacle that prevents more widespread adoption of graph diffusion.

In this work, we present GraphGUIDE (**Graph G**eneration **U**sing **I**nterpretable **D**iffusion on **E**dges), an alternative graph-diffusion framework that addresses these limitations. GraphGUIDE relies on a diffusion process which is fully discrete, defined by flipping edges in and out of existence based on a Bernoulli random process. We define three different diffusion kernels, which add, delete, or flip edges randomly. Thus, at each point in both forward and reverse diffusion, the intermediate is an interpretable, well-defined graph. More importantly, this allows for *full, interpretable control* of conditional generation using the appropriate kernel *without any reliance on a predefined set of labels*.

To summarize, our main contributions are the following:

- We present a novel framework, GraphGUIDE, for interpretable and controllable graph generation based on discrete diffusion of graph edges.
- We derive three discrete diffusion kernels based on the Bernoulli distribution, including a more efficient and stable parameterization of a kernel based on symmetrically flipping bits, and two novel kernels based on asymmetrically setting bits.
- We compare our method to other recent state-of-the-art graph-generation methods on benchmark datasets, showing that it achieves comparable generation quality.
- We demonstrate that GraphGUIDE enables full control of arbitrary structural properties—which do not need to be predefined at training time—during generation, thus allowing the injection of custom priors such as the presence or absence of specific graph motifs.

2 Related work

2.1 Conditional generation in diffusion models

A major goal of generative modeling is to perform *conditional generation*, where we wish to draw a sample from $q_0(x)$ which satisfies a specific label or property. Within the diffusion-model literature, there are effectively two methods for conditional generation.

Classifier-guided conditional generation was proposed in Song et al. (2021), in which an external classifier $f(x_t)$ is trained on x_t to predict some label y. Input gradients from this classifier are then used during the generative process to bias the generation of an object toward one which has the label y. While elegant in its mathematical justification (a simple invocation of Bayes' Rule), it relies on an external classifier which is trained on noisy inputs x_t from across the diffusion timeline and a predefined set of labels y. This method is also only readily applied to diffusion models trained in a continuous-time and continuous-noise setting, due to its reliance on gradients.

In contrast to classifier-guided conditional generation, Ho et al. (2021) proposed an alternative method: classifier-free conditional generation. Instead of relying on an external classifier, the neural network p_θ (which defines the reverse-diffusion/generative process) is trained with labels as an input: $p_\theta(x_t, t, y)$. This method for conditional generation has been exceedingly popular, and has been shown to generate state-of-the-art samples Rombach et al. (2021); Ho et al. (2022). Unlike

classifier-guided conditional generation, this method enjoys the freedom of not relying on any external classifier, and it can be applied to discrete-time and/or discrete-noise diffusion settings.

Unfortunately, both methods for conditional generation suffer from some limitations. Firstly, both methods merely supply a reverse-diffusion-influencing signal to the generative process (i.e. through biasing gradients or through an auxiliary input). This acts as a *soft constraint* which not only is uninterpretable, but also cannot be controlled or modified manually by a human during reverse diffusion. Secondly, both methods require a predefined set of labels y (to train either an external classifier or the diffusion model itself). The addition of new labels or properties to conditionally generate would necessitate the retraining of an entire model; in the more popular case of classifier-free conditional generation, the entire diffusion model would need to be retrained.

2.2 Discrete diffusion kernels

The vast majority of diffusion models are trained with Gaussian noise, as this is the most well-developed kernel. Methods for discrete diffusion, however, have been proposed and utilized.

In the paper which first demonstrated diffusion models, Sohl-Dickstein et al. (2015) briefly proposed a discrete diffusion kernel based on the Bernoulli distribution—termed the "binomial" kernel—which flipped black-and-white pixels back and forth according to some probability. Unfortunately, the binomial kernel was parameterized in a way which made it inefficient for forward diffusion and unstable for reverse diffusion. Because of the large focus on Gaussian kernels in the literature, this discrete kernel has remained underdeveloped and unused until this point, and until now has not been reparameterized to behave efficiently and stably.

Later on, Hoogeboom et al. (2021) proposed an alternative way to diffuse over discrete states based on the multinomial distribution, where forward diffusion transforms a one-hot-encoded category vector into a uniform multinomial distribution. Austin et al. (2021) then proposed a similar framework where forward diffusion is defined a discrete Markov process; the authors showed that multinomial diffusion is a special case of this framework. Although these methods for discrete diffusion have been successfully applied to unconditional generation, there is very limited usage of these kernels for conditional generation (in graphs or other data types).

2.3 Graph generation

For the problem of graph generation, there is already a large body of literature which spans many techniques, from autoregressive to one-shot. This includes GraphRNN You et al. (2018), GRAN Liao et al. (2019), MolGAN Cao & Kipf (2018), and SPECTRE Martinkus et al. (2022), among others.

For graph generation using diffusion models specifically, a few methods have been proposed, each of which solves the problem of graph discreteness differently. Niu et al. (2020) applied continuous diffusion on the adjacency matrix using Gaussian noise—a relaxation of the discreteness of edges. This generated graph-diffusion intermediates which had fractional and negative edges. Lee et al. (2022) then adapted this framework and performed conditional generation on molecular graphs by incorporating gradients from an external property-prediction network—an application of classifier guidance, made possible by the continuous relaxation Song et al. (2021). Methods such as these, however—which diffuse on adjacency matrices—are severely hindered in their ability to conditionally generate structural properties. There are many equivalent adjacency-matrix orderings which satisfy the same structural property, which makes it difficult to inject this form of inductive bias into the generative process. Of course, these methods also suffer from the aforementioned limitations inherent to classifier-guided conditional generation.

Later on, Vignac et al. (2022) applied the Markovian diffusion framework proposed by Austin et al. (2021) on discrete graph adjacency matrices and node features. The authors showed that their method—DiGress—achieved state-of-the-art generative performance compared to other graph-generation methods, including those listed above. The authors also attempted to apply classifier-guided conditional generation using a discretized approximation of gradients, although this was limited by the conflict between continuous gradients and discrete diffusion, as well as the limitations inherent to current conditional-generation methods (i.e. they are uninterpretable soft constraints which require predefining a set of labels at training time).

3 GRAPHGUIDE FOR CONDITIONAL GRAPH GENERATION

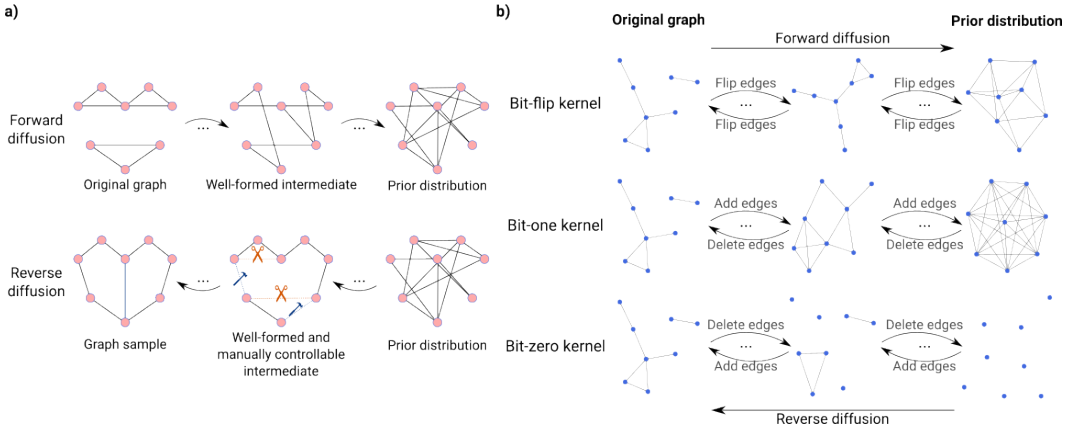

Figure 1: Interpretable and controllable graph diffusion. **a)** In the GraphGUIDE framework, we forward diffuse graphs by flipping edges in and out of existence. In reverse diffusion, if a certain set of edges or graph motifs are desired (or undesired), appropriate edges may be manually added (or pruned, respectively) to generate a graph sample with the desired property. **b)** We propose three Bernoulli diffusion kernels which flip edges on or off randomly (top), add edges randomly (middle), or delete edges randomly (bottom).

Previous work in graph diffusion created uninterpretable diffusion intermediates, or they lacked a component which allowed for successful control over conditional generation. Here, we propose GraphGUIDE as an alternative framework for graph generation which allows for more interpretable and controllable generation (Figure 1a).

In this section, we present discrete diffusion kernels for graph generation (Section 3.1), demonstrate that they are capable of achieving generative performance comparable to other state-of-the-art methods (Section 3.2), and showcase the ease at which graph generation can be controlled for arbitrary structural properties using our GraphGUIDE framework (Section 3.3).

3.1 BERNOULLI DIFFUSION ON EDGES

In order for GraphGUIDE to diffuse on graphs in a discrete and controllable manner, We define three discrete diffusion kernels based on the Bernoulli distribution (Figure 1b). Consider a binary vector x_t to be diffused. The diffusion kernels will alter the bits randomly until a prior distribution is reached. Importantly, instead of performing diffusion on some continuous states, we define the Bernoulli diffusion kernels to *directly* flip or set bits, thereby ensuring that every x_t in the forward-diffusion process is a fully well-defined binary vector. We define three diffusion kernels for this process:

1. Bit-flip kernel: at time t, flip each bit with probability β_t.
2. Bit-one kernel: at time t, set each bit to 1 with probability β_t (if the bit is already 1, do nothing).
3. Bit-zero kernel: at time t, set each bit to 0 with probability β_t (if the bit is already 0, do nothing).

For all three kernels, we assume there is a fixed noise schedule β_t with $t \in \{1, ..., T\}$. A typical noise schedule monotonically increases from $\beta_1 = 0$ to $\beta_T = \frac{1}{2}$. Note that the forward-diffusion process defines an independent Bernoulli distribution for each entry in x_t at every time t.

For each of these three kernels, we derive an analytical formula for the marginal forward distribution $q(x_t|x_0)$ and for the marginal posterior distribution $q(x_{t-1}|x_t, x_0)$. Importantly, we derive parameterizations for these kernels which satisfy two properties: 1) $q(x_t|x_0)$ is also a Bernoulli distribution for any t, and it is very tractable to compute its parameters; and 2) the reverse-diffusion posterior

distribution $q(x_{t-1}|x_t, x_0)$ is also a tractable Bernoulli distribution, so the reverse-diffusion process can be modeled by learning the posterior's parameters directly, leading to more stable behavior. These two properties are critical for the efficient training of diffusion models which stably generate high-quality samples Ho et al. (2020).

We derive closed-form formulae for our three Bernoulli kernels, including the forward distribution, the posterior distribution, and the prior distribution (Supplementary Table S1). See Appendix B for derivations.

These three Bernoulli kernels are defined on general bits. In order to generate undirected graphs, we define x_t as a binary vector denoting which edges exist in the graph (1 if the edge exists, and 0 otherwise). This vector has size $\binom{n}{2}$, where n is the number of nodes in the graph (we do not allow self-edges or multi-edges, but our work can be extended easily to accommodate both cases; see Section 4). We diffuse on an undirected graph by applying a Bernoulli kernel to this binary edge vector. That is, the forward-diffusion process adds or removes edges (or both) randomly, and the reverse-diffusion process reconstructs a graph by deciding which nodes to link or unlink. The precise behavior depends on which of the three kernels is being used.

When applied to graph edges, the bit-flip kernel slowly flips edges in and out of existence with increasing probability until the graph approaches a prior which is the Erdös–Renyi graph (with $p = 0.5$ for a typical noise schedule). In reverse diffusion, the bit-flip kernel starts with an Erdös–Renyi graph and iteratively toggles edges on and off until a final graph sample is recovered. The bit-one kernel slowly adds edges randomly to the graph until reaching the prior, which is the complete graph (i.e. all possible edges exist). In the reverse direction, bit-one diffusion successively removes edges until obtaining a graph sample. Finally, the bit-zero kernel slowly removes random edges, approaching the prior of the empty graph (i.e. no edges exist at all). In reverse diffusion with the bit-zero kernel, edges are slowly added until a final graph sample is formed. In our experiments below, we focus our work on discrete diffusion over edge existence alone, as this is sufficient for many applications, including for molecule design (See Section 4 for more details).

3.2 Generative performance

We compare the generative performance of graph-diffusion models trained with our Bernoulli kernels compared to other graph-generation methods. Over two well-known benchmark datasets (community-small and stochastic block models), we use the maximum-mean-discrepancy (MMD) metric to quantify how similar the generated graphs are to the training set (Table 1). We compute the MMD between the distributions of degrees, clustering coefficients, and orbit counts Hočevar & Demšar (2014). We report an MMD ratio: the MMD of generated graphs and the training set, normalized by the MMD of the training set and an independently sampled validation set. We compare our MMD ratios to those reported by other graph-generation methods (when available). A lower MMD is better. Note that in some cases, these other methods erroneously label MMD *squared* as MMD in their text, whereas we report MMD (i.e. values taken from other works which report MMD squared have been square rooted).

Table 1: Bernoulli edge diffusion MMD ratio

	Community-small			**Stochastic block models**		
Method	Deg. ↓	Clus. ↓	Orbit ↓	Deg. ↓	Clus. ↓	Orbit ↓
GraphRNN	2.00	1.31	2.00	2.62	1.33	1.75
GRAN	1.73	1.25	1.00	3.76	1.29	1.46
MolGAN	1.73	1.36	1.00	5.42	1.87	1.67
SPECTRE	1.00	1.73	1.00	3.14	1.26	**0.54**
DiGress	1.00	0.95	**1.00**	1.26	1.22	1.30
Bit-flip	**0.99**	**0.58**	2.55	2.73	1.23	0.94
Bit-one	1.21	0.62	1.83	**1.00**	1.21	0.81
Bit-zero	1.87	1.02	4.69	1.31	**1.19**	0.80

These experiments demonstrate that diffusion using the discrete Bernoulli kernels on graph edges achieves comparable performance compared to other state-of-the-art methods for graph generation, including other discrete graph-diffusion methods such as DiGress Vignac et al. (2022).

3.3 GRAPH GENERATION WITH INTERPRETABLE CONTROL

GraphGUIDE employs these discrete Bernoulli kernels because they result in perfectly well-defined graphs at each intermediate stage of the diffusion process. Not only are these intermediates more readily interpretable, but they also allow the generation process to be easily controlled. At any stage of reverse diffusion, edges or graph motifs that are desired can be manually retained in the intermediate (or symmetrically, edges or motifs that are not desired can be prevented from forming).

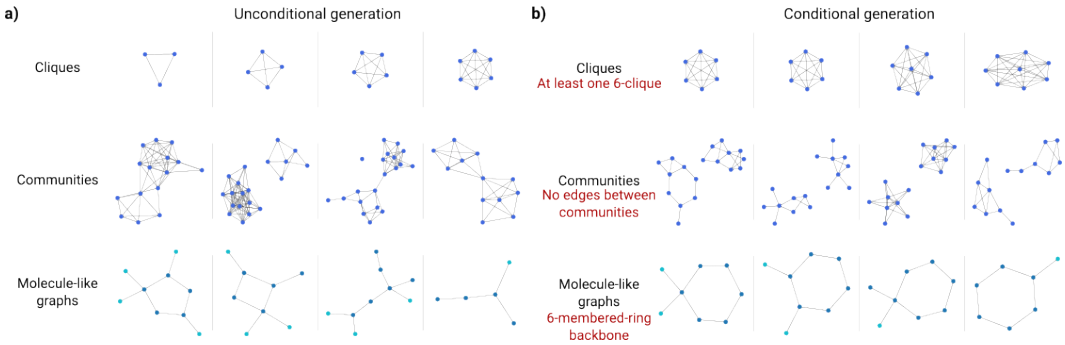

Figure 2: Examples of generated graphs with and without manual control via GraphGUIDE. On three different graph datasets, we show examples of graphs generated **a)** unconditionally, or **b)** using the GraphGUIDE framework while controlling for a desired property (listed in red).

In order to illustrate the ease at which graph generation can be controlled with GraphGUIDE, we show some example graphs generated to have specific desired properties (Figure 2). First, we trained a model using the bit-one kernel to generate cliques of sizes three to six. Graphs that were generated unconditionally (i.e. without manual control) contained cliques of various sizes (Figure 2a, top). We then *conditionally* generated graphs by manually controlling the reverse-diffusion process so that all generated graphs would have a clique of size 6 (Figure 2b, top). Recall that the bit-one kernel confers a prior distribution which is the complete graph (i.e. x_T corresponds to a graph with all possible edges). In the reverse-diffusion process, edges are gradually removed to recover a sample of x_0. For each graph generated under manual control, we arbitrarily selected 6 nodes. Throughout the reverse-diffusion process—at each step—we ensured that no edges were removed between any of these 6 nodes. If any such edges were removed at a reverse-diffusion step, they were added back before the next step. As a result, all graphs generated using this procedure contained a 6-clique. Intriguingly, the model also extrapolated from the data and generated 7- and 8-cliques, which contain 6-cliques as subgraphs.

We then trained a model using the bit-zero kernel to generate community-small graphs (as in Table 1). Graphs generated by this model typically showed two communities which were oftentimes linked to each other with one or more edges (Figure 2a, middle). We then controlled generation by ensuring no edges were added between the two communities, thereby always forming two disjoint subgraphs (Figure 2b, middle). The bit-zero kernel confers a prior distribution which has no edges, and in the reverse-diffusion process, edges are slowly added to form the final graph sample x_0. In order to perform manual control, we simply partitioned the empty graph x_T into two equal-sized sets of nodes, and ensured that no edges were ever added between nodes of different communities. The result is a set of graphs where the two communities were always disjoint.

Finally, we trained a model using the bit-flip kernel to generate a set of molecule-like graphs. Our molecule-like graphs consist of backbone and secondary nodes (atoms), where backbone nodes always have a degree of at most 4, and secondary nodes always have a degree of 1 and are linked to a backbone node. In terms of real organic chemistry, backbone nodes might correspond to carbon atoms, and secondary nodes might correspond to halogens. The backbone atoms can be linked together in rings of various sizes, or be acyclic and branched. Graphs generated from this model showed a diverse

set of molecule-like structures (Figure 2a, bottom). We then performed controllable generation and enforced throughout reverse diffusion that a 6-membered backbone ring would be formed (Figure 2b, bottom). The bit-flip kernel confers a prior distribution x_T which is an Erdös–Renyi graph, and edges are slowly removed or added throughout the reverse-diffusion process to generate a graph sample x_0. In order to perform manual control, we identified 6 backbone nodes from x_T, and enforced throughout the generative process that those 6 backbone nodes were linked together in a ring, with no other edges between them. The resulting graphs always had a 6-membered backbone ring, usually with other backbone nodes or secondary nodes attached.

Throughout these experiments, because of the full manual control offered by the discrete diffusion, **100%** of the resulting graphs always had the desired property. In contrast, when we unconditionally generated graphs from these models, we only obtained 6-cliques 68% of the time, disjoint communities 21% of the time, and 6-membered-ring molecules 12% of the time.

In this section, we demonstrated the power of conditional generation via manual control with GraphGUIDE. Notably, this was made possible by the fully discrete nature of the Bernoulli kernels, which require that all diffusion intermediates are fully well-defined graphs. This is not only important for interpretability and for humans to easily manipulate the diffusion process, but is also critical for the robustness of the neural-network predictions. Just as the neural network is trained on well-defined graphs with binary edges, manually edited reverse-diffusion intermediates are also well-defined graphs with binary edges, and therefore are more likely in-distribution for the neural network, thus not leading to unexpected or undefined behavior. Furthermore, we presented *three* Bernoulli kernels, each of which is well-suited for different controllable-generation tasks, where edges or motifs need to be retained or removed. The bit-one kernel is best for ensuring that a particular set of edges are retained; the bit-zero kernel is best for ensuring that a particular set of edges are removed; the bit-flip kernel is best for ensuring that a more complex motif (with some edges that need to be retained and others removed) forms. Using the best-suited kernel of the three helps ensure that manually controlled generation creates intermediates that are in-distribution for the model. For example, using the bit-one kernel generates a prior which is the complete graph, and so in the reverse-diffusion process, edges are gradually deleted. If the manual-control task is to retain a set of edges, then retaining those edges throughout the reverse-diffusion process ensures that diffusion intermediates are much more likely to remain in-distribution for the neural network.

Contrast this interpretability and controllability with other graph-diffusion frameworks. On continuous-diffusion frameworks, the diffusion intermediates contain edges that are fractional or negative. Not only is this far less interpretable, it is also much more difficult to control the generation in terms of desired structural features, such as the presence or absence of a particular motif. Any attempt to manually control the generative process like above would be foiled by the fact that fully well-defined graphs are not in-distribution for most of the diffusion process. Additionally, all existing graph-diffusion frameworks (continuous or discrete) rely on traditional methods for conditional generation, which inscrutably influence graph generation using a limited, predefined set of labels (Section 2).

Our experiments highlight the unique method of conditional generation offered by GraphGUIDE. Conditional generation using GraphGUIDE no longer requires an external classifier, or even a predetermined set of property labels. Because the diffusion intermediates are so easily manipulated, practically any structural property—which is defined by the presence and/or absence of edges or motifs—can be imposed with full manual control at any time in the reverse-diffusion process. This very much distinguishes our framework from other methods for conditional generation.

4 Discussion

The diffusion framework proposed in this work is unique from other diffusion methods due to the controllability of the reverse-diffusion product. Graph generation through GraphGUIDE can be thought of as a reverse-diffusion process which iteratively decides which edges to add or remove in order to recover a final graph sample x_0. This renders the generation process highly interpretable and controllable to humans.

Within the GraphGUIDE framework, nodes and their features can be thought of as static and eternal; the reverse-diffusion process determines which nodes to link up in order to form a final graph sample.

As such, in our experiments, we did not perform diffusion on node features. Of course, this requires the set of nodes to have a limited set of possible feature values. For many graph-generation tasks, this assumption holds, including graphs where only the structure is important (e.g. communities, stochastic block models, etc.), and molecular graphs (i.e. there are only a few types of atoms in typical molecular-generation tasks). Note, however, that in order to accommodate nodes with many more possible features, diffusion can also be performed on node features and edges jointly.

Although we demonstrated GraphGUIDE on undirected graphs without self-edges or multi-edges, our work can be easily extended to accommodate self-edges, multi-edges, directed edges, and graphs with edge attributes (as long as there is a relatively small set of discrete attributes). This can be done simply by having the binary vector x_t (which denotes edge presence) contain an entry for every possible edge or edge–attribute pair. Alternatively, GraphGUIDE can be combined with kernels other than the Bernoulli kernels, such as multinomial kernels like those proposed in Hoogeboom et al. (2021) and Austin et al. (2021). Graphs with many possible edge attributes or continuous edge attributes, however, do not fit very well into the GraphGUIDE framework, as a fundamental assumption made by the Bernoulli kernel (or multinomial kernels) is that all edges are binary (or have a limited number of discrete states, respectively).

Another limitation of our framework may be in the set of properties that can be conditionally generated via manual control. We demonstrated that GraphGUIDE enables conditional generation of graphs with arbitrary structural properties (e.g. molecules with 6-membered backbone rings), without the need to predefine them. Hence, in order to take advantage of controllable generation using our framework, the property desired must be definable structurally—that is, it needs to be definable by a set of known edges or graph motifs. As such, it is difficult to use this framework to generate graphs which satisfy a high-level property (e.g. a molecular graph which is a target for the β_2 adrenergic receptor), because it is not easy to identify what specific low-level structural properties (e.g. bonds or functional groups) confer this high-level property, which is a complex descriptor. Instead, high-level properties (e.g. drug targets) may be controlled for using the current standard methods of conditional generation, such as classifier-free conditional generation (i.e. training the diffusion model on a predefined set of labels) Ho et al. (2021). Notably, our framework remains orthogonal to classifier-free conditional generation, and can be easily combined with such methods so that conditional generation can be performed on a smaller set of pre-defined high-level properties as well as arbitrary low-level structural properties simultaneously.

5 Conclusion

In this work, we proposed GraphGUIDE, a novel framework for interpretable and controllable graph generation using diffusion models. To aid in generating interpretable and controllable diffusion intermediates, we presented three discrete diffusion kernels based on the Bernoulli distribution and applied them to graph edges. The resulting diffusion processes add noise to graphs by flipping, adding, or removing edges. In the reverse-diffusion process, the diffusion model iteratively decides which edges to keep or remove to recover a final graph sample. Notably, all diffusion intermediates are interpretable well-defined graphs. More importantly, by using the appropriate kernel, the generative process is highly controllable—specific edges, motifs, and other properties can be retained or prevented at each stage of the reverse-diffusion process, while still being in-distribution for the neural network. Because of this high degree of control over all parts of the generative process, practically any structural property can be conditioned on, and without relying on any predefined set of labels. Together, GraphGUIDE allows the enforcement of any arbitrary structural property on-the-fly with 100% success. Additionally, we demonstrated that these advantages in interpretability and controllability are gained without any penalty in generative performance.

We illustrated the benefits of GraphGUIDE for several kinds of undirected graphs, particularly highlighting the application to molecular graphs for drug discovery or chemical design. Our work, however, may be applied to generating other kinds of graphs in an interpretable and controllable manner, such as knowledge graphs and causality graphs. In both these situations, it would be highly beneficial to be able to control for certain substructures easily and interpretably. Furthermore, the framework defined by GraphGUIDE may be applied to data types outside of graph edges, as well. Further exploration in discrete diffusion and controllable generation can continue to have impacts in many other real-world domains.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 22:17981–17993, 7 2021. ISSN 10495258. doi: 10.48550/arxiv.2107.03006. URL https://arxiv.org/abs/2107.03006v2.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. 5 2018. doi: 10.48550/arxiv.1805.11973. URL https://arxiv.org/abs/1805.11973v2.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 11:8780–8794, 5 2021. ISSN 10495258. doi: 10.48550/arxiv.2105.05233. URL https://arxiv.org/abs/2105.05233v4.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020-December, 6 2020. ISSN 10495258. doi: 10.48550/arxiv.2006.11239. URL https://arxiv.org/abs/2006.11239v2.

Jonathan Ho, Google Research, and Tim Salimans. Classifier-free diffusion guidance, 11 2021.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. 4 2022. doi: 10.48550/arxiv.2204.03458. URL https://arxiv.org/abs/2204.03458v2.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 15:12454–12465, 2 2021. ISSN 10495258. doi: 10.48550/arxiv.2102.05379. URL https://arxiv.org/abs/2102.05379v3.

Tomaž Hočevar and Janez Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30:559–565, 2 2014. ISSN 1367-4803. doi: 10.1093/BIOINFORMATICS/BTT717. URL https://academic.oup.com/bioinformatics/article/30/4/559/205331.

Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. 2 2022. doi: 10.48550/arxiv.2202.02514. URL https://arxiv.org/abs/2202.02514v3.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. 9 2022. doi: 10.48550/arxiv.2209.15421. URL https://arxiv.org/abs/2209.15421v1.

Seul Lee, Jaehyeong Jo, Sung Ju Hwang, and South Korea. Exploring chemical space with score-based out-of-distribution generation. 6 2022. doi: 10.48550/arxiv.2206.07632. URL https://arxiv.org/abs/2206.07632v1.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation. 5 2022. doi: 10.48550/arxiv.2205.14217. URL https://arxiv.org/abs/2205.14217v1.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems*, 32, 10 2019. ISSN 10495258. doi: 10.48550/arxiv.1910.00760. URL https://arxiv.org/abs/1910.00760v3.

Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. Spectre : Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. 4 2022. doi: 10.48550/arxiv.2204.01613. URL https://arxiv.org/abs/2204.01613v1.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. 3 2020. doi: 10.48550/arxiv.2003.00638. URL https://arxiv.org/abs/2003.00638v1.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. 12 2021. doi: 10.48550/arxiv.2112.10752. URL https://arxiv.org/abs/2112.10752v2.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *32nd International Conference on Machine Learning, ICML 2015*, 3:2246–2255, 3 2015. doi: 10.48550/arxiv.1503.03585. URL https://arxiv.org/abs/1503.03585v8.

Yang Song, Jascha Sohl-Dickstein, Google Brain, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. 2021.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. 9 2022. doi: 10.48550/arxiv.2209.14734. URL https://arxiv.org/abs/2209.14734v1.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *35th International Conference on Machine Learning, ICML 2018*, 13:9072–9081, 2 2018. doi: 10.48550/arxiv.1802.08773. URL https://arxiv.org/abs/1802.08773v3.

A SUMMARY OF BERNOULLI KERNELS

Table S1: Forward, posterior, and prior distributions for three Bernoulli diffusion kernels. See Appendix B for derivations.

Kernel	Forward $q(x_t = 1 \mid x_0)$	Posterior $q(x_{t-1} = 1 \mid x_t, x_0)$	Prior $\pi(x_T = 1)$
Bit-flip	$(1-x_0)\bar{\beta}_t + x_0(1-\bar{\beta}_t)$, where $\bar{\beta}_t = \frac{1}{2} - 2^{t-1}\prod_{i=1}^{t}(\frac{1}{2}-\beta_i)$	$\frac{(x_t+\beta_t-2x_t\beta_t)(\frac{1}{2}+(2x_0-1)2^{t-2}\prod_{i=1}^{t-1}(\frac{1}{2}-\beta_i))}{\frac{1}{2}+(1-2(x_t-x_0)^2)2^{t-1}\prod_{i=1}^{t}(\frac{1}{2}-\beta_i)}$	$\frac{1}{2}^{\dagger}$
Bit-one	$x_0+(1-x_0)(1-\prod_{i=1}^{t}(1-\beta_i))$	$\frac{x_t(x_0+(1-x_0)(1-\prod_{i=1}^{t-1}(1-\beta_i)))}{x_0x_t+(1-x_0)x_t(1-\prod_{i=1}^{t}(1-\beta_i))+(1-x_0)(1-x_t)\prod_{i=1}^{t}(1-\beta_i)}$	$1^{\ddagger}$
Bit-zero	$x_0\prod_{i=1}^{t}(1-\beta_i)$	$\frac{(x_t+\beta_t-2x_t\beta_t)x_0\prod_{i=1}^{t-1}(1-\beta_i)}{(1-x_0)(1-x_t)+x_0(1-x_t)(1-\prod_{i=1}^{t}(1-\beta_i))+x_0x_t\prod_{i=1}^{t}(1-\beta_i)}$	$0^{\ddagger}$

†Assuming β_t approaches $\frac{1}{2}$ over a sufficiently long diffusion time
‡Assuming fractional β_t over a sufficiently long diffusion time

B DERIVATION OF BERNOULLI KERNELS

For all three kernels, we assume there is a fixed noise schedule β_t with $t \in \{1, ..., T\}$. A typical noise schedule monotonically increases from $\beta_1 = 0$ to $\beta_T = \frac{1}{2}$.

B.1 DERIVATION OF BIT-FLIP KERNEL

In order to "diffuse" or "noise" a binary vector x_0, we define a stochastic process in which bits are slowly flipped between 0 and 1. At time t, each bit independently flips with probability β_t, and is kept the same with probability $1 - \beta_t$. Note that if $\beta_t = 0$, then no noising happens.

Forward distribution

Here, we seek a closed-form expression for $q(x_t = 1|x_0)$ so that we may compute the forward distribution in an efficient way. We have that $x_t = b_t - x_0$, where b_t is a binary vector denoting whether or not a flip (relative to x_0) happens over diffusion time from 0 to t. From the piling-up lemma, we have that $b_t \sim Bern(\frac{1}{2} - 2^{t-1}\prod_{i=1}^{t}(\frac{1}{2} - \beta_i))$.

This gives us a forward distribution that is also distributed according to a Bernoulli distribution, where $q(x_t = 1|x_0) = (1 - x_0)\bar{\beta}_t + x_0(1 - \bar{\beta}_t)$, where $\bar{\beta}_t = \frac{1}{2} - 2^{t-1}\prod_{i=1}^{t}(\frac{1}{2} - \beta_i)$

Prior distribution

$\lim_{t\to T}\bar{\beta}_t = \frac{1}{2}$, as long as T is a sufficiently large time horizon and β_t approaches $\frac{1}{2}$. Thus, $q(x_T = 1) = \frac{1}{2}$, regardless of the value of x_0.

Posterior distribution

Here, we seek a closed-form expression for $q(x_{t-1} = 1|x_t, x_0)$ as a parameterized distribution. By Bayes' Rule, we have $q(x_{t-1} = 1|x_t, x_0) = \frac{q(x_t|x_{t-1}=1,x_0)q(x_{t-1}=1|x_0)}{q(x_t|x_0)}$.

Let's look at each piece separately:

$q(x_t|x_{t-1} = 1, x_0) = (1 - x_t)\beta_t + x_t(1 - \beta_t) = x_t + \beta_t - 2x_t\beta_t$

$q(x_{t-1} = 1|x_0) = (1 - x_0)(\frac{1}{2} - 2^{t-2} \prod_{i=1}^{t-1} (\frac{1}{2} - \beta_i)) + x_0(\frac{1}{2} + 2^{t-2} \prod_{i=1}^{t-1} (\frac{1}{2} - \beta_i)) = \frac{1}{2} + (2x_0 - 1)2^{t-2} \prod_{i=1}^{t-1} (\frac{1}{2} - \beta_i)$

$q(x_t|x_0) = (x_t - x_0)^2(\frac{1}{2} - 2^{t-1} \prod_{i=1}^{t} (\frac{1}{2} - \beta_i)) + (1 - (x_t - x_0)^2)(\frac{1}{2} + 2^{t-1} \prod_{i=1}^{t} (\frac{1}{2} - \beta_i)) = \frac{1}{2} + (1 - 2(x_t - x_0)^2)2^{t-1} \prod_{i=1}^{t} (\frac{1}{2} - \beta_i)$

Note $(x_t - x_0)^2$ is the xor between the two (binary) variables

That is, all together, we have that $x_{t-1} \sim Bern(\tilde{p}(x_0, x_t))$, where $\tilde{p}(x_0, x_t) = \frac{(x_t + \beta_t - 2x_t\beta_t)(\frac{1}{2} + (2x_0 - 1)2^{t-2} \prod_{i=1}^{t-1} (\frac{1}{2} - \beta_i))}{\frac{1}{2} + (1 - 2(x_t - x_0)^2)2^{t-1} \prod_{i=1}^{t} (\frac{1}{2} - \beta_i)}$

B.2 Derivation of bit-one kernel

Here, we define a stochastic process on binary x_0 where bits are slowly turned into 1s. At time t, each bit independently turns into a 1 with probability β_t. If the bit already is 1, then nothing happens. Note that if $\beta_t = 0$, then no noising happens.

Forward distribution

Again, we seek a closed-form expression for $q(x_t = 1|x_0)$ so that we may compute the forward distribution in an efficient way. We have that $q(x_t = 1|x_0) = x_0 + (1 - x_0)(1 - \prod_{i=1}^{t} (1 - \beta_i))$ (i.e. if $x_0 = 1$, then $x_t = 1$; otherwise, x_t will be 1 if at least one of the β_is is successful). Again, this is a Bernoulli distribution.

Prior distribution

$\lim_{t \to T} \prod_{i=1}^{t} (1 - \beta_i) = 0$, as long as T is a sufficiently large time horizon and β_t approaches some nonzero fraction (e.g. $\frac{1}{2}$). Thus, $q(x_T = 1) = 1$, regardless of the value of x_0.

Posterior distribution

We again start with Bayes' Rule: $q(x_{t-1} = 1|x_t, x_0) = \frac{q(x_t|x_{t-1}=1, x_0)q(x_{t-1}=1|x_0)}{q(x_t|x_0)}$.

Let's look at each piece separately:

$q(x_t|x_{t-1} = 1, x_0) = x_t$ (if $x_{t-1} = 1$, then $x_t = 1$ has a probability of 1, $x_t = 0$ can't happen and has a probability of 0)

$q(x_{t-1} = 1|x_0) = x_0 + (1 - x_0)(1 - \prod_{i=1}^{t-1} (1 - \beta_i))$

$q(x_t|x_0) = x_0x_t + (1 - x_0)x_t(1 - \prod_{i=1}^{t} (1 - \beta_i)) + (1 - x_0)(1 - x_t) \prod_{i=1}^{t} (1 - \beta_i)$

That is, all together, we have that $x_{t-1} \sim Bern(\tilde{p}(x_0, x_t))$, where $\tilde{p}(x_0, x_t) = \frac{x_t(x_0 + (1 - x_0)(1 - \prod_{i=1}^{t-1} (1 - \beta_i)))}{x_0x_t + (1 - x_0)x_t(1 - \prod_{i=1}^{t} (1 - \beta_i)) + (1 - x_0)(1 - x_t) \prod_{i=1}^{t} (1 - \beta_i)}$

B.3 Derivation of bit-zero kernel

Here, we define a stochastic process on binary x_0 where bits are slowly turned into 0s. At time t, each bit independently turns into a 0 with probability β_t. If the bit already is 0, then nothing happens. Note that if $\beta_t = 0$, then no noising happens.

Forward distribution

Again, we seek a closed-form expression for $q(x_t = 1|x_0)$ so that we may compute the forward distribution in an efficient way. We have that $q(x_t = 1|x_0) = x_0 \prod_{i=1}^{t}(1-\beta_i)$ (i.e. if $x_0 = 0$, then $x_t = 0$; otherwise, x_t will be 0 if none of the β_is was successful). Again, this is a Bernoulli distribution.

Prior distribution

$\lim_{t\to T} \prod_{i=1}^{t}(1-\beta_i) = 0$, as long as T is a sufficiently large time horizon and β_t approaches some nonzero fraction (e.g. $\frac{1}{2}$). Thus, $q(x_T = 1) = 0$, regardless of the value of x_0.

Posterior distribution

We again start with Bayes' Rule: $q(x_{t-1} = 1|x_t, x_0) = \frac{q(x_t|x_{t-1}=1,x_0)q(x_{t-1}=1|x_0)}{q(x_t|x_0)}$.

Let's look at each piece separately:

$q(x_t|x_{t-1} = 1, x_0) = x_t(1-\beta_t) + (1-x_t)\beta_t = x_t + \beta_t - 2x_t\beta_t$ (if $x_{t-1} = 1$, then $x_t = 1$ means β_t failed; on the other hand, $x_t = 0$ means β_t succeeded)

$q(x_{t-1} = 1|x_0) = x_0 \prod_{i=1}^{t-1}(1-\beta_i)$

$q(x_t|x_0) = (1-x_0)(1-x_t) + x_0(1-x_t)(1-\prod_{i=1}^{t}(1-\beta_i)) + x_0x_t\prod_{i=1}^{t}(1-\beta_i)$

That is, all together, we have that $x_{t-1} \sim Bern(\tilde{p}(x_0, x_t))$, where $\tilde{p}(x_0, x_t) = \frac{(x_t+\beta_t-2x_t\beta_t)x_0\prod_{i=1}^{t-1}(1-\beta_i)}{(1-x_0)(1-x_t)+x_0(1-x_t)(1-\prod_{i=1}^{t}(1-\beta_i))+x_0x_t\prod_{i=1}^{t}(1-\beta_i)}$

B.4 Training/Sampling algorithms

Training with the Bernoulli kernels

Input: training set $\{x^{(k)}\}$
Repeat:
    Sample x_0 from training data $\{x^{(k)}\}$
    Sample $t \sim Unif(0, T)$
    Sample $x_t \sim q_t(x|x_0)$
    Gradient descent on $p_{(\theta_s,\theta_i)}(x_t, t)$ to predict x_0
Until convergence

Sampling with the Bernoulli kernels

Input: trained p_θ
Sample $x_t \leftarrow x_T$ from $\pi(x)$
for $t = T$ to 0: $\hat{x_0} \leftarrow p_\theta(x_t, t)$
    Sample $x_{t-1} \sim q(x_{t-1}|x_t, \hat{x_0})$
    $x_t \leftarrow x_{t-1}$
Return $\hat{x}$

C SUPPLEMENTARY METHODS

The code to generate the results and figures in this work is available at `https://github.com/Genentech/GraphGUIDE`.

We trained all of our models and performed all analyses on a single Nvidia Quadro P6000.

C.1 DATASETS

Community-small

We generate a dataset of two-community graphs following the definition in You et al. (2018). That is, we first pick $|V| \in [12, 20]$, which is split into two equal-sized communities (for an odd number, one community will have one more node than the other). Each community is generated by the Erdös–Renyi model with $p = 0.3$. We then add $0.05|V|$ edges uniformly between the two communities.

All nodes have a single feature with value 1.

We generate 6400 random graphs per epoch, where graphs across different epochs are generated independently.

For our generative-performance analyses, following You et al. (2018); Liao et al. (2019); Martinkus et al. (2022); Vignac et al. (2022), we pre-generated and cached 200 random graphs, and trained only on these 200 repeatedly.

Stochastic block models

We generate a dataset of stochastic-block-model graphs following the definition in Martinkus et al. (2022). That is, we first pick a number of blocks between $[2, 5]$. Each block size is independently and uniformly sampled from $[20, 40]$ nodes. The intra-block edge probability is 0.3, and inter-block edge probability is 0.05.

All nodes have a single feature with value 1.

For our generative-performance analyses, following You et al. (2018); Liao et al. (2019); Martinkus et al. (2022); Vignac et al. (2022), we pre-generated and cached 200 random graphs, and trained only on these 200 repeatedly.

Cliques

We generate a dataset of clique graphs. Each graph has 10 nodes, and contains two cliques randomly drawn from sizes $\{3, 4, 5, 6\}$, without replacement. Excess nodes are left as singletons.

All nodes have a single feature with value 1.

We generate 6400 random graphs per epoch, where graphs across different epochs are generated independently.

Molecule-like graphs

We generate a dataset of molecule-like graphs. The number of backbone nodes is selected randomly from $[4, 6]$, and the number of secondary nodes is selected randomly from $[0, 4]$. The backbone nodes are first linked into a ring or a branched structure (with 0.5 probability of each). If the backbone is to be linked into a ring, it is linked into a single simple cycle. If the backbone is to be linked into a branched structure, a random tree is constructed such that the degree of each backbone node is at most 4. In order to do this, backbone nodes are iteratively and stochastically added to a structure which is slowly built up by randomly adding a leaf node to some pre-existing node with degree less than 4.

Once the backbone is constructed, secondary nodes are added uniformly at random to the backbone. Secondary nodes are always added as "leaves" (i.e. they always have degree 1, attached to a backbone node), such that the backbone nodes have degree at most 4. The random selection is done such that each free spot on a backbone node has equal probability of having a secondary node added.

Singleton backbone and secondary nodes are added (in equal amounts, or as equal as possible) to create a graph of 10 nodes.

All backbone nodes have a single feature with value 0, and all secondary nodes have a single feature with value 1.

We generate 6400 random graphs per epoch, where graphs across different epochs are generated independently.

C.2 Model architecture

For most of our experiments, we used a graph-attention network. We compute a time embedding as $[\sin(\frac{\pi}{2}\frac{t}{T}), \cos(\frac{\pi}{2}\frac{t}{T}), \frac{t}{T}]$. The time embedding is passed through a dense layer with 256 units. This output is concatenated with the node features. This is then passed through 2 dense layers of 256 units each, followed by a ReLU after each one.

We also compute a spectrum transformation matrix for each graph, where we take the eigendecomposition of the Laplacian $L = I - D^{\frac{1}{2}}AD^{\frac{1}{2}}$ for adjacency matrix A and degree matrix D. We limit the eigenvectors to the 5 smallest eigenvalues (or the number of nodes in the graph if there are fewer than 5).

The node embeddings are passed through 5 GAT layers. In each layer, the input node embeddings are passed through a spectral convolution using the spectrum transformation matrix. The spectral convolution has the same number of units as the input node embeddings. This is concatenated with the original input node embeddings for the layer, and passed through a GAT with 8 attention heads of 32 units each. The GAT attends every node to every other, thereby ensuring that information is passed between all nodes even as edge connectivity changes over the course of diffusion. Edge presence/absence is encoded as a binary set of edge attributes which are passed to the GAT. The output of each GAT layer is passed through dropout of probability 0.1, a ReLU, and layer normalized with a summed residual connection with the GAT output. This is then passed twice through a series of dense units which maintain dimensionality, dropout, and layer normalization with a summed residual connection.

The final output of the GAT layers is passed through two dense layers of 256 units each (followed by a ReLU after each one), then a final dense layer which maps each node's embedding into a single scalar. The output is a probability of each edge, which is the product of the two endpoint nodes' scalar embeddings passed through a sigmoid.

For our benchmark of stochastic block models, we use the architecture in Vignac et al. (2022).

Our loss function is the binary cross entropy between the model output and x_0 (which is binary).

C.3 Training schedules

For all discrete Bernoulli kernels, we used a noise schedule of $\beta_t = \min(\sigma(\frac{t}{100} - 10), \frac{1}{2} - 10^{-6})$, where the 10^{-6} is for numerical stability. We set $T = 1000$ time steps.

For our experiments, we trained for 100 epochs with a learning rate of 0.001. We used a batch size of 32.

For our benchmarking of generative performance, we trained for 200 epochs with a learning rate of 0.001. We used a batch size of 32, except we used a batch size of 16 for the stochastic block models.

C.4 Generative performance

Following the convention set by You et al. (2018) and continued by other graph-generation works including Liao et al. (2019); Martinkus et al. (2022); Vignac et al. (2022); etc., we computed various multivariate measurements on the generated graphs and the training graphs (i.e. node degrees, clustering coefficients, and node orbit counts), and compared these multivariate measurements using maximum mean discrepancy (MMD).

We computed node orbit counts using Orca Hočevar & Demšar (2014).

Like previous works, we computed MMD using the Gaussian-total-variation kernel. For node degrees, we directly compared the count histograms; we used $\sigma = 1$ in the kernel. For clustering coefficients, we compared the histograms drawn with 100 bins; we used $\sigma = 0.1$ in the kernel. For node orbit

counts, we computed the average orbit count for each orbit type, and directly compared the histograms with $\sigma = 30$ in the kernel. These settings match the settings published in previous works, including Martinkus et al. (2022).

Following Martinkus et al. (2022) and Vignac et al. (2022), we also computed the MMD between the training set and an equal-sized validation set drawn from the same distribution. We report the ratio of the MMDs as the MMD between generated and training graphs, divided by the MMD between training and validation graphs.

In order to compare our MMD ratios with other works, we took the published MMD values for GraphRNN, GRAN, MolGAN, and SPECTRE from Martinkus et al. (2022). We took the MMD ratios for DiGress directly from Vignac et al. (2022).

Importantly, all cited previous works report MMD *squared*, but the values are erroneously described as simply MMD in the text. Here, we adjust for this error by simply taking the square root of the reported values before comparing them to ours.

We also note that significantly lower (by orders of magnitude) MMD values were achieved by training on only a small (e.g. 200) set of cached graphs, and computing the MMD of generated graphs to this training set. In contrast, the MMD was much larger if the training set consisted of graphs randomly generated on the fly (i.e. there is no caching of graphs), and if the generated graphs were compared to an independently sampled test set. Of course, in this work we trained and tested on the same small set of cached graphs in order to be comparable to previous methods. Unfortunately, however, this is less realistic and inherently suffers from train–test leakage. Instead, we propose that moving forward, generative models may be trained on larger samples and that MMD values are computed against an independently sampled test set.

C.5 Controllable generation

To generate cliques of size 6, we started with our model trained on the full cliques dataset, using the bit-one kernel. After sampling from the prior (i.e. the complete graph K_{10}), we selected 6 nodes arbitrarily. At each point in the reverse-diffusion process, we ensured that edges do not get removed between these 6 nodes. If a reverse-diffusion step removed any of those edges, they were added back before the next reverse-diffusion step (including the very last step which generates the final graph). In figures which show generated cliques, only one clique per graph is shown for clarity.

To generate disjoint communities, we started with our model trained on the full community-small dataset, using the bit-zero kernel. After sampling from the prior (i.e. the empty graph with size between 12 and 20), we arbitrarily partitioned the nodes into two communities of equal size (or as equal as possible in the case of an odd number of total nodes). At each point in the reverse-diffusion process, we ensured that edges do not get added between these the two communities. If a reverse-diffusion step added any of those edges, they were removed before the next reverse-diffusion step (including the very last step which generates the final graph). We consider a generated graph to have two distinct communities if the largest connected component has fewer than 60% of the nodes.

To generate molecule-like graphs with a 6-membered backbone ring, we started with our model trained on the full molecule-like dataset, using the bit-flip kernel. After sampling from the prior (i.e. Erdös–Renyi graph of size 10 and $p = 0.5$, with an average of 6 backbone nodes), we arbitrarily selected 6 backbone nodes and assigned each a ring position. Any graphs without 6 backbone nodes were tossed out from the analysis. At each point in the reverse-diffusion process, we ensured that these 6 nodes were connected in a ring (according to the ring positioning), and had no other edges between them. If a reverse-diffusion step added or removed any improper edges, they were fixed before the next reverse-diffusion step (including the very last step which generates the final graph). In figures which show generated molecule-like graphs, excess singleton nodes were removed for clarity.