TEXT-CONDITIONED GRAPH GENERATION USING DISCRETE GRAPH VARIATIONAL AUTOENCODERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Inspired by recent progress in text-conditioned image generation, we propose a model for the as-yet unexplored problem of text-conditioned graph generation. In this paper we introduce the Vector Quantized Text To Graph generator (VQ-T2G), a discrete graph variational autoencoder and autoregressive transformer for generating general graphs conditioned on text. We curate two multimodal datasets of graphs paired with text, a real-world dataset of 8000 subgraphs from the Wikipedia link network and a dataset of over 3000 synthetic graphs. Experimental results on these datasets demonstrate that VQ-T2G synthesises novel graphs with structure aligned with the text conditioning. Additional experiments in the unconditioned graph generation setting show VQ-T2G is competitive with existing unconditioned graph generation methods across a range of standard graph metrics.

1 INTRODUCTION

Graphs are a natural way of representing relational and structural data such as molecules, social networks, or knowledge graphs. The range of data types they may represent makes the problem of learning and generating graphs important, with broad applications. For example, molecular graph generation may be used in drug design Simonovsky & Komodakis (2018); De Cao & Kipf (2018), and synthetic graph models are central to network science Newman (2018); Robins et al. (2007)

The study of graph generation dates back to the 1950s with the Erdős-Rényi random graph model Erdős et al. (1960). This and similar early work in network science Albert & Barabási (2002); Holland et al. (1983); Watts & Strogatz (1998) focused on models that generate a single class of graphs with known statistical properties, with a small number of parameters to control structure. Because they are simple and not learned from data, they have limited capacity to mimic the complex dependencies and structures frequently observed in real graphs Broido & Clauset (2019). Modern approaches to graph generation use neural networks to learn a model over a dataset of graphs, and then sample novel graphs from the model that mimic those in the dataset. Such models demonstrate far stronger ability to synthesise graphs from areas including molecules, citation networks, 3D point clouds, and varieties of synthetic graphs Simonovsky & Komodakis (2018); Liao et al. (2019); You et al. (2018); Shirzad et al. (2022); Martinkus et al. (2022).

Outside of the graph generation domain, multimodal modelling has received significant attention in recent literature, most notably in the vision-language domain for text-to-image generation Ramesh et al. (2021); Nichol et al. (2021); Ramesh et al. (2022). This interest has been primarily spurred by DALL-E Ramesh et al. (2021) whose training paradigm stems from the single-modal vector-quantized variational autoencoder (VQ-VAE) Van Den Oord et al. (2017). To date, no prior work has studied multimodal modelling for graphs and text in the graph generation task. Moreover, while VQ-VAE has been studied for data such as videos Walker et al. (2021); Yan et al. (2021) and music Dhariwal et al. (2020) it has not yet been used for graph generation. This paper addresses both these gaps in the literature.

Specifically, we investigate the novel problem of generating graphs conditioned on text, *text-to-graph generation*. Given a dataset of paired graphs and text $\mathbf{G} = \{(G_1, T_1), (G_2, T_2), \dots (G_m, T_m)\}$, we learn a model over this dataset that can generate novel graphs with similar structure to those in the training set. In addition, when conditioned on text captions, the sampled graphs should have structure close to that described in the caption, or in other words, similar structure to graphs with semantically similar captions. We name our proposed model

for this problem the Vector-Quantized Text-to-Graph Generator (VQ-T2G). The model is trained in a similar procedure to DALL-E Ramesh et al. (2021). The first stage trains a VAE to encode graphs into a low-dimensional discrete latent representation. The second stage then trains a multimodal autoregressive transformer that learns a prior jointly over this discrete latent space induced by the graphs and natural language text associated with each graph. Altogether this process allows the synthesis of new graphs conditioned on text. While the process shares similarities to DALL-E, there are key challenges that make the use of text and graphs nontrivial:

1. Graphs have irregular structure and a dataset may contain vastly differing numbers of nodes or edges. This is challenging in particular for the discrete latent representation and graph reconstruction steps.
2. Datasets of graphs paired with text that are able to be used for this task do not yet exist.
3. Compared with image-text pairings, in many domains a text string describing a graph will be less informative than an image caption which, for example, names the foreground objects and their interaction.
4. Robust evaluation methods for text-to-graph generation do not yet exist. We must develop accurate and sufficient methods of evaluating the model. It is also less clear how to measure the similarity between a generated graph and text description.

We address each of these challenges in this paper. Our key contributions are as follows. First, we introduce the novel problem of text-to-graph generation. Second, we extend the framework of VQ-VAE to graph-structured data and apply it successfully to graph generation. From this, we demonstrate that text conditioning can guide the structure of generated graphs through an autoregressive prior learned over the discrete latent space of such a model. Finally, we curate two novel datasets of graphs paired with text for this text-to-graph generation problem.

2 Related work

Deep learning on graphs Graph neural networks (GNN) have proven powerful in a variety of applications such as recommender systems and social network community detection. While many GNN architectures exist in the literature, the most relevant to our work is VQ-GNN Ding et al. (2021). This is the only model to date that uses vector quantization in a GNN. The key part of VQ-GNN is its introduction of discrete node representations as a component of an approximated message passing scheme. The design targets improving memory efficiency and therefore scalability in node representation learning for large graphs. VQ-GNN is applicable in large graphs but cannot readily be used in graph generation. The approximated message passing will negatively impact performance in smaller graphs where memory issues are of less concern.

Unconditioned graph generation In the unconditioned setting, graph generators aim to learn a distribution $p(G)$ over graphs from a dataset of observed graphs, then sample from this distribution to synthesise novel graphs similar to those trained on. Autoregressive models such as GraphRNN You et al. (2018) and GRAN Liao et al. (2019) approach graph generation as an autoregressive decision process, adding nodes and/or edges to build graphs sequentially. Models such as GraphVAE Simonovsky & Komodakis (2018) instead generate the entire graph adjacency matrix in a single step.

Conditioned graph generation Including some type of conditioning in the graph generation process may be useful in applications where more model control is desirable. VQ-T2G falls into this category of graph generator, with the type of conditioning being natural language text. Many other varieties of conditioning have been explored in prior work.

GraphTune Nakazawa et al. (2022) conditions samples on a numeric value of a given global structural property, such as the modularity or clustering coefficient. The generator is trained to synthesise graphs having as close to that property value as possible. SPECTRE Martinkus et al. (2022) generates graphs in two stages. Part of the generated graph spectrum is first sampled, with the model learning to sample realistic spectrums during training. The graph structure is then generated conditioned on this sampled spectrum. The MolT5 Edwards et al. (2022) model allows for controllable

generation of molecules through conditioning on a natural language description of a molecule's desired characteristics, such as its physical properties. While apparently similar to VQ-T2G in the sense that the conditioning is text-based, MolT5 only operates on a single data modality through the SMILES representation. Since it does not consider graph structure, it remains fundamentally different to our model. Condgen Yang et al. (2019a) addresses the most similar setting to VQ-T2G. This model uses a conditional generative adversarial network (GAN) architecture to condition graph generation on a real-valued vector of graph level features. This is done through concatenating a continuous graph embedding with one or more conditioning vectors associated with the graph. For example, in a dataset of citation graphs, generated graphs could be conditioned on the publication venue and other citation metadata. The metadata is represented as a real-valued vector by conversion to an appropriate numeric value or one-hot encoding. In contrast with our setting of natural language conditioning, these features used in Condgen are relatively simple. While it was not designed for more complex conditioning such as text, we note that it is the only existing model capable of graph-level conditioning with arbitrary vectors. As such we explore representing text for conditioning in Condgen to use it as a key baseline in VQ-T2G experiments.

Text-to-graph problems Problems known as *text to graph generation* exist in two other domains. These are knowledge graph generation Guo et al. (2020); Wang et al. (2021), and scene graph generation, when the scene graph is generated from language Yang et al. (2019b); Chang et al. (2021). We note that both are fundamentally different to our problem. In contrast with our goal of generating novel graphs, these tasks are better described as graph *inference* or *graph* extraction Melnyk et al. (2021). In other words, models for these tasks are trained to parse the input text, infer relationships in it, and construct a graph reflecting those relationships. Our task instead involves generating *novel* graphs from text.

Vector-quantized representations in generative models VQ-VAE learns a discrete latent representation of data with an autoencoder then trains a second model to sample from the prior. With the prior being learned, not static, they often outperform vanilla VAEs which use continuous latent spaces. VQ-VAEs have proven powerful in generating data such as images Van Den Oord et al. (2017), video Walker et al. (2021) and music Dhariwal et al. (2020). DALL-E modifies the VQ-VAE framework to incorporate text conditioning in the second stage for text-to-image generation. As mentioned in section 1, graphs are irregularly structured, e.g. having differing sizes within datasets. As such, the VQ-VAE architecture is not immediately applicable in graph generation and we modify it to use graph structured data.

3 MODEL

We consider a dataset of graph-text pairs $\mathbf{G} = \{(G_1, T_1), (G_2, T_2), \dots (G_m, T_m)\}$. Each simple, undirected graph is defined by its set of nodes V and edges E as $G = (V, E)$. Graphs are paired (captioned) with text T. Under some chosen node ordering π, G is represented as an adjacency matrix $A^\pi \in \mathbb{R}^{N \times N}$ with N the maximum graph size. When $|V| < N$ the graph is padded with isolated (fake) nodes. Nodes have feature vectors of dimension d_{in} and the node feature matrix is denoted $X \in \mathbb{R}^{N \times d_{in}}$.

VQ-T2G is trained on such datasets in two stages. The first stage trains a graph autoencoder with a discrete bottleneck to encode graphs into a discrete latent space and reconstruct them from this representation. We call this model the graph vector-quantized variational autoencoder (GVQVAE). The second stage learns to sample graphs from the GVQVAE by training an autoregressive transformer over the discrete latent space, using text as conditioning. Figure 1 shows the architecture of the GVQVAE model. It takes as input a graph adjacency matrix A^π and feature matrix X and learns to reconstruct the graph structure. The decoder outputs a probabilistic adjacency matrix $\tilde{A}$ representing the predicted graph.

3.1 GRAPH VECTOR-QUANTIZED VARIATIONAL AUTOENCODER

We use a VAE with a discrete bottleneck to encode the graph G represented by its adjacency matrix A^π into a vector z of discrete latent variables, then reconstruct the original graph from this representation. We call this model the Graph Vector-Quantized Variational AutoEncoder (GVQVAE).

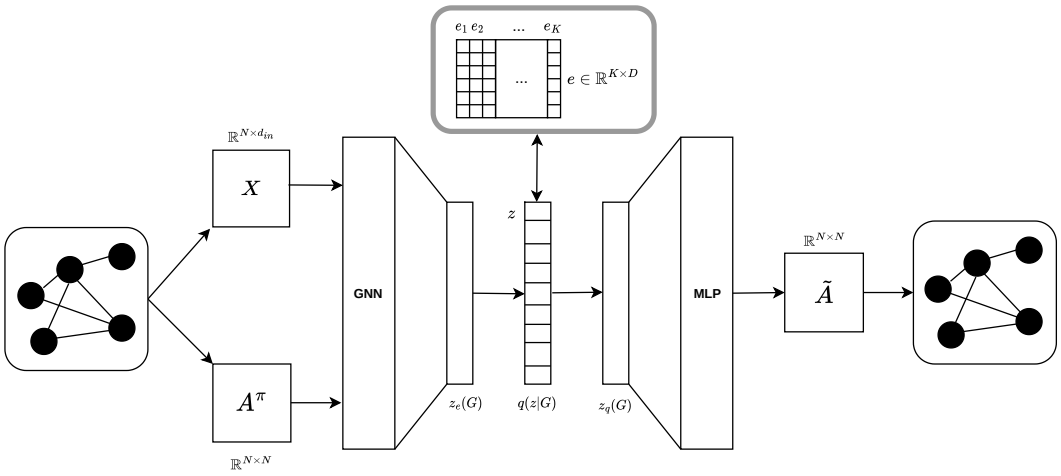

Figure 1: Architecture of the GVQVAE model. The input graph (left) is represented under node ordering π as adjacency matrix A^π. This adjacency matrix and the graph's node feature matrix X are the input to the encoder. The graph is encoded then the output mapped to the nearest codebook vectors before being input to the decoder, to construct probabilistic adjacency matrix $\tilde{A}$. The graph is then reconstructed from this probabilistic matrix.

The encoder is defined by a variational posterior $q_\phi(z|G)$ and the decoder by a generative distribution $p_\theta(G|z)$. Each element of z maps to a continuous vector in the model's codebook of size $K \in \mathbb{N}$. The codebook is an embedding space $e \in \mathbb{R}^{K \times D}$, with K the size of the codebook, D the dimension of each codebook vector, and e_i denoting the i'th embedding vector. As in Van Den Oord et al. (2017), encoder outputs $z_e(G)$ are mapped to their nearest embedding vector to produce a vector of codebook indices. During the forward pass the decoder takes as input the encoding of the graph mapped to the nearest codebook vectors, $z_q(G)$. However the operation that maps encoder outputs to codebook vectors has no defined gradient, so in the backwards pass the gradient skips the codebook and is passed directly to the encoder. The GVQVAE loss function $\mathcal{L}_G$ is similar to the original VQ-VAE loss with the first term replaced with a graph reconstruction loss:

$$\mathcal{L}_G = \log p(G|z_q(G)) + \|\text{sg}[z_e(G)] - e\|_2^2 + \beta\|z_e(G) - \text{sg}[e]\|_2^2 \tag{1}$$

As usual, sg denotes the stopgradient operator. The first term is the graph reconstruction loss, measured with binary cross-entropy loss. The second term is the VQ loss. This encourages embedding vectors to move closer to encoder outputs, as measured by l_2 distance. The final term is a commitment loss which encourages the encoder outputs to not stray too far from codebook vectors. Since backpropagation bypasses the discrete bottleneck, only the second term optimises the codebook. The first term is optimised by both the encoder and decoder, and the final term is optimised by the encoder.

Encoder The encoder begins with message passing GraphSAGE Hamilton et al. (2017) layers with ReLU activation and BatchNorm to obtain a low-dimensional embedding for each node. These node representations are then flattened to a single vector and passed through linear layers to give a graph-level latent representation. This representation is then reshaped to N_Z vectors of dimension D which is then mapped to the nearest codebook codes. The hyperparameter N_Z denotes the number of codes in the latent representation; the use of linear layers means N_Z need not be equal to the number of nodes in the original graph.

Learning on differently-sized graphs up to a pre-specified maximum number of nodes is done through padding all smaller graphs with isolated nodes up to the maximum number. In practice we choose this as the largest graph size in the dataset. For initial node representations we use a combination of positional, structural and random features. Specifically, the features for the i-th node of a graph G are the concatenation of three feature vectors. The first is the i-th row of the graph adjacency matrix A^π, with a value of 2 inserted at position i to indicate the node itself. If the graph

has fewer nodes than the maximum graph size N we pad this vector with -1's. Second is a feature of the node's degree, either the raw value or a one-hot degree vector. The final features are randomly sampled from a Gaussian distribution with mean $\mu = 0$ and are computed during each forward pass of the encoder. These random features act as regularisation during training by forcing the model to learn a more difficult problem Abboud et al. (2020) and produce a more robust model at the end of training. The type of degree vector, the number of random features and the random feature standard deviation σ^2 are hyperparameters.

Decoder The decoder has a multi-layer perceptron (MLP) architecture and uses Tanh activation and dropout after each layer. The output of the final layer is reshaped to a tensor $\mathbf{R} \in \mathbb{R}^{N \times d_{out}}$, corresponding to individual node embeddings of dimension d_{out}. An inner product followed by sigmoid is applied to this tensor to obtain a probabilistic adjacency matrix $\hat{\mathbf{A}} = \sigma\left(\mathbf{R}\mathbf{R}^\top\right) \in \mathbb{R}^{N \times N}$. The process for constructing graph edges from $\hat{\mathbf{A}}$ is detailed in the appendix.

VQ bottleneck The codebook collapse problem in vector-quantized models is common Dhariwal et al. (2020); Łańcucki et al. (2020). It refers to the tendency of models to utilise only one or few codes, ignoring the remainder of the codebook, and therefore have low representational capacity. To alleviate this issue we follow Łańcucki et al. (2020) and batchnorm the output of the encoder before passing to the VQ bottleneck, and set the learning rate of the codebook to be ten times that of the encoder and decoder. Empirically we find codebook collapse to be rare only when including both of these. We also perform codebook refits at pre-selected steps during training. These refits are done by fitting k-means to the encoded training graphs and setting the codebook vectors to equal the learned cluster centers. We find this speeds up learning and utilisation of the codebook when performed early on in training. However if done later in training, may cause immediate worse performance which the model may never recover from.

3.2 Transformer decoder

The autoregressive transformer learns the prior $p(z)$ over the GVQVAE discrete latents, incorporating text as conditioning to control generated graph properties. Sequences of graph tokens (representing codebook vectors) are used as input to the GVQVAE decoder to generate samples. The text and graph tokens are jointly modelled by a decoder-only transformer similar to Radford et al. (2019). Text and graph tokens are concatenated to form a single sequence, and the transformer utilises global attention over the entire sequence. The text vocabulary is learned through byte-pair encoding Sennrich et al. (2015) of all texts T. To handle the multimodality of the text and graph data we keep text and graph tokens separate in the full vocabulary: after the text vocabulary is learned, K additional graph tokens are added to represent the 'vocabulary' of codebook indices. This allows us to then train a multi-modal transformer model over sequences of concatenated text and graph tokens.

Following Ramesh et al. (2021) we lowercase text prior to encoding. The number of text token positions in the model is set to the maximum encoded length in any caption, and encoded texts shorter than this are padded to this max length using a single text padding token. A special start-of-graph token is also appended after the final text token position to indicate the beginning of the graph tokens. To ensure valid graph sequences during testing the model is forced to only produce tokens from the graph vocabulary. The model uses positional encoding for both text and graph tokens. While pretrained checkpoints of large language models have been used for text-conditioned generative models of images Saharia et al. (2022) and molecules Edwards et al. (2022), we train the transformer from scratch in order to keep model size small.

4 Datasets

We introduce two graph-text datasets to evaluate our model.

While datasets of text and graph pairs for knowlege graphs Jin et al. (2020); Wang et al. (2021) and scene graphs exist, these are unsuitable for our task of text conditioned graph generation. In these datasets the text pairings are not a caption for (or description of) the graph. A single caption could describe a large number of distinct images, and we similarly require datasets in which text

descriptions may describe many possible graphs. Our two datasets are as follows. First is a real-world dataset of egocentric networks from the English Wikipedia page link network, and second, a dataset of synthetic graphs. Graphs in the synthetic dataset are paired with a natural language description of the graph type and overall structural features. Graphs in the Wikipedia dataset are paired with the concatenation of the article title and first sentence from the ego (center) node. Instead of directly specifying the graph structure these captions act as a way to describe *semantics related to the topic of the graph*. In contrast with the synthetic dataset it directly describes the type and/or topology, this text can be seen as a type of *query for the graph*.

4.1 WIKIPEDIA EGOS

For our novel real-world dataset we draw two-hop ego networks from category-wise subgraphs of the English Wikipedia inter-page link network. Nodes correspond to articles and edges mean that one page has an in-text link to the other. Following common practice You et al. (2018); Shirzad et al. (2022) we do not consider edge direction. The egocentric nature of these graphs gives rise to a natural text pairing with the text of the central article. We retain the article title and first sentence as conditioning text. The dataset contains 8000 graphs with $60 \leq |V| \leq 160$. We do not sample from the entire Wikipedia link network, instead we construct eight topical subgraphs of the full link network and sample 1000 graphs from each. Further details of these topical subgraphs and sampling are in Appendix B. We motivate the use of egos sampled only from these topical subgraphs of the Wikipedia link network as we are interested in learning on and generating only the semantically-related neighbourhood of articles to the central page. This setting is to show that there exists correlation between the semantic context of the text description and the local graph structure in the semantically-related neighborhood of a node, and that we may learn this with a model like VQ-T2G.

4.2 SYNTHETIC

Our synthetic dataset consists of sixteen distinct varieties of synthetic graphs. Similar diverse synthetic datasets have been used in recent graph learning tasks Corso et al. (2020); Veličković et al. (2020), ours extends these ideas with additional graph varieties and supplements the graphs with text captions. Graphs have size $20 \leq |V| \leq 160$, and the dataset contains 3186 graphs in total. Captions are constructed methodically to describe the graph in a few short sentences. They include a description of the graph variety, along with some other attributes such as the number of nodes or parameters used to generate the graph. Each synthetic graph may be described in multiple ways and we randomly construct one description for each. Synthetic datasets used in prior graph generation work You et al. (2018); Liao et al. (2019); Martinkus et al. (2022) are generally small and consist of a single type of graph, in contrast to ours. We desire a larger and more varied dataset for two reasons. Firstly, we cannot easily show the power of text-conditioning when training graphs have little variety. Secondly, it is difficult to create sufficient variety in the text descriptions when graphs come from one or few graph families. Real-world text is complex and learning a relatively small range of text is uninformative.

4.3 GRAPH-ONLY DATASETS

So we may more closely compare to previous graph generation work, we also evaluate with two datasets of graphs without text. The first is a two-community graph dataset as used in You et al. (2018) and Martinkus et al. (2022) (60-160 nodes). The second is the wiki ego dataset from above, but without using the texts.

5 EXPERIMENTS

We verify the ability of VQ-T2G to generate graphs from text captions using the datasets outlined in section 4. First we compare distributions of graph statistics generated from the texts of graphs in the test set against the ground truth graphs (i.e. the graphs actually paired with those texts). In addition we perform visual inspection and evaluation of generated graphs. As this is a novel problem setting and few general graph generative models allow any conditioning, the only other graph generation model we may compare to while using text conditioning is Condgen Yang et al.

	Synthetic				Wiki ego, in-topic				Wiki ego, out-of-topic			
	Degree	Clust.	Orbit	Spect.	Degree	Clust.	Orbit	Spect.	Degree	Clust.	Orbit	Spect.
Condgen	0.33	$9.0e^{-2}$	$\mathbf{8.9e^{-4}}$	0.32	0.32	**0.12**	$\mathbf{6.4e^{-4}}$	0.30	0.71	1.01	1.00	0.65
VQ-T2G	$\mathbf{8.5e^{-2}}$	$\mathbf{4.5e^{-2}}$	$4.5e^{-2}$	$\mathbf{2.3e^{-2}}$	**0.14**	0.19	$9.5e^{-2}$	**0.11**	**0.13**	**0.19**	$\mathbf{8.4}e^{-2}$	$\mathbf{7.9}e^{-2}$

Table 1: Comparison with Condgen on the synthetic dataset and wiki ego dataset in the text-conditioned experiments. Smaller MMD scores are better. Degree: degree distribution, Clust.: clustering coefficient, Orbit: 4-node orbit counts, Spect.: graph Laplacian spectrum.

(2019a). As described in Section 2, this model uses a GAN to generate graphs conditioned on a feature vector. In Condgen we set the feature vector to be the byte-pair encoded text description for the graph. This allows use of conditioning with text and therefore clear comparison to our model.

For both datasets we use a split of 90% of data for training and 10% for testing. This split is shared in both the GVQVAE and transformer training to prevent data leakage between stages. Full training setup and hyperparameters for all experiments are detailed in Appendix A

5.1 Evaluation metrics

Our metrics follow those commonly used in the graph generation literature You et al. (2018); Liao et al. (2019); Shirzad et al. (2022); Martinkus et al. (2022). That is, we use the maximum mean discrepancy (MMD) over four graph statistics: degree distribution, clustering coefficient distribution, the count of all orbits with 4 nodes, and the eigenvalues of the normalised graph Laplacian. The first three statistics represent local graph properties and the fourth represents global graph structural properties. A variety of kernels have been used in the MMD in the graph generation literature, such as the earth mover's distance (EMD), total variation (TV) distance, or radial basis function (RBF) O'Bray et al. (2021). We use the TV distance as it is fast to evaluate on larger datasets such as ours, while being consistent with the EMD. These statistics will typically be computed between the test set and an equal number of graphs generated by the model. We follow this method for the synthetic dataset. Note that for our text-conditioned setting these statistics primarily indicate whether the model has learned the dataset distribution overall, but does not necessarily inform the performance of text conditioning. To address this, for the Wiki ego dataset we instead compute these statistics over each of the eight topics. Specifically, we generate graphs from all captions in the test set, then split these by the topic the graph's caption originally came from. For each of these sets, we compute the MMD score between the set of generated graphs and the set of test set graphs from that topic, denoting this the *in-topic* scores. We then compute the MMD score between that set of graphs and the test graphs from all other topics (the *out-of-topic* scores). Finally, we take the average over the eight scores for both in-topic and out-of-topic scores.

Naturally, a model that effectively learns to use the text conditioning will have low MMD scores for the in-topic graphs. Recall the text captions for this dataset *correlate* with the graph structure, they do not specify the structure directly. To ensure the model did not simply overfit (using the text) to some part of the in-topic graph distribution, a well-trained text-conditioned model should also score well against out-of-topic graphs. Intuitively, the graphs in this dataset all have a particular structure, that is, two-hop ego networks from the Wikipedia link network. The differences between classes (topics) here are less pronounced than in an image dataset. As such we would expect, when averaging over a set of generated graphs from one topic, that the statistics would share similarities with the statistics of the dataset as a whole too.

5.2 Results

5.2.1 Text-conditioned graph generation

We outline our results in experiments performed on our two graph-text datasets, in the text-conditioned setting Results for these text-conditioned experiments are reported in Table 1. On the majority of the MMD metrics measured, our model significantly outperforms Condgen. Notably, Condgen scores far worse on the out-of-topic Wiki ego MMD scores compared to the in-topic scores. This indicates Condgen has not learned to utilise the text conditioning well and overfits to

	Two-community				Wiki ego, unconditioned			
	Degree	Clust.	Orbit	Spect.	Degree	Clust.	Orbit	Spect.
GRAN	0.23	$\mathbf{3.9e^{-2}}$	$8.5e^{-2}$	$\mathbf{2.4e^{-2}}$	$\mathbf{6.7e^{-3}}$	$2.7e^{-2}$	$1.3e^{-2}$	$3.3e^{-2}$
SPECTRE	0.29	$5.6e^{-2}$	$\mathbf{2.5e^{-2}}$	0.23	0.44	$4.0e^{-2}$	$2.0e^{-2}$	0.23
Condgen	**0.19**	$6.5e^{-2}$	$4.5e^{-2}$	0.1	$5.0e^{-2}$	$1.6e^{-2}$	$1.1e^{-2}$	$5.9e^{-2}$
VQ-T2G	0.23	$6.0e^{-2}$	$3.4e^{-2}$	$8.1e^{-2}$	$1.5e^{-2}$	$\mathbf{1.4e^{-2}}$	$\mathbf{7.1e^{-3}}$	$\mathbf{1.6e^{-2}}$

Table 2: Results of unconditioned graph generation of VQ-T2G against existing graph generator models. Smaller MMD scores are better. Degree: degree distribution, Clust.: clustering coefficient, Orbit: 4-node orbit counts, Spect.: graph Laplacian spectrum.

each topic, to an extent. Contrasting this with VQ-T2G's scores it is clear our model has learned to incorporate the text conditioning in generated graphs far more successfully. On the synthetic dataset, VQ-T2G outperforms Condgen on two metrics.

5.2.2 Unconditioned graph generation

We next evaluate the model in the unconditioned graph generation setting. We again compare against Condgen Yang et al. (2019a) along with the unconditioned models GRAN Liao et al. (2019) and SPECTRE Martinkus et al. (2022). In addition to the graph datasets described in Section 4, we perform one further experiment using a dataset of protein graphs, with results for this listed in Appendix C.

To train VQ-T2G in the unconditioned setting we train the GVQVAE as normal, then train the transformer over only graph tokens (i.e. a text length and text vocabulary of zero). For the unconditioned Wiki ego dataset we reuse the GVQVAE from the conditioned experiments and only train the transformer separately. To perform unconditioned training with Condgen we simply set every graph's feature vector to a single zero then train as normal.

Table 2 lists results for these unconditioned experiments. As with the text-conditioned experiments in the main paper body, we generate the same number of graphs as there are in the test set. We use the same 90/10 dataset split for the unconditioned Wiki ego training as the conditioned training. The two-community dataset is much smaller than our datasets and contains only 500 graphs. For this dataset we use an 80/20 split instead.

5.2.3 Visual evaluation

Examples of generated graphs and their conditioning text from the synthetic dataset are in Figure 2. We compare samples from our model against those generated by Condgen, along with the ground-truth graph matching the caption. While it is clear that neither of the models generate graphs that exactly match the text, the graphs generated by VQ-T2G are far closer to the real graph than those generated by Condgen. It is unsurprising that Condgen's results are not as accurate as the model was not designed for text conditioning. VQ-T2G is able to generate graphs that are approximately the correct topology but still has much room to improve. Longer and more precise text descriptions could assist with this. Also of note is that the conditioning text contains the number of nodes of the graph as an integer. This is likely a non-optimal part to include in the dataset and may be another partial reason for VQ-T2G's performance.

6 Conclusion

In this paper we propose the novel problem of text-conditioned graph generation. We introduce the Vector-Quantized Text-to-Graph Generator (VQ-T2G) for this problem and demonstrate its capability on two datasets of graph-text pairs. The idea of VQ-T2G shares similarities to that of VQ-VAE and DALL-E in its learning of discrete latent variables, with a powerful autoregressive transformer for sampling. Our autoencoder, the GVQVAE, is the first work to adopt discrete latent variables for graph reconstruction or generation. Experimental results demonstrate VQ-T2G successfully learns to generate graphs from text conditioning. The structure of generated graphs is clearly consistent

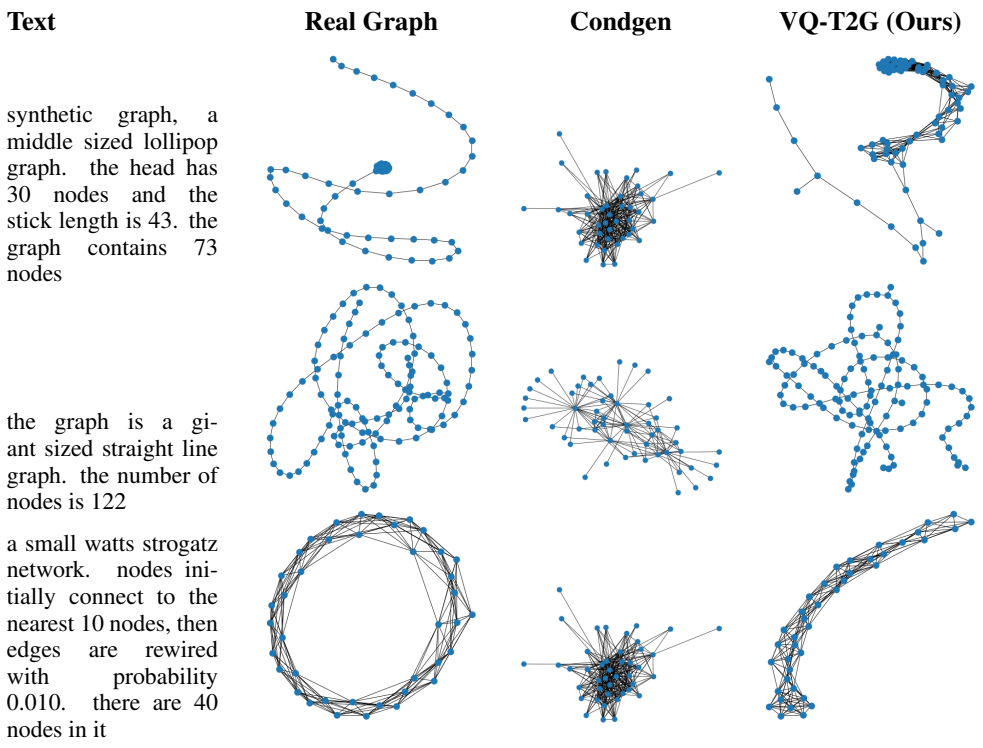

Figure 2: Visualization of graphs from the synthetic dataset generated by Condgen and VQ-T2G, along with the text used to generate the graph and the real graph paired with that text.

with the conditioning text, although there is clear room for improvement in future work. Further, VQ-T2G shows high performance in the unconditioned graph generation setting and is competitive with state-of-the-art unconditioned models.

REFERENCES

Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Anna D Broido and Aaron Clauset. Scale-free networks are rare. *Nature communications*, 10(1): 1–10, 2019.

Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. Scene graphs: A survey of generations and applications. *arXiv preprint arXiv:2104.01111*, 2021.

Cristian Consonni, David Laniado, and Alberto Montresor. Wikilinkgraphs: a complete, longitudinal and multi-language dataset of the wikipedia link networks. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 13, pp. 598–607, 2019.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Lio, and Petar Velickovic. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Dimitar Dimitrov, Philipp Singer, Florian Lemmerich, and Markus Strohmaier. What makes a link successful on wikipedia? In *Proceedings of the 26th International Conference on World Wide Web*, pp. 917–926, 2017.

Mucong Ding, Kezhi Kong, Jingling Li, Chen Zhu, John Dickerson, Furong Huang, and Tom Goldstein. Vq-gnn: A universal framework to scale up graph neural networks using vector quantization. *Advances in Neural Information Processing Systems*, 34, 2021.

Carl Edwards, Tuan Lai, Kevin Ros, Garrett Honke, and Heng Ji. Translation between molecules and natural language. *arXiv preprint arXiv:2204.11817*, 2022.

Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Qipeng Guo, Zhijing Jin, Xipeng Qiu, Weinan Zhang, David Wipf, and Zheng Zhang. Cyclegt: Unsupervised graph-to-text and text-to-graph generation via cycle training. *arXiv preprint arXiv:2006.04702*, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Zhijing Jin, Qipeng Guo, Xipeng Qiu, and Zheng Zhang. Genwiki: A dataset of 1.3 million content-sharing text and graphs for unsupervised graph-to-text generation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 2398–2409, 2020.

Adrian Łańcucki, Jan Chorowski, Guillaume Sanchez, Ricard Marxer, Nanxin Chen, Hans JGA Dolfing, Sameer Khurana, Tanel Alumäe, and Antoine Laurent. Robust training of vector quantized bottleneck models. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2020.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pp. 4257–4267, 2019.

Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. *arXiv preprint arXiv:2204.01613*, 2022.

Igor Melnyk, Pierre Dognin, and Payel Das. Grapher: Multi-stage knowledge graph construction using pretrained language models. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

Shohei Nakazawa, Yoshiki Sato, Sho Tsugawa, Kenji Nakagawa, and Kohei Watabe. Graphtune: A learning-based graph generative model with tunable structural features. *arXiv preprint arXiv:2201.11494*, 2022.

Mark Newman. *Networks*. Oxford university press, 2018.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions. *arXiv preprint arXiv:2106.01098*, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pp. 8821–8831. PMLR, 2021.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p*) models for social networks. *Social networks*, 29(2):173–191, 2007.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Hamed Shirzad, Hossein Hajimirsadeghi, Amir H Abdi, and Greg Mori. Td-gen: Graph generation using tree decomposition. In *International Conference on Artificial Intelligence and Statistics*, pp. 5518–5537. PMLR, 2022.

Filipi Nascimento Silva, Matheus Palhares Viana, Bruno Augusto Nassif Travençolo, and L da F Costa. Investigating relationships within and between category networks in wikipedia. *Journal of informetrics*, 5(3):431–438, 2011.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Int. Conf. Artificial Neural Networks*, pp. 412–422. Springer, 2018.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SkgKO0EtvS`.

Jacob Walker, Ali Razavi, and Aäron van den Oord. Predicting video with vqvae. *arXiv preprint arXiv:2103.01950*, 2021.

Luyu Wang, Yujia Li, Ozlem Aslan, and Oriol Vinyals. Wikigraphs: A wikipedia text-knowledge graph paired dataset. *arXiv preprint arXiv:2107.09556*, 2021.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393 (6684):440–442, 1998.

Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*, 2021.

Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. Conditional structure generation through graph variational generative adversarial nets. In *NeurIPS*, pp. 1338–1349, 2019a.

Xu Yang, Kaihua Tang, Hanwang Zhang, and Jianfei Cai. Auto-encoding scene graphs for image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10685–10694, 2019b.

Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

A Training details

Models were trained using a single NVIDIA GeForce RTX 3080 GPU with 10GB VRAM. We used Python version 3.9, and versions of key packages are: PyTorch v1.12.1 with CUDA v11.3, PyTorch Geometric v2.0.4, Transformers 4.21.1. Approximate training times for datasets (the GVQVAE and transformer combined) are: 7 hours for the Wiki ego dataset (both conditioned and unconditioned), 6 hours for the synthetic dataset, and 1.5 hours for the two-community dataset.

Wiki ego dataset The GVQVAE was structured as follows. For the encoder, GraphSAGE layers have 128 channels followed by the linear layers with 2400 hidden units. This output is projected down to a representation of 120 codes, each with dimension $D = 16$. The codebook has size $K = 448$. In the decoder, the first layer has 2800 hidden units, followed by 3600 units in the second and third layers. The final layer output of size $N \times d_{out}$ where $d_{out} = 20$, followed by the inner product to make the probabilistic adjacency matrix $\tilde{A} \in \mathbb{R}^{160\times160}$. Training was done for 300 epochs with a batch size of 64, with initial encoder and decoder learning rate of $1e-4$. The codebook learning rate is set to 10 times higher than that of the encoder and decoder. The transformer is structured as follows. Texts are tokenized with byte-pair encoding. A vocabulary size of 4096 is used for the texts. An additional 448 tokens, representing the graph tokens (codebook IDs), are then added to this vocabulary. The transformer model has 4 layers each with an embedding size of 256. We use multi-head attention with 4 heads. The transformer is trained for 750 epochs with a batch size of 32 and initial learning rate $2e-4$.

Synthetic dataset The model setup and training for the synthetic dataset is similar to that of the Wiki ego dataset, other than the following differences. The codebook has size $K = 256$. The text vocabulary for the transformer has size 512. Transformer embedding size was 128.

Wiki ego dataset, unconditioned For the unconditioned Wiki ego setting we use the same GVQ-VAE checkpoint as was trained for conditioned setting. We simply make a copy of the model folder and train a separate transformer model there. The text vocabulary size is set to zero as no texts are used. We keep the start-of-graph token at the beginning of sequences and train only on the encoded graphs. Training is run with a batch size of 32 and and learning rate $2e-4$. Without texts to be jointly learned the training converged much more quickly and was stopped at 500 epochs.

Two-community dataset The model for the two-community dataset was much smaller as the dataset is less complex. The GVQVAE was structured as follows. For the encoder, GraphSAGE layers have 64 channels followed by the linear layers with 480 hidden units. This output is projected down to a representation of 60 codes, each with dimension $D = 8$. The codebook has size $K = 64$. In the decoder, the first layer has 1024 hidden units, followed by 2048 units in the second and third layers. At the output we set $d_{out} = 16$. Training was done for 200 epochs with a batch size of 8, with initial encoder and decoder learning rate of $1e-3$. The codebook learning rate is set to 10 times higher than that of the encoder and decoder. With a text and graph vocabulary of 0 and 64 respectively, we use a much smaller transformer too. We keep a 4-layer model with 4 heads, but use an embedding dimension of 64. The transformer is trained for 250 epochs with a batch size of 8 and initial learning rate $1e-5$.

B TEXT-GRAPH DATASET CONSTRUCTION

Details of our two curated text-graph paired datasets are outlined below, including our method and reasons to construct them. We also list some basic statistics of the graphs and texts.

B.1 WIKIPEDIA EGOS

The Wiki ego dataset contains two-hop egocentric networks sampled from subgraphs of the full English Wikipedia link network. We construct eight of these subgraphs then sample 1000 ego graphs from each, for a total dataset size of 8000 graphs. These are referred to as the *topic-wise subgraphs* as they each come from exploring sections of the Wikipedia link network containing articles related to a particular area. The reason for using these topic-wise subgraphs over the full link network is as follows.

Firstly, similar to many real-world large graphs Watts & Strogatz (1998) the Wikipedia link network exhibits a power-law degree distribution Dimitrov et al. (2017). As such, there is a high propensity for articles (nodes) to contain links to or from at least one high-degree node. The presence of these high-degree nodes cause many two-hop ego networks to be extremely large. Additionally, these nodes may not contain high relevance to the ego node. Even if the ego graph sizes were not a concern, the induced subgraph of the hub node and its direct neighbors would appear frequently in

ego samples and bias our dataset. That is, a model's performance may be seemingly inflated if it were to overfit to one or a few of these high-degree node neighborhoods.

Secondly, the topology of the link structure differs between topics across Wikipedia, such as links in *Mathematics* being sparse compared to *Physics* which has a dense core Silva et al. (2011). Therefore by only considering local neighborhoods of distinct areas from the link network, ego sampling will produce graphs with distinct topology, related to the respective areas. Joining each ego network with a portion of the content of the ego node (article) gives us the opportunity to jointly learn on the graphs and text. Moreover, with distinct differences in graph topology by topic, the performance of text-conditioning in generated graphs may be clearly evaluated.

Ego dataset construction The topic-wise subgraphs of the Wikipedia link network from which the ego graphs are sampled from must be sufficiently large for repeated ego sampling. We find topical subgraphs of size $8000 \leq |V| \leq 12000$ are appropriate and we construct them as follows. From a category selected at random, add the articles in it to a list, then traverse the subcategories, subsubcategories, and so on, adding to the list all articles from each up to depth 3 (or further depending on the total articles). The topical subgraph is then the largest connected component of the induced subgraph from the link network containing each of those articles. Note that instead of the full English Wikipedia link network we opt to use the 2018 link network from WikiLinkGraphs Consonni et al. (2019). This is a cleaned version of the full link network which for example, discards edges if the link does not appear in the main article text. As the WikiLinkGraphs dataset only contains the graph structure, we supplement it with the category hierarchy and article text collected from the Wikipedia API. This gives us only the pages in similar semantic areas to that of the initial category, so ego graphs sampled from these induced subgraphs are generally topical overall. If we were to sample our ego graphs from the entire link network rather than topical subgraphs we would (1) have ego nets that rapidly grow to unfeasible sizes, (2) include articles far from the topic of the centre node, due to hub-type pages.

The eight top-level categories traversed from were: *Cooking*, *Artificial intelligence*, *18th-century literature*, *Astrophysics*, *Military of Australia*, *Historic sites in Asia*, *Tourism in Germany*, and *Horticulture*. There is minimal overlap in the articles in each set, most pairs have fewer than five articles overlapping. Any article occurring in more than one of these sets is excluded from selection as the center of an ego graph. When sampling two-hop egos we select ego nodes randomly without replacement and stop after reaching 1000 graphs with $60 \leq |V| \leq 160$. For each ego graph's paired we use the concatenation of the ego node's title and first sentence, collected from the Wikipedia API. Graphs in the complete dataset have an average of 109 nodes. Texts have an average character length of 183, and range between 26-571 characters.

B.2 SYNTHETIC DATASET

The dataset of synthetic graphs contains fifteen varieties (or classes) of both random and non-random synthetic graphs with $20 \leq |V| \leq 160$. The varieties are: barbell graphs, lollipop graphs, ring-of-cliques graphs, standard 2D grid graphs, triangular lattice graphs, hexagonal lattice graphs, Watts-Strogatz random graphs, star graphs, path graphs, binary trees, ternary trees, starlike trees, banana trees, firecracker graphs, sunlet graphs, helm graphs, and fan graphs. There are 3186 graphs in total. We set a maximum of 500 of each graph type to ensure a balanced dataset.

Texts for each graph are constructed methodically. A two sentence caption is created by randomly selecting a description for each of the graph class, size, and structure. Each graph has a unique text caption. Texts do not precisely specify the entire graph topology, they serve as an approximate description. Graphs in this dataset have an average of 86 nodes. Texts have an average character length of 137, and range between 50-228 characters.

C EXTENSION TO GRAPHS WITH 500 NODES

We perform one final experiment to investigate the scalability of VQ-T2G on a dataset of 918 protein graphs with up to 500 nodes. This is a commonly used dataset for evaluating unconditioned graph generative models Liao et al. (2019); Martinkus et al. (2022). Results for this experiment are in table

	Proteins			
	Degree	Clust.	Orbit	Spect.
GRAN	$2.0e^{-3}$	$\mathbf{4.7e^{-2}}$	0.13	$5.1e^{-3}$
SPECTRE	$\mathbf{1.3e^{-3}}$	$\mathbf{4.7e^{-2}}$	$\mathbf{2.9e^{-2}}$	$\mathbf{2.0e^{-3}}$
VQ-T2G	0.14	0.12	0.36	$4.5e^{-2}$

Table 3: Results of unconditioned graph generation on the proteins dataset. Smaller MMD scores are better. Degree: degree distribution, Clust.: clustering coefficient, Orbit: 4-node orbit counts, Spect.: graph Laplacian spectrum.

3. Again we compare against GRAN and SPECTRE in the unconditioned setting. Notably VQ-T2G does not come close to beating these models in any of the metrics.

Datasets of this nature are difficult for VQ-T2G. Contributing factors include the larger maximum number of nodes, the smaller dataset size (compared to our text-graph datasets), and the complex structures found in real-world graphs such as this. In particular, transformers are well-known to be data hungry, and this dataset is very small for using one. Note that while the two-community dataset is even smaller with 500 total graphs, those graphs are relatively less complex, with a lower maximum size and less variety in sizes so the model is able to handle it. It is the combination of large graphs, complex graphs, and relatively small dataset that leads VQ-T2G to perform especially poorly.

While a larger dataset is likely to help VQ-T2G, another key factor in its low performance here is the one-shot generation of the probabilistic adjacency matrix followed by sampling edges independently. As graphs get larger, there will be key structural features the independent sampling is unable to capture effectively and the performance will degrade.

This paper primarily focused on the introduction of the text to graph problem, not unconditioned graph generation able to scale to large graphs. However, improving scalability may be a direction for future research.