Scene graphs are powerful abstractions that capture relationships between objects in images by modeling objects as nodes and relationships as edges. Generation of realistic novel scene graphs has applications like scene synthesis and data augmentation for supervised learning. Existing graph generative models are predominantly targeted toward molecular graphs, leveraging the limited vocabulary of atoms and bonds and also the well-defined semantics of chemical compounds. In contrast, scene graphs have much larger object and relation vocabularies, and their semantics are latent. To address this challenge, we propose VARSCENE, a variational autoencoder for scene graphs, which is optimized for the maximum mean discrepancy (MMD) between the ground truth scene graph distribution and distribution of the generated scene graphs. VARSCENE views a scene graph as a collection of star graphs and encodes it into a latent representation of the underlying stars. The decoder generates scene graphs by learning to sample the component stars and edges between them. Our experiments show that our method is able to mimic the underlying scene graph generative process more accurately than several state-of-the-art baselines.

1. Introduction
Scene graphs (Johnson et al., 2015; 2018; Chang et al., 2021) have emerged as a popular data structure to represent images. The nodes in these graphs represent objects and their attributes, and edges capture relationships between them. They are a succinct and human-consumable summary of the content within the image. Their popularity is partially derived from the availability of rich datasets (Lu et al., 2016; Krishna et al., 2016) that contain ground-truth scene graphs associated with images. Extracting scene graphs from images continues to be a very active field of research (Tang et al., 2020b; Yang et al., 2018; Gu et al., 2019). The success of scene graphs can be seen in the range of visual understanding tasks that utilize them: retrieval (Schroeder & Tripathi, 2020; Wang et al., 2020; Maheshwari et al., 2021), editing (Dhamo et al., 2020), question answering (Teney et al., 2017) and captioning (Milewski et al., 2020).

While there has been progress in the field of image generation, the end goal of achieving rich, diverse and complex scenes is far from satisfactory (Casanova et al., 2020; Li et al., 2019). In the current paper, we propose the novel task of scene graph generation as a stepping stone to scene synthesis. Here, our goal is to learn to synthesize scene graphs based on a set of training examples of scene graphs. Our work is relevant to the theme of generating new and complex scenes, where scene graphs are often used as a means for conditioning image synthesis (Johnson et al., 2018). We decouple the generation of scene semantics (sets of objects and how they relate to each other) from its visual manifestation, and focus specifically on the first part in the current paper. Where required, we will use existing methods to convert our synthesized scene graphs to images (Johnson et al., 2018; Tseng et al., 2020).

Driven by the success at text and image synthesis, there is much recent interest in deep generative models for graphs, e.g., GraphVAE (Simonovsky & Komodakis, 2018), GraphRNN (You et al., 2018), NeVAE (Samanta et al., 2019), MoiGAN (De Cao & Kipf, 2018), GRAN (Liao et al., 2019), etc. However, most of them are tailored to molecule discovery and cannot be effectively used to generate scene graphs, as we demonstrate in our experiments. Very recently, Garg et al. (2021) proposed an unconditional scene graph generator, which, however, is not optimized to capture the underlying data distribution.

1.1. Present Work
We propose VARSCENE, a novel variational autoencoder, specifically designed for scene graph generation. (We use graph ‘generation’ and ‘synthesis’ interchangeably.) In contrast to prior VAE-based graph generative models which maximize Evidence Lower Bound (ELBO) for model training, VARSCENE is optimized to generate graphs having minimum distribution discrepancy with the ground truth
We evaluate the training graph distribution, while using its generative capacity to introduce plausible variations. We thus obtain a model from which we can sample graphs representing realistic scenes. Moreover, due to the encoder-decoder architecture, our approach facilitates conditional scene graph generation, which would otherwise not be possible.

Generating scene graphs with meaningful semantics is a challenging task. Designing simplistic masking functions, as for molecular graphs, does not work in this context, since the notion of underlying semantics is far more complex than in molecules. Responding to this challenge, VarScene views a scene graph as a collection of stars, where a star is comprised of a ‘hub’ node with its incident edges (‘spokes’). Our encoder embeds the graph into a collection of latent representation vectors based on its component stars. At the other end, unlike most existing generative models (You et al., 2018; Samanta et al., 2019; De Cao & Kipf, 2018), our decoder does not expand the graph by generating a node or an edge at a time, but instead by sampling stars. Specifically, it first samples stars from a trainable distribution in a sequential manner, and then connects these stars to generate semantically meaningful scene graphs. Such a model allows us to generate novel scene graphs while retaining the semantic correlations between the different components of a scene. In contrast to generative models for molecular graphs, which work with few node and edge labels, VarScene is able to generate (and be trained on) scene graphs consisting of a large number of node and edge labels.

In theory, the variational autoencoder described above should be able to learn the underlying data distribution, given sufficient training data. However, due to the use of an approximate posterior and lower bound of the true objective, it may not show optimal performance in practice. In response, we further re-train the decoder to directly minimize the maximum mean discrepancy (MMD) between the distributions of generated and ground truth scene graphs. Such a construction allows our method to effectively trade-off between the underlying MMD and the fidelity to the prior decoder model previously obtained using ELBO maximization.

We evaluate VarScene on three real world datasets. Our experiments show that VarScene is able to mimic the underlying distribution of scene graphs more accurately than several baselines.

2.1. Graph Generative Models

Driven by the success of text and image generative models, there is much recent interest in deep generative models for graphs, primarily designed for molecule discovery, e.g., GraphVAE (Simonovsky & Komodakis, 2018), GraphRNN (You et al., 2018), NeVAE (Samanta et al., 2019), MolGAN (De Cao & Kipf, 2018), GRAN (Liao et al., 2019), etc. However, the edge labels in molecular graphs are restricted by the types of bonds present in chemical compounds, which does not exceed 10. On the other hand, the node labels correspond to the unique elements, which is typically <15 in the ZINC (Sterling & Irwin, 2015) and QM9 (Blum & Reymond, 2009; Rupp et al., 2012) datasets. In contrast, the distinct number of node and edge labels in scene graph datasets are significantly higher than those in molecular graphs. E.g., Visual Genome (Krishna et al., 2016) has 75,729 unique node labels and 40,480 unique edge labels. Even the smaller Visual Relationship Detection dataset (Lu et al., 2016) has 100 unique node labels and 70 unique edge labels. Apart from the smaller number of node and edge labels, molecules and protein compounds follow well-defined graph semantics, e.g., valency rules of atoms, non-existence of strained bridges and rings, and occurrence of specific motifs in molecules. These place constraints on the set of plausible graphs, which can be exploited by generative models for molecular graphs. Codifying knowledge about the visual world to aid the training process may not be straightforward in the context of scene graphs.

2.2. Scene Graph Extraction from Images

A graph where the nodes represent objects with edges being relationships between them encapsulates the semantic content of an image in a succinct manner. We refer to the task of constructing such a graph from the visual modality as scene graph extraction (although it is sometimes called ‘generation’, which can be confusing). Literature associated with this (essentially inference) task is vast (Xu et al., 2017; Yang et al., 2018; Li et al., 2018a; Tang et al., 2020b), with alternatives characterized by the modeling approaches — e.g., do they use conditional random fields or recurrent/convolutional network variations, are the models trained in an adversarial manner, and so on. Most of these models’ inputs are derived from the current image alone — these would be visual features computed from regions of interest that help detect the object type or identify the relationship between previously identified objects (Li et al., 2017; Dai et al., 2017; Zhang et al., 2017; Klawonn & Heim, 2018). More recently, inference methods aided by external knowledge have been developed. This knowledge can take the form of word embeddings (Lu et al., 2016) and knowledge graphs (Zareian et al., 2020). The task considered in the current paper is that of scene graph synthesis, where we do not have access to the image modality. In the absence

---

1Our code is accessible at https://cse.iitb.ac.in/~abir/codes/varscene.zip
of this visual information, toward synthesizing realistic and plausible scene graphs, we mirror the existing practice of utilizing information from the linguistic domain in the form of word embeddings of the objects and relationship names.

2.3. Image Generation from Scene Graphs

While there has been progress in the field of image generation, the end goal of achieving rich, diverse and complex scenes of high quality still needs work (Caesar et al., 2018). In the current paper, we propose that an existing collection of scene graphs can be utilized to train a generative model of such graphs. Sampling from such a model provides us with the semantic content for a novel scene. The sampled graph can then be used as input into a conditional image generation model (Johnson et al., 2018; Ashual & Wolf, 2019; Tseng et al., 2020). Conditional image generation models that operate on other input modalities also exist. For example, textual phrases (Xu et al., 2018; Tan et al., 2019), semantic segmentation masks (Tang et al., 2020a), and object positions and layouts (Sun & Wu).

3. Preliminaries

3.1. Notation and Definitions

Scene Graphs, Objects, Relations. Given a collection of directed scene graphs \( G = (V, E) \) along with the set of objects \( T \) and the set of relations \( R \), we denote \( t_u \) as the object represented by the node \( u \in V \) and \( r_e \) as the relation represented by the edge \( e \in E \). For example, if the node pair \((u, v)\) in the scene graph contains the semantic content: “car \(\rightarrow\) moving on \(\rightarrow\) road”, then we use \( t_u = \text{car}, t_v = \text{road} \) and \( r_e = \text{moving on} \). We denote \( t_u \) and \( r_e \) as the feature vectors for node \( u \) and edge \( e \) respectively. Such feature vectors may be obtained by various means, depending on the application. Here we use BERT embeddings of the names of the object \( t_u \) and the relation \( r_e \). Finally, we define \( \text{nbr}(u) \) as the neighbors of node \( u \).

Star and Neighbor-Stars. Given a graph \( G \) and a node \( u \), the star \( s \) rooted at \( u \) is represented by a pair consisting of the node feature vector \( t_u \) and the multiset of features of edges incident on \( u \). I.e., the representation of \( s \) is \( s := \{ t_u, \{ r_{(u,v)} \mid v \in \text{nbr}(u) \} \} \). We denote \( u \) as root(\( s \)). Here, the star consists of only one node, i.e., the central node, characterized by the underlying object and the associated open-ended edges characterized by the relations. Thus, the identity of \( u \), or the identity or type of its neighbors are not included in the representation of \( s \). Only the object type of \( u \) and the types of incident edges matter. Therefore, it is quite possible that two stars having different root node IDs have identical representation, i.e., \( s = s' \) but \( \text{root}(s) \neq \text{root}(s') \). Given a dataset of graphs \( D \), we denote the set of stars in its “star vocabulary” as \( S(D) = \{ s \mid s \in G, G \in D \} \).

Given a star \( s \) in graph \( G \), we define its neighbor-stars \( \mathcal{N}(s) \) as those stars whose roots are connected to the root of \( s \) via an edge. We also define \( \gamma(s, s') \) as the relation type \( r_e \) of the edge \( e \) that connects root(\( s \)) and root(\( s' \)). Formally, \[
\mathcal{N}(s) = \{ (s', \gamma(s, s')) \mid (\text{root}(s), \text{root}(s')) \in E(G) \text{ for some } G \in D \}.
\]

Thus, \( \mathcal{N}(s) \) contains pairs of stars \( s \) and relation \( r_e \), where \( s \) and \( s' \) are connected via the edge \( e \). When clear from context, we will drop the edge type and write “\( s \in \mathcal{N}(s) \)” instead of “\( s, \gamma \in \mathcal{N}(s) \)”.

Maximum Mean Discrepancy (MMD). We will use MMD to compute the discrepancy between the distribution of the generated graphs and the corresponding true graph distribution (Goyal et al., 2020; You et al., 2018). We define MMD between distributions \( P \) and \( Q \) as:

\[
\text{MMD}^2(P, Q) = \mathbb{E}_{G_1,G_2 \sim P} k(G_1, G_2) + \mathbb{E}_{G_1',G_2' \sim Q} k(G_1', G_2') - 2 \mathbb{E}_{G \sim P, G' \sim Q} k(G, G').
\]

(1)

Here, \( k(G, G') \) is the kernel induced by a suitable RKHS. We have samples \( D \) and \( D' \) drawn from \( p_1 \) and \( p_2 \) respectively. Thus, an unbiased estimator of MMD, denoted as \( \text{MMD}(D, D') \), is given by

\[
\frac{1}{n^2} \sum_{G_1,G_2 \in D} k(G_1, G_2) + \frac{1}{n'^2} \sum_{G_1',G_2' \in D'} k(G_1', G_2') - \frac{2}{n \cdot n'} \sum_{G \in D, G' \in D'} k(G, G').
\]

(2)

In practice, we use the RBF kernel on suitably chosen representation vectors \( \nu(G) \) and \( \nu(G') \) of the graphs, i.e.,

\[
k(G, G') = \exp(-\|\nu(G) - \nu(G')\|^2/2).
\]

(3)

3.2. Overview of Graph Variational Autoencoder

Graph VAEs are trainable generative models involving an observable graph \( G \) and its latent representations\(^2\) \( Z \), together with three distributions: (i) a prior distribution \( p_0(Z) \) over the latent code \( Z \), (ii) the encoder \( q(Z \mid G) \) which embeds the observed graphs \( G \) into (a distribution over) the latent code \( z \), and (iii) the decoder \( p(G \mid Z) \) which samples observable graphs \( G \) based on the latent representation \( Z \). Computation of the log-likelihood of observables under a VAE model requires marginalization over the distribution of \( Z \), which is usually intractable. Therefore, the evidence lower bound (ELBO) on the log-likelihood is typically maximized:

\[
\sum_{G \in D} \left[ \mathbb{E}_{Z \sim q(\cdot | G)} [\log p(G \mid Z)] + KL(q(Z \mid G) || p_0(Z)) \right]
\]

(4)

The encoder \( q(Z \mid G) \) is intended to approximate the true posterior of \( Z \). Therefore, the quality of the encoder, and thus the tightness of the lower bound, depends on the ex-

\(^2\)Z is usually a set of latent codes associated with embedding vectors of nodes, edges or substructures.
4. **VarScene Model**

In this section, we develop **VarScene** in full detail, formulating the underlying MMD optimization problem and presenting its different components and design decisions.

4.1. Overview

Our goal is to design a variational autoencoder that is able to synthesize realistic scene graphs in both unconditional settings (given a corpus of real scene graphs), and conditional settings (given a corpus and also a specific scene graph as a starting point). While we want the synthetic graphs to exhibit stochastic variation beyond training graphs, we want the distribution of various graph properties to remain close to those measured on the training sample. The cornerstone of our approach is to build the decoder that minimizes the distribution of various graph properties to remain close to those measured on the training sample. We next use this trained decoder to guide the training of **VarScene**.

4.2. Design of MMD-Optimized Decoder \( p^{MMD} \)

**Learning \( p^{MMD} \) Given the True Generator \( p_{true} \).** If we are given the true generator \( p_{true} \), then, in principle, we can directly estimate a generative distribution \( p^{MMD} \in \mathcal{P} \) over scene graphs, which minimizes MMD between \( p^{MMD} \) and the true distribution:

\[
\text{minimize } \text{MMD}(p_{true}, p^{MMD}).
\]

The above setup faces two bottlenecks: (i) It does not have access to the true distribution \( p_{true} \). It can only use the observed scene graphs present in the training dataset. (ii) Specifying \( \mathcal{P} \) without any prior knowledge about the generative process is difficult in practice.

**Learning \( p^{MMD} \) Given Observed Graphs.** Respecting the above challenges, we approximate (5) through several intermediate steps. First, we use the observed scene graphs to estimate \( \text{MMD} \), as we cannot access the true graph distribution \( p_{true} \). Specifically, given \( D \), a set of observed scene graphs, we replace the objective in the optimization (5) with its sample estimate defined in Eq. (2) as follows:

\[
\text{minimize } \mathbb{E}_{D^\prime \sim p_{true}} \text{MMD}(D, D^\prime)
\]

However, absence of any prior knowledge of the underlying generative process \( \mathcal{P} \) makes it difficult to search for \( p^{MMD} \). Now, suppose that we have a trained VAE model \( (p_0, q_0, \hat{p}_0) \) modeled by neural networks with estimated parameters \( \hat{\phi} \) and \( \theta \), where the encoder \( q_0(\cdot | G) \) embeds the graph into a latent representation \( Z \), the decoder \( \hat{p}_0(\cdot | Z) \) generates a graph \( G \) conditioned on \( Z \) and \( p_0 \) is a prior distribution on \( Z \). We can use this trained model to guide the training of \( p^{MMD} \), where we use the same neural network \( \hat{p}_0 \) to parameterize \( p^{MMD} \approx p_0^{MMD} \) with a new parameter vector \( \theta \). To this end, given a trained VAE model \( (p_0, q_0, \hat{p}_0) \) we learn \( p_0 \) which minimizes the required MMD, while penalizing its KL divergence with respect to the trained decoder \( \hat{p}_0 \), i.e., we solve:

\[
\text{minimize } \mathbb{E}_Z \left[ \text{MMD}(D, D(Z)) + \rho \text{KL}(p_0^{MMD}(\cdot | Z) \| \hat{p}_0(\cdot | Z)) \right],
\]

where \( D(Z) \) is a set of generated graphs by \( p_0^{MMD} \) conditioned on the latent representations \( Z \). Here, \( p_0^{MMD} \) and \( p_{\hat{p}_0}^{MMD} \) share the same parameterized class of generative models \( \mathcal{P}(\theta) \). In the outer expectation, \( Z \) can be drawn from the prior distribution \( p_0 \) for unconditional graph generation or from the trained encoder \( q_0 \) for conditional graph generation. Moreover, \( \rho > 0 \) is a tunable coefficient for the KL regularizer which allows us to trade off between minimizing MMD and the fidelity of \( p^{MMD} \) to the original decoder \( p \). Such a KL divergence based regularization allows \( p_0^{MMD} \) to generate plausible scene graphs with meaningful semantics.

Apart from VAEs, such regularization using KL divergence with respect to a prior distribution has also been used in reinforcement learning (Todorov, 2009), social welfare based control (Tabibian et al., 2020), property oriented molecule generation (Samanta et al., 2019), etc. However, our key goal here is MMD optimization for scene graph generation, where we must steer the property of the entire set of generated graphs, whereas the above tasks are concerned with property of one instance, e.g., generating molecule with a desired property, re-ranking tweets for misinformation mitigation, etc.

**Computation of \( \text{MMD}(D, \{G\}) \).** As suggested by Eq. (2) and (3), computation of \( \text{MMD}(D, \{G\}) \) requires us to featureize \( G \) into \( v(G) \). Graph kernels (Vishwanathan et al., 2010) provide some guidance to our choice of \( v \), but judging whether a synthetic graph is "close to natural" requires the comparison of a diverse bouquet of graph properties,
covering complex interactions between topology, node and edge labels. Because our unit of graph generation is a star, we will focus, for the most part, on a definition of \( \nu(G) \) in terms of the number of different stars present in the graph. (For evaluation, though, we will include other important properties.) We gather the universe of stars \( S(D) \) present in the dataset \( D \), defined as

\[
\nu_s(G) = \# \text{ occurrences of } s \text{ in } G
\]

and then compute \( \nu(G) = [\nu_s(G) : s \in S(D)] \), which is then used to compute the required MMD using Eq. (3).

**Gradient Computation for Solving (7).** We solve optimization (7) using a gradient descent algorithm on the finite sample estimate of the underlying objective, i.e.,

\[
C_\theta(\{G\}) = E_Z \left[ \frac{\rho}{|D(Z)|} \sum_{G \in D(Z)} \log \frac{p_{\theta-MM}(G \mid Z)}{p_{\theta}(G \mid Z)} \right].
\]

Computing the gradient of such a quantity may seem difficult, since the samples \( \{G\} \) are drawn from the distribution itself which is being trained. To circumvent this problem, we use the log-derivative trick (Williams, 1992) to estimate the gradient \( \nabla_\theta C_\theta(\{G\}) \) as follows:

\[
\nabla_\theta \left\{ \sum_{G \in D(Z)} \frac{\rho}{|D(Z)|} \log \frac{p_{\theta-MM}(G \mid Z)}{p_{\theta}(G \mid Z)} \right\} + \frac{\rho}{|D(Z)|} \sum_{G \in D(Z)} \left( \frac{1}{2} \log \frac{p_{\theta-MM}(G \mid Z)}{p_{\theta}(G \mid Z)} + \log p_{\theta-MM}(G \mid Z) \right). \tag{10}
\]

The term within the above gradient can be viewed as a pseudo loss function which can be easily coded in any ML library to solve the optimization in Eq. (7).

### 4.3. Neural Architecture of **VARSCENE**

In this section, we present the neural architecture of the various components of **VARSCENE**, beginning with a brief outline.

**Outline.** **VARSCENE** consists of a base variational autencoder \( (p_\theta, q_\phi, p_0) \) and the final generative distribution \( p_{\theta-MM}^0 \). The encoder \( q_\phi(\cdot) \) views a graph as a set of stars \( \{s_0, s_1, \ldots\} \), where we call the star \( s_0 \) to be the pivot star. Then, we consider various stars \( \{s_i \mid \text{DISTANCE(root}(s_0), \text{root}(s_i)) = \Delta \} \) lying at a given number of hops \( \Delta \) from the pivot star \( s_0 \), and obtain an aggregated representation vector \( z_\Delta \) over these stars. Thus, our encoder \( q_\phi(\cdot) \) encodes the stars into the corresponding latent representations \( Z = \{z_0, \ldots, z_{\Delta_{\text{max}}} \} \), where \( \Delta_{\text{max}} \) is the maximum distance of any star from the pivot \( s_0 \). The corresponding decoder \( p_{\theta-MM}^0(\cdot) \) generates stars in a sequential manner by sampling from a softmax distribution conditioned on the latent representations \( Z \). Given a set of training graphs along with their object and relation types, i.e., \( D = \{G\} \) along with the object types \( \{T\} \) and the relation types \( \{R\} \), **VARSCENE** first learns the encoder \( q_\phi(\cdot) \) and decoder \( p_0(\cdot) \) by maximizing ELBO (evidence lower bound) of the likelihood function. More specifically, **VARSCENE** consists of four components:

1. Prior: \( p_0(Z) \) with \( \Delta_{\text{max}} \sim \text{Poisson}(\lambda) \)
2. Encoder: \( q_\phi(Z \mid G, T = \{t_u\}, R = \{r_e\}) \)
3. Base decoder: \( p_0(\{s_i \mid \{\gamma_{ij}\} \mid Z) \)
4. MMD-optimized decoder: \( p_{\theta-MM}^0(\{s_i \mid \{\gamma_{ij}\} \mid Z) \)

where, \( Z = \{z_0, \ldots, z_{\Delta_{\text{max}}} \} \) is the set of star representations at different distances and \( \gamma_{ij} = \gamma(s_i, s_j) \). Figure 1 gives an overview of our encoder decoder architecture.

**Encoder \( q_\phi \).** Given a graph \( G = (V, E) \) along with the object types \( T = \{t_u\} \) associated with the nodes and the relation types \( R = \{r_e\} \), the encoder \( q_\phi(\cdot) \) aims to characterize the scene graphs as a collection of star graphs \( \{s_i\} \) and represent the graph as a collection of embeddings of these stars, i.e., \( Z = \{z_\Delta \mid 0 \leq \Delta \leq \Delta_{\text{max}}\} \). Such an approach is likely to preserve the spatial semantics of the visual content in a scene graph more effectively than existing graph generative models (Samanta et al., 2019; De Cao & Kipf, 2018; Li et al., 2018b) which represent a graph as a collection of node embeddings. In **VARSCENE**, we compute the graph representation \( Z \) in two steps. First, we compute the node and edge embeddings using a graph neural network (GNN) and then combine these to compute star embeddings.

**Node and Edge Embeddings.** Given a graph \( G \) and the object types \( T = \{t_u\} \) and \( R = \{r_e\} \), we compute the representations of the nodes \( x_u \) and edges \( x_e \) using a GNN proposed by Gilmer et al. (2017). More specifically, we start with the initial node representation \( x_u(0) \) which transforms the feature vector \( t_u \) of the object type associated with node \( u \) using a neural network \( F_0^\phi \).

\[
x_u(0) = F_0^\phi(t_u)
\]  

Then, given a depth limit \( K \), we aggregate structural information from \( k = 1, \ldots, K \) hops from each node into an embedding vector in a recursive manner. Formally:

\[
x_u(k) = F_0^\phi(\{x_u(k-1), x_e(k-1), r_{(u,v)}\}) \tag{12}
\]

\[
x_u(k) = F_0^\phi(\{x_u(k-1) \mid v \in \text{nbr}(u)\}) \tag{13}
\]

\[
x_u(k) = F_0^\phi(x_u(k-1), x_e(k-1)). \tag{14}
\]

Here, \( F_0^\phi \) and \( F_0^\phi \) are modeled as universal approximator neural networks; and, \( F_0^\phi \) is a symmetric aggregator.

**Star Embedding.** Given \( S \) as the set of stars in \( G \), we represent each star \( s \in S \) with depth-\( K \) embeddings:

\[
h_s = F^S_0(x_{\text{root}(s)}, \{x_{\text{root}(s)} \mid v \in \text{nbr}(s)\}) \tag{15}
\]

Next, we generate the latent representations \( \zeta_s \) from a normal distribution, which parameterizes the mean and the variance using neural networks \( \mu_\phi(h_s) \) and \( \sigma_\phi(h_s) \), i.e.,

\[
\zeta_s \sim \text{NORMAL}(\mu_\phi(h_s), \sigma_\phi^2(h_s)). \tag{16}
\]
In theory, $\zeta$ encodes the information about the substructures up to $K$ hops away from $s$ and thus the collection of the representations $\{\zeta_s\}$ should be able to accurately characterize the graph. However, this may not be true in practice, as it does not explicitly specify the relative positions of the stars. To ameliorate this limitation, we first randomly draw the pivot star $s_0$ from set of all stars $S$ and compute the distances $\{\Delta_s\}$ of all stars $s \in S$ from $s_0$. Next, we aggregate $\{\zeta_s\} \mid \Delta_s = \Delta$ into a latent vector $z_{\Delta}$ as follows:

$$z_{\Delta} = F_\phi^z(\{\zeta_s \mid \Delta_s = \Delta\}) \text{ for } 0 \leq \Delta \leq \Delta_{\text{max}}$$

(17)

where $\Delta_{\text{max}}$ is the maximum possible distance from $s_0$ to any star $s \in S$. Finally, we represent the graph as $Z = \{z_0, \ldots, z_{\Delta_{\text{max}}}\}$.

Decoders $p_\theta$ and $p_{\theta_{\text{MMD}}}$. The decoders $p_\theta$ and $p_{\theta_{\text{MMD}}}$ share the same neural network structure. They take the latent representations $Z = \{z_0, \ldots, z_{\Delta_{\text{max}}}\}$ as input and generate a new graph $G$. To do so, they incrementally draw new stars $s_j$ from a database of stars $S(D)$ and connect them with previously sampled stars $s_i$, $i \leq j - 1$. The decoders first sample a star $s_0$ and set $\Delta = 0$. Then, at the step $\Delta \geq 1$, they select the stars $\{s_i \mid \text{DISTANCE}(\text{root}(s_0), \text{root}(s_i)) = \Delta - 1\}$ having distance $\Delta - 1$ from $s_0$ and connect one of them with another star $s_j$ using one of the open edges $e$, i.e., $\gamma(s_i, s_j) = r_e$. In this context, $s_j$ is either drawn from the database of stars $S(D)$ or chosen from one of the existing stars in the graph.

To connect a new star $s_j$ at an open edge $e$ of a star $s_i$ lying at a distance $\Delta$ from $s_0$, our decoders first featureize all the potential stars $s := \{r_{(u,v)} \mid v \in \text{nbr}(u)\}$ as $s = [l_{r_{s}}, \sum_v r_{(u,v)}]$; then represent the pair $[s, z_{\Delta}]$ using a logit, and finally, feeds the logit into a softmax distribution. This softmax distribution is used to sample the new star $s'$. Formally, given the snapshot of the scene graph $G_\Delta$ at step $\Delta$, we have:

$$p_\theta(G \mid Z) = p_\theta \left( \{s_i\}, \{\gamma(s_i, s_j)\} \mid Z \right)$$

$$= \prod_{\Delta=0}^{\Delta_{\text{max}}} \prod_{\gamma(s_i, s_j) = \Delta} p_\theta(s_j, \gamma(s_i, s_j) \mid G_\Delta, Z)$$

(18)

where, $\Delta_{\text{max}} = \text{DISTANCE}(\text{root}(s_0), \text{root}(s_i))$. Moreover, we model $p_\theta(s_j, \gamma(s_i, s_j) \mid r \mid G_\Delta, Z)$ as,

$$\text{MASK}(s_i, s_j, r) \{s_j \notin G_\Delta\} \text{exp}(M_\theta(s_j, z_{\Delta}))$$

$$\sum_{s \in S(D) \setminus G_\Delta} \text{MASK}(s_i, s, r) \text{exp}(M_\theta(s, z_{\Delta}))$$

$$+ \text{MASK}(s_i, s_j, r) \{s_j \in G_\Delta\} \left( \{s : \text{MASK}(s_i, s, r) = 1, s \in G_\Delta\} \right).$$

(19)

Here, $M_\theta$ is a neural network and the masking function $\text{MASK}$ aims to ensure that generated scene graphs are semantically meaningful. Specifically, $\text{MASK}(s, s', r) = 1$ encodes that $s'$ is connected to $s$ via the relation $r$ in at least one scene graph in the dataset $D$, i.e., $\text{MASK}(s, s', r) = \{(s', r) \in N(s)\}$. The first term computes the probability when $s_j$ is not present in $G_\Delta$ and is drawn from $S(D)$. The second term computes the probability when $s_j$ is already present in $G_\Delta$.

Prior Distribution $p_0$. $p_0(z_0, \ldots, z_{\Delta_{\text{max}}})$ is modeled using a Normal distribution, i.e., $p_0(z_0, \ldots, z_{\Delta_{\text{max}}}) = \text{NORMAL}(0, 1, \Delta_{\text{max}})$. Finally, we maximize ELBO($\theta_0, p_\theta$)
Table 1. Dataset statistics where Folds = |Dtr| : |Ddev| : |Dtest|.

<table>
<thead>
<tr>
<th>Dataset</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual Genome</td>
<td>110000</td>
<td>81:9:9</td>
<td>3.23</td>
<td>2.31</td>
<td>16943</td>
</tr>
<tr>
<td>Small-sized Visual Genome</td>
<td>124854</td>
<td>55:12:32</td>
<td>5.31</td>
<td>4.87</td>
<td>150</td>
</tr>
<tr>
<td>Visual Relationship Detection</td>
<td>7721</td>
<td>74:12:12</td>
<td>4.29</td>
<td>3.78</td>
<td>100</td>
</tr>
</tbody>
</table>

5. Experiments

In this section, we provide a comprehensive evaluation of VARSCENE, addressing the following research questions:

**RQ1:** How does VARSCENE compare against state-of-the-art graph generators in terms of its ability to capture the true graph distribution?

**RQ2:** Does the MMD-optimized decoder \( p^\theta_{\text{MMD}} \) provide any performance gain as compared to the base decoder \( p_0 \)?

**RQ3:** How does the quality of the images created from scene graphs generated by VARSCENE compare against the quality of images created via state-of-the-art graph generators?

**RQ4:** What do the generated scene graphs look like? Does the variational code space ensure smooth transition between scene graphs? Appendix D contains additional experiments.

5.1. Experimental setup

**Datasets.** We use three datasets: (i) Visual Genome (VG) (Krishna et al., 2016), (ii) Small-sized Visual Genome (SVG) (Xu et al., 2017) and (iii) Visual Relationship Detection (VRD) (Lu et al., 2016). Visual Genome has a total of 16493 object categories and 8411 relationship categories. VARSCENE can deal with the large object and relationship vocabularies, but recent papers also report results on SVG, a dataset derived from VG by reducing the number of object and relationship categories to 150 and 50 respectively. This reduction is achieved by combining related labels and removing poor-quality labels (Xu et al., 2017; Garg et al., 2021).

For the SVG dataset, we start with pre-processed graphs that were provided with the code of (Garg et al., 2021), that we obtained from the authors. For VG and VRD datasets, we construct graphs directly from the json formatted scene graphs. We took each graph from all three dataset, and split them into weakly connected components, i.e., sub-graphs where each node is reachable from every other node when considering the edges to be undirected.

For the VG dataset, the total number of graphs after splitting exceeded \(~800K\). Due to the large number of graphs, we sampled 90K train graphs at random, and a further 10K validation graphs and 10K test graphs. For the SVG dataset, train-test split was provided in (Garg et al., 2021), hence we used the same set of test graphs and the validation set was obtained by sampling graphs from the train set at random. For the VRD dataset, train-test graphs were provided, but the number of graphs was small. We therefore sampled graphs from the train and test sets to create the validation graphs. Table 1 summarizes details about these datasets.

**State-of-the-art Competitors.** We compare VARSCENE against several competitive graph generative methods: two generative models which were developed for molecule synthesis, viz., (1) DeepGMG (Li et al., 2018b), (2) MolGAN (De Cao & Kipf, 2018); two domain-agnostic graph generative models, viz., (3) GraphRNN (You et al., 2018), (4) GraphGen (Goyal et al., 2020), and one graph generative model specifically aimed at synthesizing scene graphs, similar to our setup, i.e., (5) SceneGen (Garg et al., 2021).

**Training, Validation and Testing.** Given a dataset of scene graphs \( D \), we split them in training \( (D_{tr}) \), validation \( (D_{dev}) \) and test \( (D_{test}) \) folds, where the exact size of these folds \( |D_\bullet| \) varies across datasets due to their varying sizes (see Table 1). We first train the encoder \( q_\phi \) and the base decoder \( p_0 \) using the training set \( D_{tr} \) and then train the MMD optimized decoder \( p^\theta_{\text{MMD}} \) using the validation set \( D_{dev} \). Finally, \( p^\theta_{\text{MMD}} \) is evaluated using \( D_{test} \).

**Evaluation Metrics.** We evaluate the quality of a generative model by measuring the similarity between the graphs generated by the trained model and the graphs in the test set. However, unlike normed-distance in Euclidean feature space, a single similarity measure cannot capture the similarity between two graphs in terms of all the properties. Hence, we resort to several similarity metrics.

**Star-Sim:** The star distribution similarity is measured in terms of the cosine similarity between the distributions of the stars present in the test and generated graphs, i.e.,

\[
\cos \left( E_{G \sim p^\theta_{\text{MMD}}} \nu(G'), E_{G \sim D_{test}} \nu(G) \right) \text{ with } \nu(G) = [\nu_s(G)]
\]

where \( \nu_s(G) \) is the number of occurrences of star \( s \) in \( G \), as defined in Eq. (8).

**Edge-Sim:** It is measured in terms of the cosine similarity between the distributions of the edge bigrams of the form of the tuple \( e_b = (r_{(u \bullet)}, t_u, r_{(u \bullet)}) \) across the test and generated graphs, i.e.,

\[
\cos \left( E_{G \sim p^\theta_{\text{MMD}}} \nu(G'), E_{G \sim D_{test}} \nu(G) \right) \text{ with } \nu(G) = [\nu_{e_b}(G)]
\]

where \( \nu_{e_b}(G) \) is the number of occurrences of edge bigram \( e_b \) in \( G \).

**Node-Sim:** It is measured in terms of the cosine similarity between the distributions of the node bigrams of the form of the tuple \( n_b = (t_u, r_{(u \cdot v)}, t_v) \) across the test and generated graphs, i.e.,

\[
\cos \left( E_{G \sim p^\theta_{\text{MMD}}} \nu(G'), E_{G \sim D_{test}} \nu(G) \right) \text{ with } \nu(G) = [\nu_{n_b}(G)]
\]

where \( \nu_{n_b}(G) \) being the number of occurrences of node bigram \( n_b \) in \( G \).

**SP-K:** Shortest path kernel (Vishwanathan et al., 2010).

**WL-K:** Weisfeiler-Lehman kernel (Vishwanathan et al., 2010).

**NSPD-K:** Neighborhood pairwise distance graph kernel (Costa & De Grave, 2010).

In all kernel-based metrics, we compute the similarity as \( E_{G \sim p^\theta_{\text{MMD}}} K(G', G) \). Note that most of these mea-
Table 2. Performance of different graph generative models which include two variants of our model, i.e., \textsc{VarScene} \text{unc} (unconditional generation, \(Z \sim p_0()\)), \textsc{VarScene} \text{cond} (conditional generation, \(Z \sim q_0(\cdot | G)\)), and the baselines, viz., DeepGMG (Li et al., 2018b), MolGAN (De Cao & Kipf, 2018), GraphGen (Goyal et al., 2020), GraphRNN (You et al., 2018), SceneGen (Garg et al., 2021), measured in terms of cosine similarity between stars (Star-Sim), edge bigrams (Edge-Sim), node bigram (Node-Sim), Shortest path kernel (SP-K) (Vishwanathan et al., 2010), Weisfeiler Lehman Kernel (WL-K) (Vishwanathan et al., 2010), Neighborhood Subgraph Pairwise Distance Kernel (NSPD-K) (Costa & De Grave, 2010). In all cases, the encoder \(q_\theta\) and the base decoder \(p_\theta\) were trained using the training set \(D_\text{tr}\) and the MMD optimized decoder was obtained using the validation set \(D_\text{dev}\). In all cases, the weight of the KL divergence term in Eq. (7) was set as \(\rho = 1000\). Numbers in bold (underline) indicate the (second) best performer.

### 5.2. Results

**Comparison with Baseline Graph Generators.** We first address research question RQ1 by comparing \textsc{VarScene} against baselines, in terms of the metrics defined above. We present results for two variants of \textsc{VarScene}, viz., unconditioned generative model \textsc{VarScene} \text{unc} where \(Z \sim p_0()\) and conditional generative model \textsc{VarScene} \text{cond} where \(Z \sim q_0(\cdot | G)\) with \(G \in D_\text{tr}\). Table 2 summarizes the results. We make the following observations.

(1) \textsc{VarScene} outperforms all the competitors in terms of Star-Sim, SP-Kernel and WL-Kernel with a significant boost. There is no consistent winner between \textsc{VarScene} \text{unc} and \textsc{VarScene} \text{cond}. Since \textsc{VarScene} \text{cond} generates graphs which are structurally similar to the existing training graphs, one may expect that it is likely to mimic the underlying distribution more closely than \textsc{VarScene} \text{unc}. However, this need not always be the case, if there is a large empirical distribution shift between finite training and test folds. \textsc{VarScene} \text{unc} often captures such drift, as it does not look into the training graphs during new graph generation.

(2) SceneGen outperforms the other baselines by a substantial margin in a majority of the cases. Since it is specifically aimed at scene graph generation, it is able to capture the underlying scene graph distribution more effectively than \textsc{VarScene} \text{unc}. However, this need not always be the case, if there is a large empirical distribution shift between finite training and test folds. \textsc{VarScene} \text{unc} often captures such drift, as it does not look into the training graphs during new graph generation.

(3) Among the other baselines, DeepGMG shows good performance in terms of WL-Kernel or NSPD-Kernel across Visual Genome and Visual Relationship Detection datasets. Although it was primarily applied for molecule generation, its design choice is quite domain agnostic. The performance of MolGAN is extremely poor due to its design choices that are specifically aimed at molecule generation.

(4) While \textsc{VarScene} outperforms SceneGen in terms of edge and node bigram similarity measures for the Visual Genome dataset, SceneGen appears to be the best performer for the two other datasets. However, this apparent superior performance of SceneGen may be misleading — careful investigation revealed that SceneGen generates many isolated nodes whose divergence from the test distribution cannot be captured by bigram similarity.

Table 3 summarizes the effect of this augmented set of relation types, which shows that SceneGen actually performs not as well as \textsc{VarScene} \text{cond}.

**Effect of MMD-Optimized Decoder.** Next, we address re-
search question RQ2. Specifically, we compare the quality of the scene graphs generated by our MMD optimized decoder \( p_{\theta}^{\text{MMD}} \) with the base decoder \( p_{\theta} \). Table 4 summarizes the results, which shows that our MMD-optimized decoder is able to mimic the true distribution of stars, as well as edge bi-grams, more accurately than the base decoder. Other metrics are reported in Appendix D.

**Image Quality.** Thus far, we have assessed the quality of the generated graphs, rather than the images that might be generated from them. Here, we address research question RQ3 by evaluating the quality of the images corresponding to the synthesized scene graphs. Specifically, we use `sg2lm`\(^3\), a scene graph to image generation system (Johnson et al., 2018) that comes pre-trained on the Small-sized Visual Genome dataset. Similar to Garg et al. (2021), we evaluate the quality of the images using Fréchet Inception Distance (Heusel et al., 2017), Inception Score (Salimans et al., 2016) and Precision & Recall (Sajjadi et al., 2018). Numbers in **bold** (**underline**) show the best (second best) performer. \( \downarrow \) means smaller is better; \( \uparrow \) means larger is better.

**Table 5.** Evaluation of various generative models by assessing the quality of the images obtained from the corresponding scene graphs. Performance is measured in terms of Fréchet Inception Distance (Heusel et al., 2017), Inception Score (Salimans et al., 2016) and Precision & Recall (Sajjadi et al., 2018). Numbers in **bold** (**underline**) show the best (second best) performer. \( \downarrow \) means smaller is better; \( \uparrow \) means larger is better.

(2) DeepGMG outperforms the other baselines in majority of the cases.

**Visualization of Generated Scene Graphs.** Finally, we address research question RQ4, where we visualize the graphs \( G' \) generated by conditioning on an existing graph \( G \). Specifically, we draw \( G' \sim p_{\theta}^{\text{MMD}}(\cdot | Z) \), where \( Z \sim q_{\phi}(\cdot | G) \). Figure 2 provides some samples, which show that VARSCENE is able to generate graphs which provides similar images to the image corresponding to \( G \).

---

\(^3\)https://github.com/google/sg2lm
References


VarScene: A Deep Generative Model for Realistic Scene Graph Synthesis


Sun, W. and Wu, T. Learning layout and style reconfigurable gans for controllable image synthesis. IEEE transactions on pattern analysis and machine intelligence.


A. Broader Impact

Generating realistic and complex scenes is a key task in image retrieval, image editing, question-answering, etc. In this work, we show that it is possible to sample such scene graphs from a distribution estimated from a set of real scene graphs, without explicitly inspecting training images. Any bias (e.g., gender, race) in the training scene graphs may propagate to the synthetic images produced by our system. Our model has sufficient flexibility to attempt to reduce or mitigate such bias. Specifically, one can replace the first term quantifying MMD in our objective function in Eq. (7) with a suitable alternative which can mitigate the biases in the generated graphs.

B. Additional Details of VarScene

We provide more details about VarScene as follows.

t and r. Given an object type t and relation type r, we use BERT embedding (Devlin et al., 2018) to obtain the corresponding representations t and r. Here, we used the code from https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2 and thus have dimension(t) = dimension(r) = 384.

Specifications of the Encoder. We summarize the components of the encoder as follows:

1. The GNN module of qφ contains four networks: F^0_φ, F^edge_φ, F^agg_φ and F^node_φ, which follow similar architecture as in (Gilmer et al., 2017). We summarize their architecture in details as follows:
   • F^0_φ: It consists of a single layer neural network with linear activation function which outputs a 64 dimensional node features.
   • F^edge_φ: It is a single layer neural network with linear activation function. It takes the pair of node features and the corresponding relation type of the edge between them as input and then outputs a 64 dimensional edge embedding vector.
   • F^agg_φ: It is a sum-pool aggregator.
   • F^node_φ: It is a GRU which sequentially takes x_u(k − 1) and x_u(k − 1) as input and outputs a 64 dimensional node embedding.

2. F^S_φ is kept the same as F^node_φ. It is a GRU which is supplied only one element, the concatenation of x_root(s) and sum({x_{root(s)},v} | v ∈ nbr(root(s))}). It outputs a 64 dimensional vector.

3. µφ in Eq. (16) is a two layer neural network which consists of a 128 dimensional hidden layer, ReLU activations and outputs a 64 dimensional mean vector.

4. σφ in Eq. (16) is a two layer neural network which consists of a 128 dimensional hidden layer, ReLU activations and outputs a 64x64 matrix.

5. F^z_φ in Eq. (17) is a sum aggregator.

Specification of the Decoder. Here, Mθ in Eq. (19) is a two layer neural network which consists of a 32 dimensional hidden layer, ReLU activations and outputs a 16 dimensional mean vector.

C. Details About Experiments

C.1. Implementation Details of Baselines

DeepGMG. We used the version released with the official PyTorch implementation of Goyal et al. (2020)4.

MolGAN. We used an open-source PyTorch implementation (De Cao & Kipf, 2018)5.

GraphRNN. We used the label adapted version released with the official PyTorch implementation of Goyal et al. (2020)6.

GraphGen. We used the official PyTorch implementation (Goyal et al., 2020)7.

SceneGen. We obtained the code from the authors (Garg et al., 2021).
We summarize the total number of parameters in our model and the baselines in Table 6, which shows that our model uses minimum number of parameters and still outperforms all the baselines. Moreover, note that, our number of parameters is independent of the dataset size, whereas the complexity of baseline models widely varies across most of the datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameter Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepGMG</td>
<td>VG</td>
</tr>
<tr>
<td>MolGAN</td>
<td>623,102,433</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>19,561,966</td>
</tr>
<tr>
<td>GraphGen</td>
<td>25,058,630</td>
</tr>
<tr>
<td>SceneGen</td>
<td>277,012</td>
</tr>
<tr>
<td><strong>VARScene</strong></td>
<td><strong>221,249</strong></td>
</tr>
</tbody>
</table>

Table 6. Number of parameters in each trainable model.

C.2. Hyperparameters

**VARScene.** For the base decoder \((p_0)\), we trained the model for 1000 epochs. For the optimized decoder \((p_\text{MMD})\), we again trained for 1000 epochs. The batch size \(|B| = 1024\). For training \(p_0\), the learning rate was kept as \(10^{-4}\). For \(p_\text{MMD}\), the learning rate was kept as \(10^{-5}\). Moreover, we used the Adam optimizer with a \(10^{-5}\) weight decay factor.

**SceneGen.** We trained their model for 300 epochs. The batch size was \(|B| = 64\). A step learning rate scheduler was used for both nodes and edges, with initial learning rate \(10^{-5}\), final learning rate \(10^{-4}\) and decay factor 0.95. The Adam optimizer was used for updates. These hyper-parameters were the default values.

**DeepGMG, GraphRNN, GraphGen.** We trained the model for 10000 epochs. Batch size \(|B| = 2048\). Initial learning rate was 0.003 and decayed by a factor of 0.3 after epochs 100, 200, 400 and 800. The Adam optimizer was used for parameter updates. These hyper-parameters were the default values.

**MolGAN.** For VRD and SVG datasets, we trained the model for 12500 epochs. For the VG dataset, we trained the model for 800 epochs (since the model took a lot of time for training). Batch size \(|B| = 16\). Learning rate was kept as \(10^{-4}\). The Adam optimizer was used for updates with \(\beta_1 = 0.5, \beta_2 = 0.999\). These hyper-parameters were the default values.

C.3. Infrastructure details

We performed all experiments on an Intel Xeon server with 1 TB RAM running Ubuntu 18.04.5 LTS. Our code ran on any of a TITAN RTX GPU with 24 GB RAM and two TITAN X (Pascal) GPUs with 12 GB RAM each.

C.4. Evaluation metrics for Image Synthesis

Let \(\mathcal{I}_r\) be a set of real images, and \(\mathcal{I}_g\) be a set of images that were output by an image generation model. Metrics that compute the quality of the generated set typically use a feature representation \(\mathbf{x}_r\) and \(\mathbf{x}_g\) respectively for elements of these sets. Typically, the activation values of the penultimate layer of a pre-trained image classification model, most often the Inception v3, is used for this purpose. Let \(\mathbf{X}_g\) be the feature representation matrix for the set of generated images, and the corresponding matrix for real images is given by \(\mathbf{X}_r\). The metrics below utilize these feature representations in different ways, each of which assumes that \(\mathcal{I}_g\) and \(\mathcal{I}_r\) are of the same size.

**Fréchet Inception Distance.** (Heusel et al., 2017): The FID is a distance between two multivariate Gaussians each of which is represented by a mean and covariance matrix: \((\mathbf{m}_r, \mathbf{C}_r)\) and \((\mathbf{m}_g, \mathbf{C}_g)\). The FID is given by \(\|\mathbf{m}_r - \mathbf{m}_g\|_2^2 + \text{trace}(\mathbf{C}_r + \mathbf{C}_g - 2(\mathbf{C}_r \times \mathbf{C}_g)^{0.5})\). In the context of evaluating the output of models that generate images, the summary statistics \((\mathbf{m}_r, \mathbf{C}_r)\) are computed over the feature representation matrices \(\mathbf{X}_r\) as defined above.

**Inception Score.** (Salimans et al., 2016): The IS compares the marginal class label distribution over a collection of real images to the conditional label distribution given a set of generated images. It is defined as \(\exp(\mathbb{E}_{I \sim \mathcal{I}}[KL(p(y|I)||p(y))])\). \(I\) here represents an individual output of the image generation model. The vector \(y\) is taken to be the set of labels of an image classification model, again typically taken to be the Inception v3 trained on ImageNet. The intuition is that a good image generation output will have \(p(y|I)\) with low entropy indicating the presence of recognisable objects. But at a set level, we would like a set of varied images, and this is captured by the KL term.

**Precision & Recall.** (Sajjadi et al., 2018): The precision is given \(P(\mathbf{X}_r, \mathbf{X}_g) = \mathbb{E}_{x \sim \mathbf{X}_r} f(x, \mathbf{X}_r)\), and recall \(R(\mathbf{X}_r, \mathbf{X}_g) = \mathbb{E}_{x \sim \mathbf{X}_r} f(x, \mathbf{X}_g)\). The binary function \(f(x, \mathbf{X})\) returns if the sample \(x\) lies within the volume of the set \(\mathbf{X}\). In the current context, Recall represents the fraction of the real images that were generated. While Precision is the fraction of generated...
images that are similar to real images.

D. Additional Experiments

Effect of MMD-Optimized Method. Table 4 compared \( p_0^{\text{MMD}} \) against \( p_0 \) using Star-Sim and Edge-Sim measurements. Here, in Table 7, we present additional similarity and kernel measurements.

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0^{\text{MMD}} )</td>
<td>0.8660</td>
<td>0.5268</td>
<td>0.6226</td>
<td>0.0840</td>
<td>0.1183</td>
<td>0.0143</td>
</tr>
<tr>
<td>( p_0 )</td>
<td>0.5867</td>
<td>0.2588</td>
<td>0.2212</td>
<td>0.1105</td>
<td>0.0135</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0^{\text{MMD}} )</td>
<td>0.9140</td>
<td>0.9372</td>
<td>0.9377</td>
<td>1.4588</td>
<td>1.9218</td>
<td>0.2275</td>
</tr>
<tr>
<td>( p_0 )</td>
<td>0.8988</td>
<td>0.9339</td>
<td>0.9306</td>
<td>0.9426</td>
<td>1.4560</td>
<td>0.2156</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0^{\text{MMD}} )</td>
<td>0.9182</td>
<td>0.6964</td>
<td>0.8111</td>
<td>0.9621</td>
<td>0.6417</td>
<td>0.0633</td>
</tr>
<tr>
<td>( p_0 )</td>
<td>0.7120</td>
<td>0.4195</td>
<td>0.5616</td>
<td>1.9098</td>
<td>0.6944</td>
<td>0.0586</td>
</tr>
</tbody>
</table>

Table 7. Performance for the scene graphs generated by the MMD optimized decoder \( p_0^{\text{MMD}} \) and the base decoder \( p_0 \). Numbers in bold indicate the best performer. We observe that \( p_0^{\text{MMD}} \) outperforms \( p_0 \) in a majority of cases. In all cases, we used conditional graph generation.

Effect of Optimizing Other Similarity Metrics. In Eq. (7), we focus on minimizing MMD between the generated and test scene graphs in terms of the RBF kernel (3). Here, we replace this MMD metric with negative of the six similarity measures used for evaluation, i.e., Star-Sim, Edge-Sim, Node-Sim, SP-Kernel, WL-Kernel and NSPD-Kernel. Tables 8–10 summarize the results for different datasets, and show that the superiority of our method is consistent across them.

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0^{\text{MMD}} )</td>
<td>0.6996</td>
<td>0.4668</td>
<td>0.1579</td>
<td>0.0137</td>
<td>0.0994</td>
<td>0.0144</td>
</tr>
<tr>
<td>( p_0 )</td>
<td>0.6696</td>
<td>0.3760</td>
<td>0.1145</td>
<td>0.0037</td>
<td>0.0378</td>
<td>0.0126</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0^{\text{MMD}} )</td>
<td>0.7702</td>
<td>0.5497</td>
<td>0.4433</td>
<td>0.0558</td>
<td>0.0680</td>
<td>0.0129</td>
</tr>
<tr>
<td>( p_0 )</td>
<td>0.7393</td>
<td>0.3636</td>
<td>0.3726</td>
<td>0.0622</td>
<td>0.0674</td>
<td>0.0126</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0^{\text{MMD}} )</td>
<td>0.7350</td>
<td>0.4709</td>
<td>0.4039</td>
<td>0.0865</td>
<td>0.0753</td>
<td>0.0133</td>
</tr>
<tr>
<td>( p_0 )</td>
<td>0.6819</td>
<td>0.4088</td>
<td>0.3422</td>
<td>0.0989</td>
<td>0.0784</td>
<td>0.0131</td>
</tr>
</tbody>
</table>

Table 8. Performance of different graph generative models (analogous to Table 2) for different optimization objectives in the second stage of our learning for Visual Genome dataset. The metrics against \( \text{VARSCENE} \) , in parentheses, imply that in Eqn. (7) we replace MMD with negative of these similarity metrics. Here \( \text{VARSCENE} \) (Metric) means that in Eq. 7 we replace MMD with \(-\text{Metric}\). Moreover, for last few rows, we maximize the sum of multiple similarities. For example, \( \text{VARSCENE} \) (Metric_1, Metric_2) indicates that in Eq. 7, we replace MMD with \(-(\text{Metric}_1 + \text{Metric}_2)\). We write Star for Star-Sim, Edge for Edge-Sim, Node for Node-Sim, SP for SP-Kernel, WL for WL-Kernel and NSPD for NSPD-Kernel. Recall that \( \text{VARSCENE} \) as-it-is stands for optimizing the discrepancy between the star distributions of the generated and the test set.
### Table 9. Performance of different graph generative models (analogous to Table 8) for different optimization objectives in the second stage of our learning for Small-sized Visual Genome dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepGMG</td>
<td>0.897</td>
<td>0.6401</td>
<td>0.4961</td>
<td>0.1092</td>
<td>0.3589</td>
<td>0.0538</td>
</tr>
<tr>
<td>MolGAN</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0496</td>
<td>0.4084</td>
<td>0.0879</td>
</tr>
<tr>
<td>GraphGen</td>
<td>0.5223</td>
<td>0.6461</td>
<td>0.3713</td>
<td>0.0545</td>
<td>0.2117</td>
<td>0.0432</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>0.2590</td>
<td>0.0775</td>
<td>0.0920</td>
<td>0.3833</td>
<td>0.5991</td>
<td><strong>0.0705</strong></td>
</tr>
<tr>
<td>SceneGen</td>
<td>0.8644</td>
<td><strong>0.8828</strong></td>
<td><strong>0.9311</strong></td>
<td>0.6876</td>
<td>0.5840</td>
<td><strong>0.0663</strong></td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node)</td>
<td>0.9210</td>
<td>0.7446</td>
<td>0.8287</td>
<td>0.6726</td>
<td>0.6058</td>
<td>0.0625</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (WL)</td>
<td><strong>0.9469</strong></td>
<td>0.7268</td>
<td>0.8546</td>
<td>0.9840</td>
<td>0.6342</td>
<td>0.0679</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP)</td>
<td>0.9328</td>
<td>0.7246</td>
<td>0.8576</td>
<td>0.8961</td>
<td>0.6444</td>
<td>0.0691</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (NSPD)</td>
<td>0.9349</td>
<td>0.7231</td>
<td>0.8498</td>
<td>0.8096</td>
<td>0.6215</td>
<td>0.0669</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node, Edge)</td>
<td>0.9254</td>
<td>0.6918</td>
<td>0.8443</td>
<td>0.9661</td>
<td>0.6373</td>
<td>0.0679</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Star, Node, Edge)</td>
<td>0.9308</td>
<td>0.7039</td>
<td>0.8346</td>
<td>0.9214</td>
<td>0.6397</td>
<td>0.0679</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP, Node, Edge)</td>
<td>0.9394</td>
<td>0.7322</td>
<td>0.8577</td>
<td>0.8346</td>
<td>0.6239</td>
<td>0.0684</td>
</tr>
</tbody>
</table>

### Table 10. Performance of different graph generative models (analogous to Table 8) for different optimization objectives in the second stage of our learning for Visual Relationship Detection dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepGMG</td>
<td>0.7459</td>
<td>0.7338</td>
<td>0.6035</td>
<td>0.9977</td>
<td>1.4125</td>
<td>0.2038</td>
</tr>
<tr>
<td>MolGAN</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0149</td>
<td>0.9721</td>
<td>0.2121</td>
</tr>
<tr>
<td>GraphGen</td>
<td>0.6478</td>
<td>0.7586</td>
<td>0.6431</td>
<td>0.3184</td>
<td>0.7964</td>
<td>0.1770</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>0.5455</td>
<td>0.2941</td>
<td>0.7163</td>
<td>0.2125</td>
<td>0.7639</td>
<td>0.1888</td>
</tr>
<tr>
<td>SceneGen</td>
<td>0.8112</td>
<td><strong>0.9486</strong></td>
<td><strong>0.9552</strong></td>
<td>0.6069</td>
<td>1.1274</td>
<td>0.2130</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node)</td>
<td>0.9194</td>
<td>0.9374</td>
<td>0.9403</td>
<td>1.0327</td>
<td>1.5619</td>
<td>0.2301</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (WL)</td>
<td><strong>0.9140</strong></td>
<td>0.9372</td>
<td>0.9377</td>
<td>1.4588</td>
<td>1.9218</td>
<td>0.2275</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP)</td>
<td>0.9216</td>
<td>0.9290</td>
<td>0.9363</td>
<td>1.1150</td>
<td>1.6175</td>
<td>0.2349</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (NSPD)</td>
<td>0.9177</td>
<td>0.9219</td>
<td>0.9343</td>
<td>1.4194</td>
<td>1.9164</td>
<td>0.2290</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node, Edge)</td>
<td>0.9063</td>
<td>0.9296</td>
<td>0.9240</td>
<td>1.2975</td>
<td>1.7934</td>
<td>0.2238</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (WL)</td>
<td>0.8898</td>
<td>0.9281</td>
<td>0.9217</td>
<td>1.1509</td>
<td>1.6318</td>
<td>0.2345</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP)</td>
<td>0.8979</td>
<td>0.9249</td>
<td>0.9251</td>
<td>1.4002</td>
<td>1.9244</td>
<td>0.2294</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (NSPD)</td>
<td>0.9122</td>
<td>0.9317</td>
<td>0.9328</td>
<td>1.1432</td>
<td>1.6431</td>
<td>0.2379</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node, Edge)</td>
<td>0.9010</td>
<td>0.9290</td>
<td>0.9354</td>
<td>1.5389</td>
<td>1.9726</td>
<td>0.2334</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (WL)</td>
<td>0.8893</td>
<td>0.9011</td>
<td>0.9444</td>
<td>1.0247</td>
<td>1.5555</td>
<td>0.2276</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP)</td>
<td>0.8898</td>
<td>0.8858</td>
<td>0.9065</td>
<td>1.2732</td>
<td>1.7864</td>
<td>0.2204</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node, Edge)</td>
<td>0.8891</td>
<td>0.9254</td>
<td>0.9190</td>
<td>1.1111</td>
<td>1.6225</td>
<td>0.2332</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (WL)</td>
<td>0.9029</td>
<td>0.9298</td>
<td>0.9261</td>
<td>1.4332</td>
<td>1.8953</td>
<td>0.2256</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP)</td>
<td>0.9223</td>
<td>0.9324</td>
<td>0.9425</td>
<td>1.1342</td>
<td>1.6316</td>
<td>0.2357</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (Node, Edge)</td>
<td>0.9222</td>
<td>0.9267</td>
<td>0.9455</td>
<td>1.4657</td>
<td>1.9168</td>
<td>0.2292</td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (WL)</td>
<td>0.9103</td>
<td>0.9248</td>
<td>0.9364</td>
<td>1.2289</td>
<td>1.6996</td>
<td><strong>0.2398</strong></td>
</tr>
<tr>
<td><strong>VARSCENE</strong>&lt;sub&gt;0&lt;/sub&gt; (SP)</td>
<td>0.9173</td>
<td>0.9264</td>
<td>0.9416</td>
<td><strong>1.5802</strong></td>
<td><strong>2.0036</strong></td>
<td>0.2339</td>
</tr>
</tbody>
</table>

VarScene: A Deep Generative Model for Realistic Scene Graph Synthesis
VarScene: A Deep Generative Model for Realistic Scene Graph Synthesis

Visual Genome (VG)

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>varScene</td>
<td>0.5963</td>
<td>0.4576</td>
<td>0.4055</td>
<td>0.2293</td>
<td>0.1176</td>
<td>0.0147</td>
</tr>
<tr>
<td>varSceneCond</td>
<td>0.8660</td>
<td>0.5268</td>
<td>0.6226</td>
<td>0.0840</td>
<td>0.0779</td>
<td>0.0135</td>
</tr>
<tr>
<td>varSceneCond (Star)</td>
<td>0.7779</td>
<td>0.5289</td>
<td>0.4732</td>
<td>0.0863</td>
<td>0.0794</td>
<td>0.0134</td>
</tr>
<tr>
<td>varSceneCond (Star)</td>
<td>0.6589</td>
<td>0.2630</td>
<td>0.3659</td>
<td>0.1204</td>
<td>0.0869</td>
<td>0.0133</td>
</tr>
</tbody>
</table>

Small-sized Visual Genome (SVG)

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>varScene</td>
<td>0.9235</td>
<td>0.7049</td>
<td>0.8335</td>
<td>1.0046</td>
<td>0.6556</td>
<td>0.0685</td>
</tr>
<tr>
<td>varSceneCond</td>
<td>0.9182</td>
<td>0.6964</td>
<td>0.8111</td>
<td>0.9621</td>
<td>0.6417</td>
<td>0.0633</td>
</tr>
<tr>
<td>varSceneCond (Star)</td>
<td>0.9339</td>
<td>0.7339</td>
<td>0.8505</td>
<td>0.7679</td>
<td>0.6104</td>
<td>0.0667</td>
</tr>
<tr>
<td>varSceneCond (Star)</td>
<td>0.9235</td>
<td>0.7307</td>
<td>0.8322</td>
<td>0.7654</td>
<td>0.6153</td>
<td>0.0622</td>
</tr>
</tbody>
</table>

Visual Relationship Detection (VRD)

<table>
<thead>
<tr>
<th>Model</th>
<th>Star-Sim</th>
<th>Edge-Sim</th>
<th>Node-Sim</th>
<th>SP-Kernel</th>
<th>WL-Kernel</th>
<th>NSPD-Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>varScene</td>
<td>0.9194</td>
<td>0.9374</td>
<td>0.9403</td>
<td>1.0327</td>
<td>1.5619</td>
<td>0.2301</td>
</tr>
<tr>
<td>varSceneCond</td>
<td>0.9140</td>
<td>0.9372</td>
<td>0.9377</td>
<td>1.4588</td>
<td>1.9218</td>
<td>0.2275</td>
</tr>
<tr>
<td>varSceneCond (Star)</td>
<td>0.9067</td>
<td>0.9294</td>
<td>0.9275</td>
<td>1.0799</td>
<td>1.5911</td>
<td>0.2306</td>
</tr>
<tr>
<td>varSceneCond (Star)</td>
<td>0.9074</td>
<td>0.9210</td>
<td>0.9301</td>
<td>1.4106</td>
<td>1.8871</td>
<td>0.2242</td>
</tr>
</tbody>
</table>

Table 11. Performance provided by two variants of varScene, viz., varScene(RBF) which is the default variant having $k$ as the RBF kernel as defined in Eq. (3) and varScene(Star-Sim) which uses $k(G,G') = \cos(\nu(G), \nu(G'))$. Numbers in bold indicate the best performer.

Effect of Using RBF kernel. Recall that we use RBF kernel in Eq. (3) in our experiments. Here, we compare such an approach against using a simple cosine-similarity based kernel computation, that is, $k(G,G') = \cos(\nu(G), \nu(G'))$. Table 11 summarizes the results, which shows that using RBF kernel provides the performance boost for most similarity metrics.

Efficiency Analysis. Here, we analyze efficiency of different methods. Table 12 summarizes the results, which shows that training the base VAE model of varScene is fastest on Visual Genome which is the largest dataset. For smaller datasets, i.e., Small-sized Visual Genome and Visual Relationship Detection, the training time is comparable with baselines. However, we note that training the MMD optimized decoder is slower in all datasets. The reason for more time in MMD optimization is that we need to sample a set of graphs (we sample 1000 graphs) at each iteration for computing $\hat{\text{MMD}}$ in the objective as per Eqn. 9. For SVG and VRD datasets specifically, the graph generation process takes a higher amount of time. This is because its decoders sample stars beyond a limit (we set this limit as 50). In such cases, we sample graphs repeatedly until we get a valid graph or until we have repeated the procedure for a certain number of times (we set this limit as 20).

<table>
<thead>
<tr>
<th>Model</th>
<th>VG</th>
<th>SVG</th>
<th>VRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepGMG</td>
<td>2.0642</td>
<td>3.6121</td>
<td>2.5645</td>
</tr>
<tr>
<td>MolGAN</td>
<td>0.1274</td>
<td>0.1128</td>
<td>0.1414</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>1.1582</td>
<td>0.0530</td>
<td>0.0435</td>
</tr>
<tr>
<td>GraphGen</td>
<td>0.0635</td>
<td>0.0429</td>
<td>0.0319</td>
</tr>
<tr>
<td>SceneGen</td>
<td>0.1945</td>
<td>0.0650</td>
<td>0.0510</td>
</tr>
<tr>
<td>varScene ($p_\theta$)</td>
<td>0.0334</td>
<td>0.0650</td>
<td>0.4188</td>
</tr>
<tr>
<td>varScene ($p_{\text{MMD}}$)</td>
<td>5.3866</td>
<td>22.1826</td>
<td>24.9556</td>
</tr>
</tbody>
</table>

Table 12. Time (in seconds) for one iteration for a fixed batch size $|\mathcal{B}| = 64$. 