Department of Informatics and Computer Engineering

Interdepartmental Program of Graduate Studies
Information Processing Systems and Machine Intelligence

Generating 3D Indoor Environments with Consistent Styles

Author: Paschalidis Georgios
Supervisor: Assoc. Prof. Psarakis Emmanouil

Patras 2022
The only factor that defines a result is your personal effort. Dedicated to those who never stop dreaming and to my sister.
In the current Master thesis, we address the scene generation task, with our main focus being on indoor scenes synthesis. Existing approaches pose the scene generation task as a layout creation problem. Namely the task is to populate a scene with a set of labelled bounding boxes that correspond to a set of furniture pieces. In particular, these methods typically seek to learn a probability distribution over a set of attributes that define each object such as their shape, category, orientation and position in the scene. During generation, the generated bounding boxes are replaced with 3D objects by retrieving them from a library of assets based on various criteria such as size, object category etc. Naturally, since the object retrieval process is independent from the layout generation there are no guarantees that the generated objects will be coherent in terms of style and appearance. To this end, in this work, we propose a novel generative model for indoor scenes that takes into consideration the per-object style during the generation process. We believe that this is a crucial step towards generating realistic environments. In particular, we build on top of ATISS [18], which is the state-of-the-art indoor scene generation pipeline. Specifically, we extend its capabilities by also incorporating a style prediction module. Furthermore, we also propose a novel retrieval procedure that instead of simply relying on the size to replace bounding boxes with 3D models, takes into account the per-object style. Our experimental evaluation showcases that our model consistently generates stylistically meaningful scenes, (i.e. the nightstands next two a bed or the chairs around the table should have similar appearance), while performing on par with ATISS wrt. the scene generation quality. Finally, we also introduce various metrics that can be used for evaluating the generated scenes in terms of the style coherence.
ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Professor Emmanouil Psarakis. Even before our collaboration, in terms of this thesis, Prof. Psarakis was always willing to adapt the schedule of his lectures so that people who were also working, like myself, could attend the classes and have equal opportunities to the wealth of knowledge. When I resigned from my job, in order to completely focus on my academical obligations, he was eager to advise me, asked me about my future plans and supported my serious decision. Most importantly, I would like to thank him for his trust during the preparation of my Master thesis as well as his valuable comments and suggestions throughout our collaboration.

Moreover, I would like to thank my parents. They were, are and will always be by my side regardless of my decisions. Always supportive with faith and confidence in my abilities, my plans and endeavors.

Furthermore, I must thank my friend Nikos, who believed in me and supported the decision I took eight months ago. I still remember his words. "The only factor that defines a result is your personal effort". To my friend Kostas with whom I have shared so many experiences and dreams, I would like to express my gratitude for his support and faith during all this process. I would also like to thank Aggelos for his contribution. His attitude towards sharing his huge knowledge and abilities at any moment with joy and willingness, in combination with the computational resources with which he always provides me, facilitated incredibly the whole project.

Finally I would like to thank my two siblings Panagiotis and Despoina for all they have done for me during this period of time. First Panagiotis for his push in order to set other goals in my life and for his important help during the whole period of my Master studies. To Despoina, I have to say a huge thanks for all she has done for me so far. Either with her knowledge or with computational resources, she was always there to support and facilitate my efforts. More than anything however I would like to thank her for believing in me the whole time and caring about my concerns as hers. She has always wanted a brighter future for me, a future with prospects and chances, in order to be able to evolve and attain my full potential. I only hope I can pay off all this support, during this period. I owe you guys. PhD loading .....
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>List of Figures</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>List of Tables</td>
<td>xii</td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>RELATED WORK</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Generating scenes using probabilistic grammar</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Generating graph-based scenes</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Generating image-based scenes</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Generating scenes using transformer architectures</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>Generating scenes using radiance fields</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>METHOD</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Loss function definition</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Network architecture</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Extracting object parameters</td>
<td>34</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Class</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Size</td>
<td>35</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Orientation</td>
<td>37</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Translation</td>
<td>37</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Style</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Defining parameters distributions</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>Retrieval</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>EXPERIMENTAL EVALUATION</td>
<td>50</td>
</tr>
<tr>
<td>4.1</td>
<td>Training image Autoencoder</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>Training voxel grid Autoencoder</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Experimental results</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Evaluation</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>Permutation study</td>
<td>68</td>
</tr>
<tr>
<td>4.6</td>
<td>Failure cases</td>
<td>74</td>
</tr>
<tr>
<td>5</td>
<td>CONCLUSIONS</td>
<td>76</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 2.1 A scene graph example. .................................. 6
Figure 2.2 Meta-Sim architecture .................................. 7
Figure 2.3 Architecture of SceneGraphNet ....................... 9
Figure 2.4 Architecture of 3D-SLN ................................. 10
Figure 2.5 A hierarchical graph example. ......................... 12
Figure 2.6 GRAINS model architecture .......................... 13
Figure 2.7 SceneFormer architecture .............................. 18
Figure 2.8 ATISS architecture during inference .................. 20
Figure 2.9 Architecture of GSN generator ....................... 23
Figure 2.10 Architecture of GAUDI decoder ..................... 24
Figure 3.1 The proposed architecture of our model ............. 33
Figure 3.2 Occurrence frequency of objects across bedrooms
and living rooms of the dataset ................................. 36
Figure 3.3 Occurrence frequency of style categories across bed-
rooms and living rooms of the dataset ......................... 39
Figure 3.4 Scene generation with bounding boxes ............... 45
Figure 3.5 Simple style based retrieval .......................... 47
Figure 3.6 Autoregressive style based retrieval ................... 48
Figure 4.1 Learning curves of our image autoencoder ........... 51
Figure 4.2 Image autoencoder reconstructions ................... 51
Figure 4.3 Interpolation between style features of our image
autoencoder ....................................................... 52
Figure 4.4 Learning curves for our voxel grid autoencoder .... 53
Figure 4.5 Voxel grid autoencoder reconstructions ............... 54
Figure 4.6 Learning curves obtained from the trained models
for bedrooms .................................................... 55
Figure 4.7 Learning curves obtained from the trained models
for living rooms ............................................... 55
Figure 4.8 Scene generation using all the trained models in the
case of bedrooms ............................................. 56
Figure 4.9 Scene generation using all the trained models in the
case of living rooms .......................................... 57
Figure 4.10 Scene generation in the case of bedrooms for the
case, where we are using CLIP image encoder and
learnable variances ............................................. 58
| Figure 4.11 | Style similarity measure | 59 |
| Figure 4.12 | Perfect alignment between bounding box and floor layout. | 63 |
| Figure 4.13 | Two cases where the overlapping condition does not hold | 64 |
| Figure 4.14 | Retrieving same objects of different size. | 68 |
| Figure 4.15 | Learning curves obtained from our permutation study. | 71 |
| Figure 4.16 | Failure cases from our models | 74 |
| Figure 4.17 | Failure cases across the dataset | 75 |
## LIST OF TABLES

| Table 4.1 | Mean Style Similarity Measures for bedrooms. | 61 |
| Table 4.2 | Mean Style Similarity Measures for living rooms. | 62 |
| Table 4.3 | Characteristic probabilities and FID score for bedrooms | 66 |
| Table 4.4 | Characteristic probabilities and FID score for living rooms | 67 |
| Table 4.5 | Adding some dependency in distributions parameters prediction, in the case of size and translation. | 69 |
| Table 4.6 | Predicting the parameters of translation and size distribution by using one MLP for each one. | 70 |
| Table 4.7 | Trained variations during permutation study. | 70 |
| Table 4.8 | Metrics for the 8 variations we trained in our permutation study. | 72 |
The main reason why we humans have dominated planet Earth is undoubtedly our intelligence. Our ability to evaluate situations, to decide and act based on various parameters, that are not necessarily related to our survival is one of the most important characteristics that distinguishes us from all other species in our planet. This ability has fascinated humanity for generations and triggered its curiosity, in order to explore it and better understand how it is ”implemented” in our brains. In the recent decades, after years of research on better understanding the human’s perceptual mechanisms, we now try to endow machines with similar perceptual capabilities. This field is called Artificial Intelligence and is the main reason that our world changes rapidly on a daily basis. Artificial Intelligence has rapidly evolved in the recent years primarily due to two factors. The first one being the abundance of annotated data that can be used to train our algorithms and the rapid advancements in hardware systems that facilitate processing large amount of data efficiently.

As it happens with all sciences, Artificial Intelligence can be divided into many branches, in order to be developed faster and applied more effectively in different tasks and problems. One of the most promising fields of Artificial Intelligence, where remarkable achievements have been made over the years is Computer Vision. The basic difference of Computer Vision with other AI fields is the fact that the data that are being used are images or videos. I believe that developing efficient and reliable Computer Vision algorithms is an important step towards achieving true AI. For the majority of people vision is the most important sense. We use our eyes to observe the world and gather information about our surroundings. Apart from that, our eyes and our brain are the most complex organs in our bodies, and some of the processes that are taking part inside them are still unknown to us. The difficulties that arise due to that complexity are creating extremely interesting problems. This is also the reason why a very big part of the scientific community deals with computer vision and why many big companies invest billions every year, in order to accelerate and assist the Computer Vision evolution.

One of the most interesting and intriguing problems in the field of Computer Vision is the generation of 3D indoor and outdoor environments.
The ability to generating realistically looking synthetic scenes, is something totally unique and extraordinary. However except of the impressive part of creating new environments, this ability is something very important for our modern community. There are many examples of people that need such tools in order to facilitate their work. Many architects, civil engineers and decorators are using such applications in order to experiment with their tasks easier and faster, and of course in order to enrich their means of inspiration. Furthermore the computer games industry also relies on that field of computer vision. Virtual reality games are dominating more and more in the preferences of the people. The chance of participating in a virtual reality game as the central hero is totally captivating and promises a totally new and different mean of entertainment, which seemed impossible some years ago. Also the army can benefit from this technology and train virtual soldiers in missions, which take place in foreign and hostile environments. It is obvious that all these are impossible without the ability to generate realistic virtual environments.

In the current Master thesis, we develop a novel neural network architecture for generating 3D indoor environments. Unlike prior methods, we focus on learning a generative model capable of generating stylistically reasonable object arrangements. By observing the results from existing models, we note that despite the fact that they can create new 3D scenes, the relations between the objects do not always look plausible. For example, a generated bedroom with two nightstands could contain a white and a brown nightstand. This is relatively unconventional since typically nightstands tend to have a similar appearance. For that reason our goal in the current project is to create scenes whose objects are related based on their style. But what is style? Is style something that can be formulated in a mathematical way? Someone can argue that style is something fully generic that will not be useful in such tasks. Without a doubt, style, as we perceive it from our everyday life, is something totally subjective. However when we are considering a computer’s perception for style, we need to keep in mind that it has neither our experience in style interpretation nor our ability of arguing for what is stylistically correct and what is not. As a result we define style in a much simpler way. Specifically we assume that style is a parameter which connects objects in a scene mathematically and also prevents scenes from having universally accepted wrong orientations of objects. Having this in mind we can state now that our goal it to generate 3D indoor environments with consistent styles.
Our approach is something totally new and very promising in the field of Artificial Intelligence. Imagine how amazing would be to have a model which can decide about the generation of the scene based on stylistic criteria. Such tasks imitate the ability of humans to decide and make choices based on personal preferences, which is one of the biggest goals of Artificial Intelligence.

In the next chapter, we present the state-of-the-art approaches in 3D indoor scene generation and give a brief explanation of their method, the data that they are using and the way they train their architectures. These approaches are divided into five categories based on the way they generate and represent new scenes, the type of data they are using and the components in their architecture. The first category contains methods that are using probabilistic grammars in order to describe scene’s objects and their parameters. They are trying to learn the rules and the structure of this grammar in order to produce new scenes. The second category which contains the majority of the approaches which we will mention next, refers to methods which represent scenes as graphs and also produce scenes in that form. In order to be able to analyze, learn and interpret the relations between objects in a scene, graph based methods provide a powerful tool, that allows using graphs with multiple nodes and edges, such connections to describe scenes. However in order one scene to be rendered in a graph form, it requires manually design and moreover the relations that will be depicted are based on the point of view of the designer, making as a result the whole approach a bit biased. The third category includes approaches that are using images of scenes, in order to train their architectures and try from them to derive the necessary information of the scenes. Fourth and fifth category which are the most recent and achieve the best results in scene synthesis, have incorporate state of the art networks in their architecture. In the fourth category belong approaches which utilize the architecture of the transformer network [23], while in fifth category approaches which are using neural radiance fields [17]. We introduce a transformer-based approach that extends the capabilities of [18] which is the state-of-the-art in indoor scene generation using 3D super vision. Particularly we seek to built a generative model that autoregressively predicts distributions of attributes for every object in a scene, as in [18], while incorporating and implementing also style prediction for these objects. Using the predicted distributions we can sample them in order to generate new scenes.

In the third chapter, we present our method and the way we insert the style as a parameter in our model. First we give an analytically description
of how our task is captured mathematically and define our optimization objective. Next, we analyze the architecture we are using, we explain its components and their functionality and describe how we apply the previous defined loss in our architecture. In order for the reader to have a better understanding of the whole implementation we also explain the way we collect and extract the data from the dataset which we used in order to be able to train our architecture. For every attribute of the objects we describe the type of distribution we learn through training and the way we optimized its learning process. Regarding the style we present the ways we have followed in order to describe it mathematically. We have modeled style across objects with four different ways and compare the corresponding results both with each other and with ATISS approach. Finally, we also present our retrieval process that ensures that given the predicted parameters from our network we can stylistically coherent scenes.

In the fourth chapter, we present our experiments and results. First we describe experiments regarding how we extract style information from the objects in order to use it during training in our network. The rest experiments refer to the whole architecture and the way we train it. We compare our results using various metrics that evaluate the style consistency among furniture pieces in a scene. We also experiment using two type of indoor scenes, bedrooms and living rooms, in order to showcase the generalization of our model. Finally, we also ablate the optimal order under which our model autoregressively predicts the attributes of each object.

In the last chapter, we present our conclusions and our contributions in the specific problem. We are referring to aspects and topics where our method would be very useful and stating the advantages over other previous approaches. Finally, we also include some proposals for further improvements of this project and also we mention ideas for future work relevant to the topic of this thesis.
In this section we present state of the art approaches upon 3D scene generation. Some of them have been implemented only for outdoor environments, but their structure and methodology could also be applied for generation of indoor scenes. These approaches can be divided into categories according to their implementations. Therefore, the following subsections correspond to each one of these categories.

2.1 Generating Scenes Using Probabilistic Grammar

The concept of probabilistic grammar is met mainly in natural language processing (NLP). It was introduced firstly as Context Free Grammar (CFG) and then by adding the concept of probability in CFG, the method of Probabilistic Context Free Grammar (PCFG) was formed. Both methods try to decode and describe the structure of sentences. Both using an alphabet which consists of terminal and non terminal nodes. The way that these nodes are linked together is described by some production rules, which are predefined by the grammar author. These rules could be applied only in non terminal nodes in order to produce sequences. Terminal nodes are used as the terminal of a sequence and that is why production rules are not applied to them. The term context free means that these rules could be applied in any alphabet despite its content. Therefore, given an alphabet, if we apply these production rules we can create different sequence of nodes and generate different sentences, which can be visualized using tree graphs. The difference between CFG and PCFG is that in PCFG for every possible sequence of the graph we assign a probability, so that all these probabilities sum up to one. However, this idea can be extended in order to model other problems too. When there is the concept of a sequence, such as in sentences, some problems can be modeled using probabilistic grammars. An indoor or outdoor scene can also be described as a sequence of objects, which are placed in specific order according with the relation that connects them. For example, we can imagine an indoor scene as a sequence of a floor, some furniture, objects upon that furniture and even people that interact with the objects inside that scene. Therefore, if we assume that we have a scene, indoor or outdoor, we can describe it as a tree graph by
using a specific probabilistic grammar in order to connect its parts with the appropriate connection rules. If our grammar is context free we can use the same production rules and create different scenes by using the available alphabet of our grammar.

In Meta-Sim [13] they use a probabilistic scene grammar in order to sample scene graphs. Every scene graph is composed by nodes and connection lines. The nodes in the scene graph may have some or any attributes. In [13] they create a generative model, which samples scene graphs from a given grammar, keeps the structure of the scene graphs unchanged and modifies their attributes. The goal of this process is by modifying the attributes of the sampled scene graphs to create new ones.

![Figure 2.1: An example of a scene graph which is composed by nodes, their attributes and the connection edges. Figure from [13].](image)

By rendering these new scene graphs the resulting images must follow similar distribution with the corresponding real ones. The first part of their architecture which modifies the attributes of the scene graphs called Distribution Transformer and is modelled with graph convolutional networks (GCNs). The whole architecture is trained in three phases. In the first one, they use a MSE loss between the new attributes that are generated by the Distribution Transformer and the input attributes. They choose to use this loss in the beginning of training in order to tune the model, due to the fact that the learning task in this case is quite difficult. After some iterations they stop using the MSE loss and instead they optimized their model by using the Maximum Mean Discrepancy (MMD) metric. This metric is a similarity measure between distributions and so by using this as loss function, they want to push the distribution transformer to generate attributes based on a distribution closed to the desired one. Finally in the end of every epoch they
use a meta objective function. Essentially, they use the until that moment trained model and generate new scenes. With the new rendered scenes they train another task network and measure its performance upon a validation set that comes with the grammar. The performance accuracy is subtracted by the previous loss. By doing that they penalize the low performance in the meta objective task and reward the high one. However, this approach due to the fact that preserves the structure of input scene graphs, depends from the validity of that structure, and as a result this a factor that limits the utilization range of this method in specific probabilistic scene grammars.

![Architecture of Meta-Sim](image)

**Figure 2.2:** Architecture of Meta-Sim. By using a probabilistic grammar the model samples scene graphs, whose attributes are modified by using a distribution transformer. The new scene graphs are rendered into new images and the whole architecture is optimized by minimizing the MMD metric and maximizing the performance of these images in a meta objective task. Figure from [13].

In order to alleviate this limitation of Meta-Sim, Meta-Sim 2 [5] was introduced. Opposite with Meta-Sim, this approach utilizes a probabilistic grammar of scenes and images, not in order to sample its scene graphs, but in order to sample its production rules. By sampling one rule at a time step, together with the attributes of the node of interest, with this method they predict the following rules autoregressively. As a result they generate from scratch both the structure and nodes attributes of a scene graph. The generated scene graphs are rendered to new images by using a non differentiable renderer. The non differentiable renderer in combination with the discrete distribution of the generated scene graph structures, were two parameters that make the optimization process of this problem more difficult. Under this condition they project the real and generated scenes into a latent space, by using a feature extractor and they capture the probability
distributions of real and generated latent vectors. Finally, they use the KL divergence loss in order to train their model. However, because the part of sampling rules and generating autoregressively the scene graph structure is non differentiable, in order to minimize the KL divergence loss they use the REINFORCE algorithm which is used in Reinforcement Learning and Robotics.

In [19] they propose a scene grammar variational autoencoder (SG-VAE) that has some similarities with [5]. As in [5] the production rules of the grammar are being learnt. In this approach they use a CFG, which is built entirely by them. By using a dataset with indoor scenes, they searched relations and dependencies between the objects of the scenes, by utilizing a specific algorithm, in order to represent them in casual graphs. When the casual graph is formed they choose particular nodes, by assuming that they are terminal. Beginning from these nodes and following the connections of the constructed casual graph they create production rules. By using then a repetitive algorithm they test these rules in the objects of the dataset and try to maximize the number of the real scenes that can be described by using these rules. When this process is optimized they fed their CFG to a VAE. Given a scene obtained by this CFG, they represent its production rules as one-hot vectors of length equal to the whole number of production rules of this CFG. They feed then their Grammar VAE, with these hot vectors, together with pose and shape parameters of the objects in the scene. The Variational Autoencoder processes in different points hot vectors and pose, shape parameters, until concatenating them after some layers. By doing that when they sample from the latent space a vector and decode it, they obtain not only valid scenes, that follow the production rules, but also the generated scenes take into account the pose and shape parameters of the objects in the scene. As a result the configuration of the final scene becomes more functional and practical and refers to human design.

2.2 Generating graph-based scenes

The graphs in this category must not be confused with the scene graphs that we examined in the previous section. These graphs depict more information about the relations between the objects in the scene and their form is not necessary a tree graph [28]. In [28] they built a model in order to be able to complete partial scenes. The model takes as inputs the objects of the scene and a free spot inside it and finds the object type, which is most likely to occupy that spot. Essentially, it computes the distribution probability over
2.2 Generating Graph-Based Scenes

all object types conditioned in the already objects of the scene and the given empty spot. First of all, they create a graph that represents a given scene. The nodes of the graph correspond to the objects of the scene, while the edges to the relations between them. Using this graph they create a message passing architecture which is trained iteratively, using MLP layers and GRU units. For every node-object they use some of its given parameters, obtained from the dataset (category, centroid position, size) and they encode them as a latent vector. By using the corresponding latent vectors between one target node and the rest in the graph, they feed MLP structures. Particularly, they use one MLP for every pair of nodes between the target node and the rest, in order to create a message representation between these nodes. These message representations are divided in six types. For every type a different MLP structure with its known learnable parameters is used, in order the desired message type to be generated. By having multiple types of messages they manage to decode more efficient the relations between the objects and raise the prediction accuracy of their model.

![Diagram](image)

Figure 2.3: Different type of messages exchange procedure between the target node and the rest of the graph, in order its vectorial representation to be updated. Figure from [28].

The generated messages between one of the above pairs are multiplied with learnable weights. These weights are learnt using an attention mechanism and depict the importance of every message in the prediction process. After this step the now scaled messages are aggregated through equal GRU units, and the outputs are concatenated, in order to form a representation for all the different type of messages that are exchanged between a pair of nodes. This representation together with the vectorial representation of the target node feed a MLP structure, in order the vectorial representation
to be updated. The whole structure is trained by removing randomly an object from the scene and replacing it with an empty node in the graph. The whole point is by repeating the above procedure to predict correct the category and the size of this object. Both category and object are computed by using two other MLPs, which take as input the vectorial representation of the empty node, which is updated at every time step. As loss functions they use a cross entropy loss between the predicted category and the real one, which are represented as one-hot vectors, and a mean square loss between the predicted size and the real one.

In [28] they model the whole network as a scene graph representation. However, in [16] they just use scene graph representations as inputs in their architecture. The difference with the previous section when we saw again scene graphs as inputs, is that here they do not involve probabilistic grammars in their method, and also their scene graphs contain much more information regarding the objects, that assemble the scene. For example they create attributes for the nodes, regarding their height and volume, which are described with words (small, tall, large, etc) and are defined by some self made rules. The spatial relations between objects are also defined by similar rules. These rules are described in the supplementary material of [16].

**Figure 2.4:** Architecture of 3D-SLN. (a) During testing a layout latent vector is sampled and together with the input scene graph are feeding the decoder, in order to generate a new scene layout. During (b) training the input scene graph and ground truth layout are encoded in order scene objects distributions to be learnt. By sampling from these distributions the new latent vectors are sent to the decoder. Figure from [16].
In this approach they try to learn the distribution of scene layouts by using a variational autoencoder. The learnt distribution is conditioned to the input graph and scene layout. This model named 3D-SLN, despite the conditional VAE, utilizes a graph convolutional network (GCN) [11], in order to be able to process the scene graph representations. The encoder essentially learns the conditioned distributions of the encoded objects in the scene (floor, furniture, supporting objects) for which makes the assumption that follow a multivariate Gaussian distribution with a diagonal covariance matrix. By using these distributions, random latent vectors-objects are sampled and together with the input scene graph representation they feed the decoder, which in turn generates new 3D scene layouts. The whole architecture is trained using the Kullback-Leibler divergence loss between objects distributions that are generated by the encoder and corresponding multivariate Gaussian distributions with zero mean vector and identity covariance matrix. Despite that loss, they use another one which is added in the KL divergence loss. This loss is calculated between the input and the generated scene layouts. Actually it calculates a $L_1$ loss between the input and the generated bounding box parameters of scene’s objects and also a rotation loss. In order to calculate the rotation loss they discretize the possible rotation angles in 24 bins and classify the predicted angles of the objects in the corresponding bin. This discretization of the range of rotation angles gives them the chance to calculate a negative log likelihood loss for the classification process of the predicted angles to the 24 bins. This is defined as their rotation loss.

The next approach [27] is an optimization based method that do not utilize neural networks. Moreover, it differs from previous approaches in the way that the scene graphs are constructed. Driven by the fact that scene graphs, which are generated using the co-occurrence between objects in the scene, does not capture meaningful spatial information between them, they introduce a different method. They form a spatial strength graph (SSG), where the nodes correspond to objects in the scene and contain some spatial information about them with respect to their nearest walls. The edges between pairs of nodes contain spatial information between the terminal nodes of each edge, with respect to their beginning nodes. In order to find which spatial relations are more important and ignore those with low spatial strength [27] they use weighted edges in the SSG. These weights are calculated through a complete spatial randomness test (CSR) or specifically through Kolmogorov-Smirnov test. For every pair of objects in the dataset they calculate all the relative positions between them through
all training scenes, and draw them in a planar region as points. For every point they find its two nearest neighbors in the plane and compute the angle between the two vectors that connect the point with its neighbors [1]. Having found this angle for every point they assume that these angles are values of a random variable $\theta$ which follows a uniform distribution. By computing the cumulative distribution function (CDF) and the empirical cumulative distribution function (ECDF) of $\theta$ they use the KS test in order to calculate the weight for a specific pair of objects. If the weight has small value, that indicates that the spatial relation between this pair of objects is not important. Having computed the weights for the edges in the SSG, they use the graph in order to find prior distributions for the positioning pattern of the objects. Sampling from these priors and utilizing again the SSG they are able to generate new scene layouts.

In [15] they implement another graph based approach, by representing scenes as hierarchical graphs. Similar with [27] in [15] they try to interpret the spatial relations between objects in the training scenes in order to generate more reasonable scenes. Their main idea is that a scene structure

![Figure 2.5: A hierarchical graph of a bedroom. Leaf nodes are connected through intermediate nodes, which reveal the spatial relation between them. In the upper levels of the graph objects are forming sub-groups which altogether synthesize the scene layout. Figure from [15].]
follows a hierarchical pattern. As a result they encode that pattern in order to learn a distribution for scene layouts, by training a variational recursive autoencoder (RvNN-VAE). From the objects of every training scene layout they form a hierarchical graph. These hierarchical graphs depict the spatial relationship between objects in the scene, and consist of various nodes. Leaf nodes refer to the objects, while intermediate nodes signify the spatial relation between neighboring nodes. These nodes express support, surround and co-occurrence relations between objects, or even relations between group of objects and their surrounding walls. In the top of the graph there is the root node which corresponds to the whole scene layout. As we can see from Figure 2.5 intermediate nodes connect objects with each other, and form smaller groups of objects that compose the root node. In order this graph to be encoded, they are using six different encoders during the encoding stage of the RvNN-VAE. Each one of them is responsible for a specific category of nodes, and encode information regarding either the objects either their spatial relations. As soon as all the nodes have been encoded they are grouped together through the root encoder. The output of this last encoder constitutes the encoding representation of the scene. This root code is fed into two linear layers $f_\mu$ and $f_\sigma$ in order to determine the predicted mean and covariance matrix of the multivariate Gaussian distribution. Using the reparameterization trick they sample from a normal distribution and synthesize a sampled vector, which is fed into another linear layer $f_d$ in order to obtain the root code which is passed in the de-

Figure 2.6: GRAINS architecture: During the encoding stage the hierarchical graph is encoded through six encoders, each one of them corresponding to a different category of nodes. A sampled root code is then passed to the decoder in order a new hierarchical graph to be generated. Figure from [15].
coder. During the decoding stage six decoders are being used. The root decoder splits the sampled root code, which corresponds to a parent node into smaller codes, which correspond to the child nodes. However, during the decoding stage the initial hierarchy of the scene is unknown, and as a result in order the model to decide which decoder to use for every child code, a node classifier is trained simultaneously with the model, which given a node makes a prediction about its type (leaf, support, surround, co-occurrence, wall) and according to that prediction the corresponding decoder is being used. As soon as all the leaf nodes have been decoded through the box decoder the output hierarchy is formed. Then by using a retrieval process, objects from the dataset with similar parameters as the predicted ones of the leaf nodes are being used. By utilizing the spatial relations that are depicted in the hierarchical graph together with the retrieved objects a new scene layout is generated. The whole model is been trained by minimizing the sum of a reconstruction loss, regarding the parameters of the objects before the encoding stage and after the decoding stage, a Kullback-Leibler divergence loss in order to compare the predicted distribution with one multivariate Gaussian with zero mean vector and identity covariance matrix and finally the node classifier loss.

The most recent graph based approach is [4] in which they utilize a VAE architecture, in order to learn the latent distribution of the input graphs. At the moment the interest of the research community, regarding scene generation, is focused on transformer based approaches, in which we refer in following section. However, in [4] they try to achieve similar performance with the transformer based methods, while simultaneously being able to learn a latent distribution of the data, which other graph based methods are unable to generate. A latent space distribution of the data is useful in applications such as controlled scene generation. Unlike previous graph based VAE architectures in [4] they do not learn just one latent distribution for the input graph, but instead for every furniture described in the graph, a unique latent space distribution is extracted. The model consists of a graph encoder that projects the input graph into a latent space and a graph decoder which produces the generated graph.

2.3 GENERATING IMAGE-BASED SCENES

Graphs based methods, due to their detailed graphs, capture spatial and functional relations between objects in training scenes, and as a result generated scenes follow also these patterns. However, there is no dataset
that contains such graphs, so this work has to be done manually from the researcher. Moreover all these relations are determined and prioritized from every researcher based on his personal criteria, regarding which of them are more or less important. As a result by utilizing these methods there is no guarantee that all the necessary relations are captured and learnt, while simultaneously their complexity remains at high levels. Image based scene generation comes to alleviate this problem, by using top-down images of training scenes that feed deep convolutional networks. Most of the times these images are provided by the datasets, but if not the procedure of generating them take less time than constructing graphs with specific rules and relation patterns.

In [24] they use orthographic projections of indoor scenes as inputs in order to learn priors distributions for the objects in the scene. Essentially, for every top-down view of a scene they create a multi channel array. The first channel is the top-down view of the scene. The next five channels are obtained by applying specific masks to the first one. These masked outputs give information about the space demarcation of the scene, the position of walls and doors, the number of objects in the scene and finally their orientation. For every type of room there are specific categories of objects, defined from the dataset that can be used. By using this information they create hot label vectors for every category and then apply them to the corresponding pixels of the scene. By concatenating them with the previous six channels they obtain for every top down image a multi channel matrix.

Their architecture is divided into three parts. The first one is responsible to decide if given a scene with already placed objects, the insertion of new objects must be continued or terminated. This decision is implemented by computing the probability of continuing the object insertion procedure conditioned with the scene. This probability is calculated through a deep convolutional network which is fed with the corresponding multi channel matrix of the scene. The output of this CNN is concatenated with a vector, which depicts how many objects of the possible categories exist in the scene. This concatenation is fed into a MLP and then into a sigmoid function. If the probability is greater than 0.5 the completion of the scene goes on. By using the multi channel matrix of the scene in combination with the number of objects per category, they try to recognize successfully if a scene is saturated from objects. In order to train this first part they use the top-down views of their dataset, which are divided into two random equal parts. One part is kept unchanged and from the other one they remove objects randomly.
The second part of their architecture implements the predictions, regarding the category and the location of an object inside a given scene. Essentially, it computes a probability distribution over all object categories, conditioned in a given scene and the possible locations in its 2D layout. This architecture is implemented with a CNN combined with a MLP and predicts one object at a time. In order to limit placement of objects in already occupied locations or generally in no optimal locations, they use three more object categories for their predictions, which do not correspond to any object, but just facilitate the prediction process. With these categories they manage to bind some locations in the scene as "remain unoccupied", "already occupied" and "outside the scene". By training this architecture with these auxiliary categories their predictions becoming more accurate. This happens because they use a much higher percentage of scenes labeled with these auxiliary categories for prediction, than the scenes that are labeled with an object category. After they have obtained the desired probability distribution they sample from it in order to specify the object category and its location. Given the specified object category they choose from their dataset a specific object. The SUNCG 3D dataset [26], which they use in their method, separates its objects not only per category but additionally it contains for every category collections of objects. Therefore, an object is retrieved firstly from the predicted category and then from the same collections that already objects in the scene belong to.

The third part of this architecture is also a CNN network combined with a MLP, and predicts the orientation which must be applied in the retrieved object. The training process both in the second and third architecture is implemented as in the first part. They remove objects from the scene and try to predict its category and orientation.

Similar with [24] in [21] they use independent decision modules in order to predict the desired parameters of the objects. However, there are differences in the number of these modules and also in their implementation. In this approach, given a top-down view of a partial or empty scene, the model predicts the next object to insert. The top down image is first encoded through a Resnet18 [9] and the encoded output is concatenated with a vector that indicates the number of objects per category in the scene, which is also encoded through a FCN [21]. The concatenation is then fed to the first decision module in order to learn a probability distribution over all possible categories. The next module, by using the encoded representation of the input top-down image combined with the predicted category, predicts the location of the object. This module generates a 2D probability distribution.
over the scene layout in order to predict the location of the object. The next decision module is responsible for the orientation of the object in the scene. As before this module must be fed with the top-down input image and the previous predictions. However, in order this input information to be more compact, instead of just passing the encoded top-down view and the encoded predictions, they translate the input image in order to be centered around the predicted location. As a result the orientation module takes only two inputs instead of three and also the following modules become translation invariant [21]. The orientation decision module learns a probability distribution over the possible orientations by utilizing a convolutional variational autoencoder (CVAE). Before moving to the next module they rotate the whole scene according to the predicted rotation angle, in order the final dimension decision module to be rotation invariant. As before, by utilizing this technique, they achieve to inform the subsequent modules about the previous predictions, just by feeding the input scene, in which they have applied the previously predicted translation and orientation. The final module is responsible for the dimensions of the bounding boxes, which contain the objects. By utilizing a CVAE, this module generates a distribution over the dimensions of the bounding boxes and is able to predict their dimensions. In order to complete a scene the model uses the predicted parameters and through a retrieval process finds the best fit from the objects dataset. In order to train their model they are using scenes from the dataset from which they have first subtracted some objects. This subtraction procedure however is not implemented randomly, as they assume that the objects in the scene follow some ordering. According with supporting relationships between them and with their occurrence frequency across the dataset, they first order the objects in a scene. Having applied that ordering they subtract some or all the objects. This technique however has some important limitations which are discussed more extensively in subsequent sections.

2.4 Generating Scenes Using Transformer Architectures

In this section we refer to approaches which utilize the architecture of Transformers. Transformer architecture was introduced by [23] and first used in NLP problems with great success. As sentences can be imagined as sequences of words, scenes can be imagined as sequences of objects too. With this fact in mind, some state of the art approaches that were introduced are utilizing transformer architectures in their models, in order
to exploit its great performance in sequence problems. In [25] they treat scenes as sequence of objects, and given one partial or empty scene they predict autoregressively all the extra objects, in order to populate the scene. Particularly they predict the parameters of one object per time, conditioned on the floor layout of the scene and the already existed or predicted objects. The prediction of the parameters follows the order class, orientation, location and dimension. They also have trained a model in which the conditioning is upon a text description of the scene. Similar with [21] they order the objects in training scenes according to the frequency appearance of their objects in all scenes of the dataset. The meaning of this ordering is that more important objects, which always exist in particular scenes, such as beds in the case of bedrooms, are placed in the beginning of the sequence, so the network predict them at first place. For each one of the predicted parameters they train a separate network. Every network is a transformer decoder which is optimized using cross entropy loss. The output of each network is given as input to the next one. Furthermore the category, orientation and location model are not conditioned in the dimensions of the already predicted objects as we can see in Figure 2.7.

**Figure 2.7:** SceneFormer architecture conditioned in a room layout. Four transformer decoders are trained separately in order to predict the parameters of the object. Each one of them is conditioned in the already predicted objects and the new prediction of the previous transformer decoder. The green color squares refer to the start tokens that are feeding each decoder, the gray refer to the alrady predicted parameters and the yellow ones refer to the predictions from the previous transformer decoder model. Figure from [25].
The input sequences are projected to learned embeddings in order to feed each model. Particularly they use position and value embeddings. The position embeddings describe the position of the objects in the sequence and the value ones the parameters of each object. In order to distinguish the location and the dimension parameters, in the case of the dimension model, they use another embedding layer in order to indicate which value embeddings refer to locations and which to dimensions. They have adapted this extra embedding layer and in the case of the location model, despite that it is not conditioned in any dimensions. During inference, in order to sample from the extracted multinomial distribution in the output of each transformer decoder, they use nucleus sampling in the case of the category model, while in the next models they just keep the value with the highest probability. Having predicted the parameters of the objects, the model then retrieves them through the dataset by using its predicted dimensions. These dimensions are compared with the dimensions of dataset objects by calculating the mean square error, between their dimensions. In the case that the retrieved object violates the predicted location bounds in a certain degree, they retrieve the object with the next smallest mean square error, and this process continues 20 times until a retrieved object fits properly in the predicted location. If no object satisfies this restriction, they sample again from the category distribution.

While SceneFormer was the first approach that introduced the usage of Transformers in the scene synthesis task, in fact the whole architecture lacks at some points. First of all the ordering of the sequences of objects, which was adopted from [21] is a trick that facilitates the training, but limits the successful generative process of both models only in specific scenarios. For example both approaches could never populate a bedroom with a bed, if the partial scene in which the model is conditioned contains only objects that have smaller frequency of appearance in the dataset than the bed. This happens because the model learns only to populate partial or empty scenes by following this ordering. Particularly, the training process is implemented by subtracting randomly some of the ordered objects in the scene and using the remaining in order to predict the next in the ordered sequence. As a result these models in order to predict successfully an object, they must be conditioned only with objects with greater occurrence frequency. Furthermore, the sampling process in [25] regarding the orientation, location and dimension models can not be considered sampling, as they just pick the value with the highest probability for each predicted probability distribution of the corresponding transformer decoder. Finally the retrieval
process of sceneFormer which repeats the process until the retrieved object fits appropriately in the predicted location and samples again from the category distribution a suitable match can not be found, violates the meaning of a generative model.

The next approach in which is based the current Master thesis is about ATISS [18], an approach which filled the gaps of SceneFormer by introducing a method that outperformed the previous approaches. In [18] they also assume that a scene is a sequence of objects. Given an empty or partial scene they predict the parameters of one object at a time, until the population of the scene with objects is complete. Oppositely with [21] and [25], in [18] they do not apply any ordering in the objects of training scenes according with their frequency of appearance in the dataset. Instead they apply a random permutation in the objects, in order every sequence with any ordering to have the same probability to be generated by the model. In ATISS, similar with previous approaches, they describe objects in a scene as bounding boxes defined by four parameters, category, translation, orientation and size. For each one of these parameters they use specific distributions in order to describe them. Particularly, the model tries to learn the parameters of these distributions during training, in order during inference to be able to sample from them and by using a retrieval process to populate a scene.

Figure 2.8: ATISS architecture during inference: The room layout and the objects of the partial scene are encoded respectively through layout and structure encoder. These encoded outputs together with a random initialized query vector $q$, which corresponds to the next object are concatenated in order to form the sequence of objects in the scene. This sequence is passed to the transformer encoder. By using the output query vector $\hat{q}$ of the transformer encoder the parameters of the new object in the scene are predicted autoregressively. Figure from [18].

The model is composed by three parts which are trained simultaneously by a single loss, by utilizing the teacher forcing technique. The first part
encodes the room layout and the objects of the partial scene. These encoded representations are concatenated together with a random initialized query vector, which is also learnable, in order to form the sequence which is passed to a transformer encoder. The output query vector of the transformer encoder is used in order to calculate autoregressively the parameters of the desired distributions. This autoregressive prediction process includes equal steps with the parameters of the object. Each step corresponds to one parameter. During training, this autoregressive process is conditioned at each step to the query vector and the ground truth encoded values that correspond to the predicted parameters of previous steps. During inference, it is conditioned to the query vector and the sampled parameters of the previous distributions, after they have been encoded. As soon as the parameters for each distribution have been calculated, the model computes the negative log likelihood of the target value to have been generated for the just computed distribution. By minimizing this log likelihood the whole architecture is being trained.

Driven by ATISS and SceneFormer in [14] they present a new approach that utilizes ideas from both of them while proposing a new method in order the generative process taking into account some ergonomic rules. Reliable datasets are very important for all research projects and many times they define and facilitate the quality of the conducted research. Due to the lack of well defined datasets and the high cost of creating high quality ones, existing models tend to produce artifacts during inference, which appear also in the training scenes and as a result models have learnt them. Such artifacts are overlaps between objects in the scene, inconsistency regarding their size, placement of objects in no operational positions and finally usage of objects that do not match with the scene. Generally, these artifacts correspond to scenes that never exist in real life and have occurred, due to the rough dedication during their creation by their creators. In order to deal with this problem in [14] they introduce an approach which use a transformer combined with some ergonomic rules. Particularly they define some ergonomic rules which are described in the form of loss functions, in order to penalize scenes that do not following them. As a result they reduce the probability such scenes being generated by their generative model. These rules are relevant with the way one room is populated by objects in real life, in order to be fully operational by its users during various activities. These ergonomic losses are combined with the transformer loss through a weighted sum. The whole model is closer to SceneFormer than ATISS. First of all, it has adopted the ordering sequences of objects during training that
SceneFormer also uses. Moreover, the sampling and the retrieval process are implemented with the same way as in [25].

### 2.5 Generating Scenes Using Radiance Fields

A revolution to 3D representations took place with the introduction of NeRF [17], a neural network that given 3D points and viewing directions can calculate the radiance fields in the image plane that is formed between these points and the corresponding cameras. Particularly rays are defined between the 3D points and cameras and upon these rays extra points are sampled. For these points NeRF calculates their optical density and radiance and through the rendering equation [12] computes the radiance in the corresponding pixels upon the image plane. Using this technique approaches of this section try to build generative models, which are trained in multiple scenes. Unlike the above mentioned approaches that represent scenes as graphs, or top-down images or as a collection of CAD models, these approaches describe scenes as a trajectory of images, where for every image is known the parameters of its corresponding camera. As every image defines a radiance field, a trajectory of images defines a group of radiance fields. The problem at this point is to find a way to build a generative model in order to learn a distribution for these radiance fields, so that after training, being able to sample from this distribution and generate images of new scenes and by combining them extracted a new trajectory - scene.

The first approach known as GSN (Generative Scene Networks) [6] follows a GAN architecture enriched with a radiance field network, similar with this in [17]. Particularly, they generate a random vector and feed their generator in order to produce a 2D-grid latent representation, which contains information for all images in the scene - trajectory. For every image in trajectory they cast rays from the corresponding camera towards every pixel of the image, and upon these rays they sample particular points. From all the rays that are casted from the camera, the sampled points, are projected into the 2D-latent representation produced by the generator and also transformed from the global coordinate system into the camera coordinate system, in order to reduce overfit effects during training. The projections together with the transformed points feed the radiance field network in order to calculate the radiances and the optical densities of the points and through volumetric rendering to extract the new image. The camera pose from which the rays are casted is obtained by a weighted sampling from an empirical distribution. From every pose in the trajectory they cast rays
2.5 Generating Scenes Using Radiance Fields

and upon these rays they sample points. By passing these points through their radiance field network they obtain from every sampled point its occupancy. The computed occupancies after they have been calculated are normalized in order to form a multinomial probability distribution, from which they sample valid camera poses. The normalization is applied by using a softmin function, in order points with high occupancies close to one after the normalization to obtain values-probabilities close to zero and as result to reduce or almost eliminate their possibility to be sampled from the multinomial distribution. A point with occupancy close to one means that is close to a surface and therefore is an invalid candidate camera pose. The whole architecture is trained by using a GAN loss combined with a reconstruction loss, in order the radiance field network to be optimized too.

Figure 2.9: GSN generator architecture: A random vector $z$ sampled from a distribution is passed to the generator, which produces a 2D latent grid $W$. A camera pose $T$ is sampled too and is casting rays towards all pixels of the target image. Upon the rays, points are sampled and then projected to the 2D latent grid $W$. The projection $w_{ij}$ together with the corresponding points which are firstly transformed into the camera coordinate system are passed into the radiance field network, which computes the radiance $\sigma$ and the optical density $a$ for these points. By using volumetric rendering they render a new image. Figure from [6].

Even if in [6] they set a new scene generation pipeline, the fact that they have adopted a GAN architecture entails that problems which GANs tend to perform, such as mode collapse, may arise also in this model. Having this in mind in [2] they follow a different approach in order to avoid using GANs. As in [6] they utilize trajectories of images that depict scenes, and for every image its camera parameters are known. The first step that is implemented in [2] is matching every trajectory with two latent representations. The first one refers to the images of the trajectory and contains information about their content, while the second one represents the camera poses of
the images. In order to obtain these representations they train separately for every trajectory an autodecoder network according with [3]. Having extracted these latent representations they utilize a decoder architecture, which consists of three parts. The first two parts are two decoders that process the two latent representations of every trajectory. The third and final part is one radiance field network. Like in GSN the whole goal here is to condition the radiance field network with a latent parameter in order to be able to control the content that is going to be rendered.

Figure 2.10: GAUDI decoder consists of three parts. The first part is one latent scene representation decoder which produces three 2D latent grids. The second is one latent pose decoder which extracts one camera pose given the latent pose, which will be used for the ray tracing. As soon as the rays have casted from this camera pose towards the pixels of the target image, points are sampled upon them which then projected into the three 2D latent grids. These three projections are then concatenated in order to create for every point upon the ray a conditioning in the final part of GAUDI decoder, which is a radiance field network. Figure from [2].

Their approach is inspired by [6]. However, their latent scene decoder instead of producing one latent 2D grid as the GAN generator in [6], creating three latent 2D grids. Alongside with that the pose decoder by taking the latent pose representation produces a camera pose which is used for the ray casting in the radiance field network. Points that are sampled upon the rays are projected into the three latent 2D grids, and the three projections are being concatenated into a single one latent vector, which feeds the radiance field network. This decoder architecture is trained by using all the different trajectories, by minimizing the reconstruction loss between the extracted camera pose and the real one, and the reconstruction loss between the final rendered and the target image. At this point in order to complete their generative model they learn a distribution over the latent image representations in order to be able to sample from this and through
their decoder to render new images and new trajectories. Particularly, they learn a Denoising Diffusion Probabilistic model according to [10].
In this chapter we present our method and give a step by step explanation of our approach. First of all, we define the loss function which we use in our model and then give a clear explanation of how we apply it in our architecture.

### 3.1 Loss Function Definition

Our goal is to build a generative model capable of generating new scenes while taking into consideration also style consistency across the objects of the scene. We follow ATISS [18] approach for this problem, while trying to extent ATISS capabilities in order to incorporate also style prediction. We describe objects in the scene with five random variables, describing the corresponding class, translation, orientation, size and style. Our model must learn during training, distributions that describe these five parameters, in order during inference to be able to sample from them and generate new scenes. Assuming, that these five distributions are described by a single distribution $p_{\theta}$ with unknown parameters $\theta$, given a set of training scenes we could use Maximum Likelihood Estimation, in order to optimize $\theta$. Specifically we want to optimize the parameters under which, the likelihood that these training scenes have been generated from this distribution is maximized. Having this in mind, we create an architecture that computes this distribution $p_{\theta}$ by trying to maximize the log likelihood of generating the training scenes, or equivalently to minimize the negative log likelihood of generating these scenes. The first step to achieve our goal, is to find a way to describe this likelihood.

Unlike with [21] and [25] that assume scenes as sequences of ordered objects, following [18], we assume that objects in a scene can have any order, and as a result we want to describe the likelihood of generating our training scenes under any order. Let us assume that we have a collection of scenes $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$. This collection is referred to a specific type of indoor scenes (e.g. bedrooms, living rooms). Every scene $\mathcal{X}_i = \{\mathcal{O}_i, \mathcal{F}_i\}$ consists of its objects $\mathcal{O}_i$ and a floor layout $\mathcal{F}_i$. The objects of the scene $\mathcal{O}_i = \{o_{ij}^{i}\}_{j=1}^{M_i}$
is a set containing its furniture and as we mention they are described by five parameters, which follow their own probability distribution. As a result, we assume that set \( \mathcal{X}, \{ \mathcal{X}_1, \ldots, \mathcal{X}_N \} \), \( O_i, \{ o^i_j \}_{j=1}^{M_i} \) are random vectors and the five parameters that describe every object \( o^i_j \) are random variables. Specifically these five parameters are class, translation, size, orientation and style and give us a clear explanation about the identity of the object, its exact position in the scene and a representative descriptor about its style.

Assuming that we have only one scene \( \mathcal{X}_i \) and the furniture of this scene can be ordered according to only one permutation \( j \), we can compute the likelihood of generating this particular scene by computing the likelihood of generating autoregressively its objects \( O_i = \{ o^i_j \}_{j=1}^{M_i} \) and the floor layout \( F^i \). Using the chain rule we can define this likelihood as:

\[
L(\theta, \mathcal{X}_i) = p_\theta(\mathcal{X}_i) = p_\theta(O_i, F^i) = p_\theta(o^1_i, o^2_i, \ldots, o^i_{M_i-1}, o^i_{M_i}, F^i) = p_\theta(o^1_i / o^j_i, F^i) \cdots p_\theta(o^i_{M_i-1} / o^i_{M_i}, F^i) p_\theta(o^i_{M_i} / F^i) p_\theta(F^i) \tag{3.1}
\]

We assume that the possible floor layouts \( F^i \) have equal probability across the dataset and follow a uniform distribution. Thus we can ignore the term \( p_\theta(F^i) \) and the likelihood of Eq. (3.1) becomes:

\[
L(\theta, \mathcal{X}_i) = p_\theta(o^1_i / o^2_i, \ldots, o^i_{M_i}, F^i) \cdots p_\theta(o^i_{M_i-1} / o^i_{M_i}, F^i) p_\theta(o^i_{M_i} / F^i) = \prod_{j=1}^{M_i} p_\theta(o^i_j / o^i_{j+1}, F^i) \tag{3.2}
\]

where \( o^i_{j+1} \) denotes the chain of random vectors \( o^i_{j+1}, o^i_{j+2}, \ldots, o^i_{M_i} \).

Now, in order to calculate the likelihood of generating scene \( \mathcal{X}_i \) in every possible ordering we just have to calculate the likelihood for every ordering using Eq. (3.2) and sum them all together, i.e: Eq. (3.3). Note that we can sum them because the orderings cannot happen at the same time, they are mutually exclusive events and therefore, the whole likelihood of the scene \( \mathcal{X}_i \) is given by Eq. (3.3):

\[
L(\theta, \mathcal{X}_i) = \sum_{u=1}^{K_i} \prod_{j_u=1}^{M_i} p_\theta(o^i_{j_u} / o^i_{j_u+1}, F^i) \tag{3.3}
\]

with \( K_i \) we refer to the permutations that can occurred using the \( M_i \) objects of the scene \( X_i \), which are \( M_i! \) in number.
In order to calculate the likelihood of the whole collection $\mathcal{X}$, we calculate the likelihood of generating every scene $\mathcal{X}_i$ using Eq. (3.3). Assuming that scenes are statistically independent, we can multiply all these likelihoods. Therefore, we can write for the likelihood of the whole collection $\mathcal{X}$:

$$L(\theta, \mathcal{X}) = \prod_{i=1}^{N} \sum_{u=1}^{K_i} \prod_{j_u=1}^{M_i} p_\theta(o_{j_u}^i / o_{>j_u}^i, F^i_i)$$

(3.4)

and its log likelihood is given by:

$$\log(L(\theta, \mathcal{X})) = \sum_{i=1}^{N} \log \left( \sum_{u=1}^{K_i} \prod_{j_u=1}^{M_i} p_\theta(o_{j_u}^i / o_{>j_u}^i, F^i_i) \right)$$

(3.5)

The task we have now is to find the parameters $\theta$ that maximize Eq. (3.5). However, by observing (3.5) we see that the form in which we have concluded includes logarithms of sums, which are intractable and can not be further simplified. However, we exploit a well known inequality that connects the arithmetic and geometric mean of $N$ non negative numbers $\{x_1, x_2, \ldots, x_N\}$, which is known as AM-GM inequality and states that:

$$\frac{x_1 + x_2 + \ldots + x_N}{N} \geq \sqrt[N]{x_1 x_2 \ldots x_N}$$

(3.6)

Note that the equality holds when $x_1 = x_2 = \cdots = x_N$

Since the likelihood of every permutation of a scene is a non negative number the above mentioned inequality can be used in Eq. (3.5) giving a lower bound of this equation. This lower bound is described by the following inequality:

$$\log(L(\theta, \mathcal{X})) = \sum_{i=1}^{N} \log \left( \sum_{u=1}^{K_i} \prod_{j_u=1}^{M_i} p_\theta(o_{j_u}^i / o_{>j_u}^i, F^i_i) \right) \geq \sum_{i=1}^{N} \log \left( \frac{K_i}{\prod_{u=1}^{K_i} \prod_{j_u=1}^{M_i} p_\theta(o_{j_u}^i / o_{>j_u}^i, F^i_i)} \right) = L(\theta, \mathcal{X})$$

(3.7)

We can now maximize this lower bound, denoted by $\underline{L}(\theta, \mathcal{X})$, which can be rewritten in the following simpler form as:
\[
\mathcal{L}(\theta, \mathcal{X}) = \sum_{i=1}^{N} \log(K_i) + \sum_{i=1}^{N} \left( \frac{1}{K_i} \log \left( \prod_{u=1}^{K_i} \prod_{j_u=1}^{M_i} p_\theta(o_{j_u}^i / o_{j_u}^i, \mathbf{F}^i) \right) \right) = \\
\sum_{i=1}^{N} \log(K_i) + \sum_{i=1}^{N} \left( \frac{1}{K_i} \left( \sum_{u=1}^{K_i} \sum_{j_u=1}^{M_i} \log(p_\theta(o_{j_u}^i / o_{j_u}^i, \mathbf{F}^i)) \right) \right)
\]

It is clear from Eq. (3.8) that the first term is constant and does not depend on the parameters \( \theta \). Therefore, we can omit it and just maximize the following function:

\[
\mathcal{L}_2(\theta, \mathcal{X}) = \sum_{i=1}^{N} \left( \frac{1}{K_i} \left( \sum_{u=1}^{K_i} \sum_{j_u=1}^{M_i} \log(p_\theta(o_{j_u}^i / o_{j_u}^i, \mathbf{F}^i)) \right) \right)
\]

The function in Eq. (3.9) is the sum of the mean values of the likelihoods from all the possible permutations for every scene. This is something much desirable in our problem because we want scenes with more or less furniture-objects, which will have more and less possible permutations respectively, to have the same chance to be generated by our predicted distribution. That is why the mean value in Eq. (3.9) facilitates this cause. However, using directly (3.9) in order to optimize our architecture is not computational efficient, because the total number of some permutations is very large and that will slow down the training process. Moreover, due to the fact that all scenes do not have the same number of possible permutations we can not organize our training data in batches. As a result we would have to keep the batch size equal to one and use stochastic gradient descent during training. For that reason we use Monte Carlo sampling with one sample by using for every scene \( \mathcal{X}_i \) a uniform distribution \( U[1, K_i] \), in order to choose randomly with same probability one of its possible \( K_i \) permutations. As a result according with that admission instead of the function described in Eq. (3.9) we maximize during training the following function:

\[
\mathcal{L}_3(\theta, \mathcal{X}) = \sum_{i=1}^{N} \left( \sum_{j_u=1}^{M_i} \log(p_\theta(o_{j_u}^i / o_{j_u}^i, \mathbf{F}^i)) \right)
\]
In order to be able to use term (3.10) as loss function in our architecture during training, we have maintained ATISS structure. At every epoch given a scene $\mathcal{X}_i$ with objects $\mathcal{O}_i = \{o^i_j\}_{j=1}^{M_i}$ and a floor layout $\mathbf{F}_i$ we sample from a uniform distribution one of the possible permutations $K_i$. According with this permutation we order the objects in the scene and form a sequence. From this sequence we keep a random number of objects starting from the beginning, which will be processed from the network, and we label as target the first object after the sequence that we kept. The task of our network is to calculate the parameters of the distribution $p_\theta$ so that the negative log likelihood of the target object having being generated from the estimated distribution to be minimized. Particularly our model predicts the distribution $p_\theta$ for the next object to add in the partial scene. As we mentioned in the beginning of the chapter every object is described in our approach by five parameters and thus for every parameter we define one distribution. At this point however we must mention one extension that we made in term described in Eq. (3.10). We do not want only to predict autoregressively the next object in the scene by using the objects of the partial scene. We want also the distributions of the next object to be calculated autoregressively, meaning that in order to calculate one distribution we use the corresponding target parameter of the just previous predicted distribution. Therefore, we must define an order under which our model computes the distribution of each parameter. This order is not unique but instead is set by the researcher according to the way he prioritizes the parameters. In ATISS they predict first the distribution of the class, then the distribution of the translation, then that of the angle and finally the distribution of the size. This order makes sense, as in a real scene completion scenario someone might follow similar steps in order to populate a scene with objects. The first decision would be upon what type of object to use, followed by where to place it, towards where to be oriented and finally what dimensions that specific object might have in order to fit in the candidate location. As a result in our approach we keep this order and add an extra distribution in the end in order to predict the style of the next object to be added in the scene. Specifically under this convention we can rewrite our loss function, according to the chain rule in the following way:
\[-\frac{1}{N} \sum_{i=1}^{N} \left( \sum_{j_u=1}^{M_i} \log \left( p_{\theta_c}(c^i_{j_u} / o^i_{>j_u}, F^i) \right) \right) \]

\[
p_{\theta_t}(t^i_{j_u} / c^i_{j_u}, o^i_{>j_u}, F^i) p_{\theta_a}(a^i_{j_u} / c^i_{j_u}, t^i_{j_u}, o^i_{>j_u}, F^i) \]

\[
p_{\theta_s}(s^i_{j_u} / c^i_{j_u}, t^i_{j_u}, a^i_{j_u}, o^i_{>j_u}, F^i) p_{\theta_z}(z^i_{j_u} / c^i_{j_u}, t^i_{j_u}, a^i_{j_u}, s^i_{j_u}, o^i_{>j_u}, F^i) \]

with \(c^i_{j_u}, t^i_{j_u}, a^i_{j_u}, s^i_{j_u}, z^i_{j_u}\) we denote respectively:

- class
- translation
- orientation
- size
- style

of the target object of the partial scene that our model have processed. This strategy of using during training the target values in order to estimate some of our predictions called teacher forcing and is a common one in autoregressive models such as ours. During inference, where we want to generate new scenes, we use in each step the predicted values of the previous steps, in order to calculate the five distributions and then sample from them. Actually every object is parameterized by nine parameters and not by five. This happens because for the size and the translation we predict three different distributions, one for every direction and dimension in the case of translation and size respectively. So we refer to five distributions just for brevity.

In Figure 3.1 we present the proposed architecture of our model both during the training and inference phase. During training, given a scene with its floor layout and its objects, the first step is to encode both the floor layout and the parameters of the objects. For the floor layout we extract firstly its mask and then we use a ResNet-18 [9] in order to encode the masked layout and create a latent vector of length 512. This ResNet-18 is trained together with the whole architecture. The partial scene we extract is encoded by the attribute encoder. Particularly every attribute of each furniture (class, angle, style, etc) is projected in a latent space with dimension 64. For the category
and style we use linear layers which are optimized during training. For the angle, size and translation we use positional encoding [23] in order to create the corresponding latent vectors. This projection oppositely with the case of class and style is fixed during the whole training. In the following equation we describe the positional encoding projection we apply to these values, where with $v$ we denote each time the values for the angle, the translation across any direction and the size across any dimension. The factor $m$ is half the dimension of the latent space, so $m = 32$ and $j = 0, \ldots, m - 1$. The scale $\sigma$ was set equal to 10.

$$\gamma(v) = \left[ \ldots, \sin\left(\frac{2\pi \sigma j}{m} v\right), \cos\left(\frac{2\pi \sigma j}{m} v\right), \ldots \right]$$  \hspace{1cm} (3.12)

By combining all these latent vectors through concatenation and linear projection we form a latent vector for the whole partial scene which also has length 512. Finally we create a random vector of length 512 which is randomly initialized and corresponds to the object, with which we want to complete the scene. This vector is learnable and is being optimized together with the parameters of our network. In Figure 3.1 we denote this randomly initialized vector as $q$. The idea is to create a whole latent representation for the whole scene, which will include also the missing object, which is initialized randomly. The reason under this choice is due to the fact that we want to utilize a specific architecture in order to process this latent scene. Having extracted latent representations for the scene layout, the objects of the partial scene and the missing object, we have with some way to process all this information as a whole entity. We want to find relations between all the furniture and the layout. The ideal way to achieve this is by utilizing as in [18] a transformer encoder. We are using a transformer encoder with six layers and the multi head attention in each layer has six heads. We concatenate the latent representations of the room layout, the partial scene and the missing object and create the input to the transformer. We assume that the latent vector, which corresponds to the layout is the start token of the given sequence. Unlike the NLP problems, where the words in a sentence have specific order, in our case we want our model to be able to process the partial scenes - sequences without depending from the ordering of the objects in it. This was also an assumption we made in order to define the loss function. This is another reason why the transformer architecture in the way we utilize it is an ideal choice. By not using positional encoding layers in order to encode the input sequence, as it is done in NLP problems, we are making our transformer encoder permutation invariant and as a
result we can process the input sequence without concerns about the order of objects in the sequence. The transformation architecture apart from the

encoder normally contains also a decoder part. The decoder part extracts a categorical probability distribution over a discrete set of data. A decoder approach however limits our choices regarding the distributions with which we can model objects parameters. That is why in [25] where they follow a such approach they model all the parameters with categorical distributions and optimized them with cross entropy losses. On the contrary we use only

Figure 3.1: The proposed architecture of our model (a) during training and (b) during inference.
a transformer encoder. Transformer encoder takes a sequence as input and due to its multi head attention layers, its residual connections and its layer normalization layers returns a new sequence of the same dimension, which although contains information about the relations of its parts-objects. From this new sequence we keep only the corresponding latent vector of the missing object and by using it we predict autoregressively the distributions of its parameters. As we can see in Figure 3.1 first of all we predict the distribution of the object category by using \( \hat{p} \) latent vector. For every next distribution we utilize apart from \( \hat{p} \) the target parameters of the missing object, which correspond to the previously predicted distributions. More specifically we do not use them in their original form but instead we encode them by using a target encoder. These encoded latent vectors are concatenated at each step with \( \hat{p} \) as we see in Figure 3.1. During inference given a partial or totally empty scene we follow a similar approach as we can see in Figure 3.1 (b). We predict one object at each step until the network predicts the end token class and terminates the scene completion procedure. As end token class we have defined one extra hot vector additional with the hot vectors of objects categories. Having permuted the training sequences before, we add the end token in the end of all of them. By doing that the network learns also to predict the end token and as a result during inference terminates automatically the population of the scene. Moreover, we create also a start token class in the case our scene is totally empty, in order to have a representation for that condition and be able to feed our encoder. In that case the rest parameters are initialized with zero vectors. By using this start token class our network learns to recognize that empty scene condition, and during inference when we give as input an empty scene it fills it with objects without any problem. Regarding the order of prediction we believe that except form the category distribution which must be placed first in the order of predictions at any case, the order of the other parameters might vary, according with the case in which we apply our model. For that reason we concluded a permutation study for the rest parameters which is presented analytically in the next chapter.

3.3 EXTRACTING OBJECT PARAMETERS

Before we move in the description of the distributions for the parameters of objects, it is very useful to present objects parameters and particularly how they were extracted and processed in order to form our training dataset. In this project we use 3D FRONT [7] dataset, which is a collection of indoor
3.3 Extracting Object Parameters

Scenes and 3D FUTURE dataset [8], which contains information about 3D objects that populate 3D FRONT scenes. Particularly, 3D FRONT provides us with information regarding:

- size
- position
- orientation

of all objects inside a scene. All these information concern the meshes of the objects and particularly the bounding boxes that contain these meshes. On the other hand 3D FUTURE contains information about the category of the objects, their style and also for every object it includes its mesh, a texture image and finally an image of the object itself. We will analyze all the parameters one by one and the approach we follow in order to be able to utilize them during training and inference.

3.3.1 Class

In this Master thesis we got involved with bedrooms and living room of the 3D FRONT dataset. Each type of room has each own categories of objects. We use for our experiments 5892 bedrooms and 2925 living rooms. In the bedrooms we have 21 categories of objects while in living rooms we have 24. In Figure 3.2 we give bar charts, where we can see the categories of objects for every type of room and moreover their occurrence frequency across all training rooms we use. We represent these categories as hot label vectors. Particularly, we sort these categories by their name and create an identity matrix $23 \times 23$ in the case of bedrooms, and $26 \times 26$ in the case of living rooms. The reason why these hot label matrices are not $21 \times 21$ and $24 \times 24$ respectively, is because we assume two extra categories the start and end category, which we analyzed previously. The start and end category are the penultimate and last hot labels respectively in the hot label matrix.

3.3.2 Size

For every object, 3D FRONT dataset gives us a scale vector. The mesh of each object is in a different scale, so in order all of them to be in a common scale we applied this scale vector to object mesh. We define the size of the object as the size across every dimension of the scaled bounding box.
Figure 3.2: The occurrence frequency of all objects for (a) bedrooms and (b) living rooms.
3.3.3 Orientation

Each object of the 3D FRONT dataset has a particular orientation in a scene. For every scene the dataset provides us with unit vectors for all its objects. By taking the cross product between these unit vectors and the unit vertical axis we obtain the axis according which we must rotate the meshes of the objects. In order to find the rotation angle we are taking the inverse cosine of the dot product between the rotation vectors and the vertical axis. Having obtained the rotation vectors we transform them to unit ones and then in combination with the rotation angles we apply a rotation transformation to the vertices of each object mesh according to the following Eq. (3.13). With $u_x, u_y, u_z$ and $\theta$ we denote the unit rotation vector and the rotation angle respectively.

$$R = \begin{bmatrix}
\cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\
u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\
u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta)
\end{bmatrix} \quad (3.13)$$

3.3.4 Translation

As translation we define the coordinates of the centroid of the bounding box that contains each object mesh. For every scene 3D FRONT dataset provide us the translation vectors which we must apply to its objects meshes in order to place them at the correct location inside the scene.

Having defined the translations of the objects of each scene in our training set, we apply a translation transformation per scene in order both the scene layout and the objects to be centered around the origin of axis. Particularly we subtract from the vertices of the already scaled, rotated and translated meshes of the objects the centroid of their corresponding scene mesh and we apply the same subtraction to the vertices of the scene mesh. Having centered all training scenes around the origin axis we transform their translation, size and angle parameters in order their values to be inside the range $[-1, 1]$. During inference we apply an inverse transformation in order to obtain the corresponding values in the previous ranges.
3.3.5 Style

Regarding style we follow various approaches. According to the first one we use the style labels that are given by 3D FUTURE dataset for each object. In Figure 3.3 we present the possible style labels that are observed both for bedrooms and living rooms together with their occurrence frequency across all training scenes. We assume that for each style label we have a style category and for each category we create a hot label vector. Particularly we follow the same approach just like in the case of the class category. Apart from this we follow three other approaches in order to define the style. In all off them we extract for every object a style feature of length 512. The difference between these three approaches is the way that this style feature extraction is implemented. As 3D FUTURE dataset provides images of all the objects, we train an autoencoder architecture given these images, in order after training to project all the images in its latent space and obtain a style feature for each object.

Apart from images 3D FUTURE dataset includes mesh representations for all its objects. Using these meshes we extract for every object a voxel grid representation with dimensions $32 \times 32 \times 32$, in order to train a voxel grid autoencoder. In this case we again assume that the style features are obtained from the output of the trained encoder as latent vectors of length 512. Because the style is something subjective the idea here was to see which case can capture better the style during our experiments, or otherwise from which factor style depends most. In the image autoencoder due to the fact that we encode images, our latent vectors capture information such as the texture of the object, its color and its projection into an image plane. On the other hand voxel grids contain information only about the 3D shape of the object and as a result this information is captured in the latent space of our trained voxel grid autoencoder.

In the final approach we utilize an already pre-trained model in order to extract our style features. Specifically we are using the image encoder of CLIP model [20]. Clip is a model with objective task to find similarities between images and text descriptions. It has been trained with millions image, text pairs and is used in many state of the art research projects as it is a very powerful model. By using the images of 3D FUTURE dataset we encode them by using the pre-trained Clip image encoder and obtain latent vectors of length 512. This is the reason we build our autoencoders with latent space dimension equal also with 512, so that the final comparison between them to be fair. Finally for each one of the above approaches after
Figure 3.3: The occurrence frequency of all possible style categories for (a) bedrooms and (b) living rooms.
we extract the latent vectors we transform them to unit vectors, simply by dividing each latent vector with its magnitude. During experiments we observe that having unit style features facilitates and alleviates the training process.

### 3.4 Defining Parameters Distributions

Having defined each parameter of our objects and the way we extract it, will make a lot easier the description of our distributions. Specifically, we have maintained the same distributions proposed in [18] in order to model class, translation, angle and size of objects. For the class category having defined the possible categories as hot label vectors we want our model to calculate during training a categorical distribution over the possible categories for the missing object of the partial scene. In order our model to learn making correct predictions for the class category we minimize the negative log likelihood of the target value, which as we have hot label vectors is equivalent with minimizing the categorical cross entropy. Therefore, if we assume that our target hot label vector is \( y = [y_1 y_2 \ldots y_C] \), where with \( C \) we denote the number of possible categories and our predicted categorical distribution is \( \hat{y} \) we can compute the categorical cross entropy loss using Eq. (3.14). During inference the probability categorical distribution is passed through a softmax function in order the probabilities to be normalized. Opposite with other approaches which choose the sample with the highest probability, we use inverse transform sampling in order to sample from this multinomial distribution. The extracted probability distribution after we have applied the softmax function sum up to 1. If we divide the range \([0, 1]\) in spaces equal with the number of categories, where the length of each space is equal with the probability of the corresponding category, then by sampling from the uniform distribution \(U(0,1)\) a random variable \(x\), we look in which space, \(x\) falls, in order to choose one category. Any number \(x\) inside the range \([0, 1]\) has the same probability to be generated from \(U(0,1)\). Therefore the higher the extracted probability of one category, the longer also the length of the corresponding space and as result \(x\) has higher probability to fall inside this space. This is how inverse transform sampling works.

\[
- \sum_{i=1}^{C} y_i \log(\hat{y}_i) \quad \text{(3.14)}
\]
In the case of translation, size and angle the model predicts a probability distribution for each parameter. Actually it predicts three distributions for the translation, one for each axis and another three for the size, one for each dimension. In order to model these parameters we use mixture models. Let us see how the likelihood is being computed in mixture models. If we assume that we have a distribution \( f(x; \mu_i, \sigma_i) \) and a mixture model of \( N \) such distributions, then we can write for the distribution \( G(x; \mu, \sigma) \) of the mixture model:

\[
G(x; \mu, \sigma) = \sum_{i=1}^{N} \pi_i f(x; \mu_i, \sigma_i)
\]  
(3.15)

where \( \pi_i, \mu_i, \sigma_i \) are respectively the weight, the mean value and the standard deviation of each distribution \( f \). The log-likelihood of such models can be written as:

\[
\log(G(x; \mu, \sigma)) = \log\left(\sum_{i=1}^{N} \pi_i f(x; \mu_i, \sigma_i)\right) = \\
\log\left(\sum_{i=1}^{N} \exp\left[\log(\pi_i f(x; \mu_i, \sigma_i))\right]\right) = \\
\log\left(\sum_{i=1}^{N} \exp\left[\log(\pi_i) + \log(f(x; \mu_i, \sigma_i))\right]\right)
\]  
(3.16)

In order to optimize neural networks using negative log likelihood loss of mixture models, the network must predict the \( \pi_i, \mu_i, \sigma_i \) for every distribution and then by using Eq. (3.16) the whole likelihood is being calculated. In our case we assume that each one of these parameters follow a mixture of \( N \) logistic distributions.

We assume that we have \( N = 10 \) logistic distributions in order to model each parameter. The closed form pdf of the logistic distribution is given by:

\[
f(x; \mu, \sigma) = \frac{e^{-\frac{x-\mu}{\sigma}}}{\sigma\left(1 + e^{-\frac{x-\mu}{\sigma}}\right)^2}
\]  
(3.17)

If we observe (3.17) we can see that in the denominator there is a multiplication with the standard deviation. As a result according with (3.16) by using directly \( \log(f(x; \mu, \sigma)) \) in order to predict the log likelihood of these parameters, if our network makes predictions close to zero for standard deviation the likelihood goes to infinity and becomes intractable. One solution would
be to select fixed standard deviations in order to train our model. However because translation, angle and size are described as numbers and we can find their ranges across the training scenes, it would not be fair to use fixed standard deviations. By utilizing the dataset we can find very good estimators for the standard deviations and as a result we oversimplify the task of our model. For that reason we use the proposed approach in [22] where they compute a discretized logistic mixture likelihood, in order to learn autoregressively a distribution over the pixel intensities of images. In [22] they are exploiting the fact that the cumulative distribution function of logistic distribution is the sigmoid function, where the standard deviation appears only inside an exponential term as we see in (3.18), without however creating any problem if its value tends to zero.

\[ F(x; \mu, \sigma) = \frac{1}{1 + e^{-\frac{x-\mu}{\sigma}}} \]  

(3.18)

They assume that pixel intensities follow a discretized mixture of logistic distributions, as pixel intensities in the RGB scale can have integer values in the space \([0, 255]\). Because they use a discrete pdf in order to model their data, the likelihood of a target value to be generated is equivalent with the probability of this target value to be generated. By dividing the support of their distribution into 256 bins, one for every color of the 256 possible, given a target color \(x\) they find the corresponding bin. By using the values of the previous and next bin \(x_-\) and \(x_+\) respectively and network’s predicted values for \(\mu_i, \sigma_i\) they calculate the term \(f(x; \mu_i, \sigma_i)\) of (3.16) for every distribution by utilizing the CDF, as \(f(x; \mu_i, \sigma_i)\) is probability. Therefore using (3.18)

\[ f(x; \mu, \sigma) = F(x_+; \mu, \sigma) - F(x_-; \mu, \sigma) \]  

(3.19)

By subtracting the values of the CDF which correspond to the bins around the desired one they find the desired probability-likelihood and optimize their model. The fact that their data is discrete gives them the ability to exploit the CDF in order to find the likelihoods for their target data. In our case we have continuous random variables, and more specifically as we mention before all the parameters can have any value in the space \([-1, 1]\]. Therefore by dividing this space into bins, given a target value we found inside which bin this value falls, and compute the values of the sigmoid CDF for the previous and next bin. By using (3.19) and the predicted values of our network we compute an approximation of the likelihood of our target value. Having defined approximately \(f(x; \mu_i, \sigma_i)\) for all the 10 distributions
we just then use (3.16) in order to define the whole likelihood. During inference where we have estimated the parameters of these distributions in order to sample one value we use first inverse transform sampling in order to sample one of the 10 distributions, according to the predicted weights \( \pi_i, i = [1, \ldots, 10] \) and then we sample a value \( X \) from this one logistic distribution (let it be \( j \)) by using (3.20), where \( u \) is sampled from a uniform distribution with zero mean and unit standard deviation \( u \sim U(0, 1) \).

\[
X = \mu_j + \sigma_j (\log(u) - \log(1 - u))
\]  

(3.20)

As far as the style concerned we used two different approaches. Firstly as we mentioned we have used the style categories labels which we described with another categorical distribution. The approach is exactly the same with the case of the class category so there is no need for further explanation. However, in the case of style features we use a Gaussian Mixture Model in order to describe them. Specifically we assume that our style features of length 512 follow a mixture of Multivariate Gaussian distributions. Similar with the previous case where we also used mixture model for our predictions, our network predict for 10 Gaussian Multivariate distributions their parameters. The only difference is that in this case we tried both constant and learnable covariance matrices for every distribution, in order to test which variation performs better. In fact we never predict a whole covariance matrix when we want to predict a distribution, as we assume that the covariance is diagonal, such as in the case of VAE architectures. Hence we predict the variances across every dimension or the main diagonal of the covariance matrix. For the first variation, regarding the variances, we assume that the variances across each dimension are constant and equal with each other. Specifically we have chosen identity covariance matrices for the GMMs of our style features. As a result the network makes prediction only for the means and weights of each distribution. For the second variation we train our network in order to learn the variances too. In the following equation we see the pdf of the multivariate Gaussian distribution where with \( D \) we denote the length of our style features.

\[
f(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D | \Sigma |}} e^{-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)}
\]  

(3.21)

By using (3.16) and (3.21) we must compute the likelihood of the Gaussian mixture model. However for these calculations we do not have to use directly (3.21) as we can obtain a simpler form for \( \log(f(x; \mu, \Sigma)) \).
\[
\log(f(x; \mu, \Sigma)) = \log\left( \frac{1}{\sqrt{(2\pi)^D \left| \Sigma \right|}} e^{-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)} \right)
\]

\[
= \log\left( \frac{1}{\sqrt{(2\pi)^D \left| \Sigma \right|}} \right) + \log\left( e^{-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)} \right)
\]

\[
= -\log\left( \sqrt{(2\pi)^D \left| \Sigma \right|} \right) - \frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)
\]

\[
= -\frac{D}{2} \log(2\pi) - \frac{1}{2} \log(\left| \Sigma \right|) - \frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)
\]

\[
= -\frac{1}{2} D \log(2\pi) + \log(\left| \Sigma \right|) + (x - \mu)^T \Sigma^{-1} (x - \mu)
\]

By using Eq. (3.16) and (3.22) we can write for the negative log likelihood of our GMM which we want to minimize

\[
-\log(G(x; \mu, \Sigma)) = -\log\left( \sum_{i=1}^{N} \exp\left[ \log(\pi_i) + \log(f(x; \mu_i, \sigma_i)) \right] \right) =
\]

\[
-\log\left( \sum_{i=1}^{N} \exp\left[ \log(\pi_i) - \frac{1}{2} \left( D \log(2\pi) + \log(\left| \Sigma \right|) + (x - \mu)^T \Sigma^{-1} (x - \mu) \right) \right] \right)
\]

During inference, we use the predicted parameters of our network for the 10 distributions and by using inverse transform sampling over the weights \( \pi_i \) we choose one multivariate Gaussian distribution. In order to sample from this distribution a style feature \( X \) we must first calculate the Cholesky decomposition of our covariance matrix. In the case of diagonal matrices Cholesky decomposition is also a diagonal matrix, whose diagonal elements are the square roots of the corresponding diagonal elements of the covariance matrix. In our case where we have identity covariance matrix the Cholesky decomposition \( C \) is the same with our covariance matrix. By sampling \( D \) numbers from a normal distribution with zero mean and unit variance, we create a vector \( u \) with the same length as the predicted \( \mu \) and then by using (3.24) we take our sample.

\[
X = \mu + u^T C
\]

(3.24)
3.5 **RETRIEVAL**

Having extracted the parameters of the objects during inference the question is how we synthesize indoor scenes, as our predicted parameters refer to bounding boxes. We can use these parameters in order to create rectangular meshes and by sampling a floor layout to synthesize a scene. In Figure 3.4 we can see some examples of these bounding boxes scenes, both for bedrooms and living rooms.

![Figure 3.4: Scene generation using bounding boxes for (a) bedrooms and (b) living rooms.](image)

In order to create scenes with furniture meshes and not bounding boxes, we use a retrieval process. In [18] they use the predicted class category and size in order to retrieve the appropriate objects from the 3D FUTURE dataset. First they filter 3D FUTURE furniture according to the predicted class category and keep only those with the same class. Then from the already filtered they choose that with the smallest mean square error loss between its bounding box dimensions and the predicted size dimensions. As we have mentioned before for each one of the 3D FUTURE objects we have extracted, its bounding box parameters are in the scene scale, and as a
result we can apply MSE loss and compare these objects according to their size with the predicted of our network. As soon as they have retrieved all the objects they place each retrieved object in the scene using the predicted parameters from the network for the translation and angle. According to size they use the size of the retrieved object and not the predicted one. All the approaches that utilize a retrieval process follow the same technique, as they can compare the extracted parameters with the corresponding parameters of the objects in the datasets only by using the predicted class and the predicted size dimensions.

In our case when we use the style labels of the dataset in order to train our network, during retrieval we add one more filtering regarding the predicted style label. More specifically we filter first the objects according to the predicted class category and then apply to the remaining one extra filtering according to the predicted style category. Finally having found the objects that match both in the class and style category with the predicted ones, we utilize the predicted size dimensions just like [18] in order to choose the best candidate object. Opposite to ATISS in order to place the retrieved objects inside the scene we scale the meshes of the retrieved objects according to our size dimensions predictions.

When we utilize style features we follow a different approach. While the retrieval process is always based on the class and the size dimensions, we create a style based retrieval approach, without having to utilize the size. However we still use a class filtering at the beginning of the retrieval and believe that this facilitates a lot the whole process. The idea of our approach is that in order to compare features we can utilize their dot products. As we know two unit vectors are more similar when their dot product is closer to 1. If it is 1 then the angle between the two unit vectors is zero and in fact the two vectors are the same. Having this in mind we transform our predicted style feature vector into a unit vector in order to be able to apply the above idea. The style features of all objects in the dataset are already unit vectors, as we transformed them to unit in order to use them in that form during training. We implement two variations of style based retrieval. In both as first step we also filter according to the class category. In the first variation we use the predicted style feature in its unit form and compute its dot product with the style features of the filtered objects. The retrieved object is this with the higher dot product. In Figure 3.5 we present the whole idea.

The second variation implements an autoregressive style based approach, as we are utilizing also the style features of the already retrieved objects
in order to retrieve a new object. Particularly, we first implement the first variation

\[\text{Objects} \quad o_1 \quad o_2 \quad o_3 \quad o_4 \quad \ldots \quad o_M\]

\[\text{Classes} \quad c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_M\]

\[\text{Style features} \quad z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_M \quad \|z_i\| = 1\]

\textit{Class network’s prediction:} \(\hat{c}\)

\textit{Objects are filtered according to} \(\hat{c}\)

\[\text{Style network’s prediction:} \quad \hat{z} = \frac{\hat{z}}{\|\hat{z}\|}\]

\[z_1 \cdot \hat{z} \quad z_3 \cdot \hat{z} \quad z_5 \cdot \hat{z} \quad z_M \cdot \hat{z}\]

\[\text{MAX} \quad z_M \cdot \hat{z}\]

\textit{Retrieved object} \(o_M\)

\textbf{Figure 3.5:} Simple style based retrieval.

but instead of just keeping the object with the highest dot product we are sorting the candidate objects in descending order, according with their dot product with the predicted style feature. From the sorted objects we keep the half with the highest dot product and then for each one of them we form a vector of style features by using the style features of the already retrieved objects. If we assume that we have \(N\) candidates and we have already retrieved \(M\) objects, we are forming \(N\) such vectors \(Z_i, i = 1, \ldots, N\) with dimensions \((M + 1) \times 512\). From each one of these vectors \(Z_i\) we can compute the corresponding matrix \(W_i = Z_i^T Z_i\), which contains all the dot products between the style features of the \(M + 1\) objects. However, because \(M_i\) is symmetric some of the dot products are repeating. Therefore, we keep the upper triangular matrix, sum all the dot products and then subtract
Class network’s prediction: \( \hat{c} \)

Objects are filtered according to \( \hat{c} \)

Style network’s prediction: \( \hat{z} \)

Sort candidate objects in descending order according to \( z_i \cdot \hat{z} \)

Keep the half

Already retrieved objects

Group the retrieved and the candidate objects

Form arrays with the unit style features of each group

Calculate matrices \( Z_1^T Z_1 \) and \( Z_2^T Z_2 \)

Calculate \( F(Z_1^T Z_1) \) and \( F(Z_2^T Z_2) \)

\[ \max(F(Z_1^T Z_1), F(Z_2^T Z_2)) = F(Z_2^T Z_2) \]

\( \text{Retrieved object: } o_5 \)

Figure 3.6: Autoregressive style based retrieval.
the dot products of the main diagonal elements. In the main diagonal the dot products are always 1 because they are calculated between the same style features. From the above sum we calculate its mean value. In Figure 3.6 where we describe our autoregressive style based retrieval pipeline, the computation of this mean value is denoted as function $F$. This mean value is a style consistency factor as it has been calculated using all the objects, and because it is a mean of dot products between unit vectors, we want to be as close to one as possible. Having calculated these mean values for all the possible candidates we keep the object for which we extract the highest mean value. This idea, which indicates style similarity between the objects in the scene is one of the metrics we use in the next chapter, in order to evaluate our results.
EXPERIMENTAL EVALUATION

In this chapter we present analytically our experiments, together with our results. We begin first from the autoencoders that we have trained in order to extract style features for the objects of the dataset and then we proceed on the experiments regarding our generative model. We evaluate our methods using specific metrics in order to find which approach behaves better and why. In the end of the chapter, we present the results of our permutation study in order to find the optimal prediction order between the estimated distributions. Moreover, we have included also some failure cases during scene generation, while utilizing our model.

4.1 TRAINING IMAGE AUTOENCODER

We have trained our image Autoencoder for 300 epochs, in order to use the encoder architecture and extract style features. The encoder is compromised by four convolutional layers with Relu activation function between them, except from the last one where we do not use any activation function. The decoder is compromised by four transpose convolutional layers with Relu activation functions between them. In the output of the decoder we use a Sigmoid activation function. All the images are in RGB scale so we transform them to the range \([0, 1]\). We do not normalize the images according to the mean and standard deviation of the dataset because the final results were not as good as without normalization. The 3D FUTURE dataset contains 9992 objects so we had 9992 images to train our autoencoder. From these images we create two random sets one with 9042 and one with 950 images, which were our training and test sets respectively. We use learning rate \(10^{-3}\), and batch size 300. We have trained our autoencoder for 300 epochs using mean square error as loss function and Adam optimizer. For the training we used Google Collab Pro+ and a P100 GPU for 15 hours. In Figure 4.1 we present the train and test loss for our image autoencoder and in Figure 4.2 we give some reconstructions that we obtained for five images of the test set, at various stages of the training.
Figure 4.1: Mean square error loss during training and testing of our image autoencoder.

Figure 4.2: Reconstructions of our Image autoencoder. In the first row we have six images from the test set and in the subsequent rows their reconstructions at various stages of training.
One problem with autoencoder architectures is that the latent space vectors are not always representative latent entities of the corresponding inputs. As a result, if we apply linear interpolation between two latent vectors and feed these interpolated vectors in the trained decoder, there is no continuity between the decoded outputs. For that reason, we randomly choose pair of images from the test set and compute their latent vectors through our trained encoder. By applying linear interpolation between these pairs of latent vectors and passing them to our trained decoder, we observe that the corresponding outputs appear a smooth transition from one to another. In Figure 4.3 we present some of these pairs. In the first row we give the input images together with their reconstructions. In the following rows we present the decoded outputs of the interpolated latent vectors.

Figure 4.3: In the first row of each grid we have pairs of images from our test set, together with their reconstructions. For each pair after calculating the latent vectors using our trained encoder, we apply linear interpolation between them and feed the interpolated vectors to our trained decoder. The decoded outputs are shown in the rows 2 – 4 of each grid.
4.2 TRAINING VOXEL GRID AUTOENCODER

Apart from images, 3D FUTURE dataset includes mesh representations for all its objects. Using these meshes we extract for every object a voxel grid representation with dimensions 32x32x32, in order to train a voxel grid autoencoder. Before feeding voxel grids in the encoder we use a dropout layer with rate 0.5, in order to reduce overfit effects. Our encoder is compromised by six convolutional 3D layers. After each layer we use a batch normalization layer, followed by a Relu activation function. The encoder apart from the convolutional part has a linear part that projects the output of the convolutional part to the desired latent dimension. Similar the decoder has also a liner part and then six transposed convolutional layers. After each transpose convolutional layer we apply again batch normalization and then a Relu activation function, except from the last layer where we utilize a Sigmoid. We spilt the 9992 voxel grids into a train and test set of 9492 and 500 voxel grids respectively. We use the same learning rate, batch size and optimizer with the image encoder. However, in that case we use binary cross entropy loss in order to optimize our architecture. For the training we have used a NVIDIA GeForce RTX 3090 GPU for 20 hours. In Figure 4.4 we give the learning curves both during training and testing and in Figure 4.5 we present five voxel grids from the test set together with their reconstructions at various stages of training.

![Figure 4.4: Binary cross entropy loss during training and testing of our voxel grid autoencoder.](image)
4.3 EXPERIMENTAL RESULTS

In order to train our method we use learning rate $10^{-4}$ and Adam optimizer. The batch size was defined during training as 128. We have trained one model for every style variation we have mentioned. For the case, where we use CLIP image encoder, we train the model both with constant and learnable variances. We generate bedrooms and living rooms by using all the style approaches that we analyze in the previous chapter. We also retrain ATISS, which is the state of the art, in order to be able to compare our implementations. In Figures 4.6 and 4.7 we have included the losses for all models during training. Specifically we present the loss for each attribute and the total loss, both for living rooms and bedrooms. In Figures 4.8, 4.9 and 4.10 we present generated bedrooms and living rooms respectively from all the models we have experimented with. By observing firstly the learning curves we see that curves are very similar and we can not drawn conclusions by observing them. In the case of style we observe that style is converging very fast and then remains constant. From Figures 4.8 and 4.9 again we can not decide which model is superior, as all are able to generate elegant scenes. The only conclusion we can make, is that by using our methods, some characteristic objects in each type of scene, such as nightstands in the case of bedrooms and dining chairs in the case of living rooms, tend to be more similar in our models. We use this conclusion in order to evaluate all the models in the following section.
Figure 4.6: Learning curves for all the models we have trained for (a) class, (b) translation, (c) angle, (d) size, (e) style, (f) total loss in the case of bedrooms.

Figure 4.7: Learning curves for all the models we have trained for (a) class, (b) translation, (c) angle, (d) size, (e) style, (f) total loss in the case of living rooms.
Figure 4.8: Scene generation in the case of bedrooms using (a) ATISS[18], (b) our model using style labels, (c) our model using style features from image AE, (d) our model using style features from voxelgrid AE and (e) our model using style features from CLIP [20] image encoder.
Figure 4.9: Scene generation in the case of living rooms using (a) ATISS\cite{18}, (b) our model using style labels, (c) our model using style features from image AutoEncoder, (d) our model using style features from voxelgrid AutoEncoder and (e) our model using style features from CLIP \cite{20} image encoder.
As we have already mentioned we can not interpret style consistency either from the learning curves either from the generated scenes. The learning curves are very similar and also ATISS is also able to generate elegant scenes. As a result during our evaluation study we use specific metrics in order to be able to compare all these models regarding the style consistency across them. We assume that the style parameter can be considered as the similarity between objects in a scene. By using the same idea with the autoregressive style based approach that we described in the previous chapter we create the Style Similarity Measure. In Figure 4.11 we describe our idea. Having predicted the objects and their style features for a scene layout by using our network, we first transform our style features into unit vectors. By using vector $Z = [z_1z_2 \ldots z_M]^T$ with dimensions $M \times 512$, if we multiply it with its transpose $Z^T$ we obtain the matrix $ZZ^T$ of dimension $M \times M$, which contains the dot products between the style features of any pair of objects in the scene. As this matrix is symmetric its upper and lower triangular matrices are exactly the same and contain the dot products between the same pairs of objects. Therefore, we define as SSM (Style Similarity Measure) of the scene the mean value of the unique pairs inside a scene. In (4.1) we give the mathematical definition of SSM for a scene with $M$ objects. We are not interested about pairs between the same objects as a dot product between the same style feature is always 1. That is why we are subtracting in (4.1) the sum of the diagonal elements.

$$SSM = \frac{2}{M^2 - M} \left[ \frac{1}{2} \left( \sum_{i=1}^{M} \sum_{j=1}^{M} z_i \cdot z_j - \sum_{i=1}^{M} z_i \cdot z_i \right) \right] \quad (4.1)$$

As a dot product between two unit vectors has the same range with the cosine of their angle, which is $[-1, 1]$ we can express SSM for each scene as percentage by using (4.2).
In order now to evaluate our models by utilizing SSM we first generate for each model as many scenes as our test set. For each model for every generated scene we calculate SSM using (4.2) and then we take the mean values across all the generated scenes. We named this mean value which represents the style similarity measure for a collection of scenes MSSM (Mean Style Similarity Measure) and it is representative of the style consistency, which is able to produce a model across its generated scenes. In the case of ATISS and the variation of our model where we use style labels we can not calculate directly MSSM. In order to be able to calculate MSSM, after each retrieval procedure we collect the retrieved objects and calculate their style features by feeding their images to CLIP [20] image encoder and then transform these features into unit vectors. With the same way we calculate MSSM for the test set of our dataset, but in that case except from CLIP image encoder we also use our autoencoders, in order to notice

\[
SSM(\%) = \left( \frac{SSM + 1}{2} \right) \times 50 \tag{4.2}
\]

Figure 4.11: Our network predicts \( M \) objects together with their style features \( z_i \). After we transform the style features into unit vectors we are calculating the dot products between all the combination of objects and obtain a \( M \times M \) table, in order to define the Style Similarity Measure of the scene. As the above tabel is symmetric the orange areas we have drawn contain the same elements.
differences between them. For the variations of our model, where we utilize style features we have calculated the MSSM for both the style based retrieval approaches we introduced. However we have also include an approach, where we do not use the predicted style features for MSSM calculation but those features which correspond to the retrieved objects we obtain by using these models. For this last case, we must mention that for the models where we use during their training style features extracted from our image and voxelgrid auoencoder, we retrieve the objects based on these style features but in order to compute the MSSM we use again CLIP image encoder. We explain later the reason behind this choice. In Tables 4.1 and 4.2 we give the MSSM values for every model and the test dataset for bedrooms and living rooms respectively, together with their standard deviation $\sigma$.

By observing Tables 4.1 and 4.2 we reach the following conclusions. First of all, when we use the simple style based retrieval we introduced our model has similar behavior with ATISS. However, when we use the autoregressive style retrieval we outperform ATISS. In the case where we utilize directly the predicted style features without any retrieval, MSSM values are much higher than in the other cases. This is also an indication of the real potential of our network regarding style consistency and the fact that we are limited by the retrieval process in order to generate realistic scenes. If we observe the MSSM values for the ground truth test scenes we can see that in cases, where we use the autoencoders in order to extract the style features the MSSM values are relatively low, especially in the case of the voxelAE. This fact however, must not be considered as a negative indication about the quality of style features that our autoencoders are able to produce and their ability to be utilized during the training of our network. MSSM metric might not be used validly in them as due to the dot products computations that take place, style features must be representative of the objects, as it seems to happen when we use CLIP image encoder in order to extract our styles features. We believe this is the reason why in these cases the $\sigma$ of MSSM is larger. However if we look in Tables 4.1 and 4.2 the MSSM values that correspond to our models, where the style features were extracted using autoencoders, we can see that their performance is comparable with other models. In these models the retrieval process is done entirely by utilizing only these style features. In order to prove that the whole training using the autoencoders features produces similar results with the other methods we use CLIP image encoder in order to define the style features for the retrieved objects and proceed to MSSM calculations.
The standard deviation of MSSM for all models is relatively low and this is much desirable as indicates that our generated scenes appear similar style consistency. As a final conclusion we can support that our model is able to outperform ATISS regarding style consistency regardless of the variations of our model.

In order to evaluate further all these models we compute the probability regarding the occurrence frequency for particular combination of characteristic objects in each type of scene. Specifically, in the case of bedrooms we...
compute the probability of generating two same nightstands in the scene, and in the case of living rooms the probability all the dining chairs in the scene to be the same. However, there are ground truth and generated bedrooms - living rooms with one or no nightstand or with no dining chairs. Therefore, we first compute the number of scenes which contains two nightstands in the case of bedrooms, or more than one dining chair in the case of living rooms and then we compute the frequency of the desired combinations of furniture in order to define these probabilities. It is something common one bedroom having same nightstands and a living room same dining chairs. That is why we have chosen to compute the probability of these combinations across the generated scenes. They reveal objectively style consistency, without relying on personal preferences. Moreover, in order to evaluate whether our method is facilitated or deteriorated we

<table>
<thead>
<tr>
<th></th>
<th>sizes retrieval</th>
<th>predicted</th>
<th>style labels</th>
<th>imgAE</th>
<th>voxelAE</th>
<th>CLIP [20]</th>
<th>simple</th>
<th>autoregressive</th>
<th>MSSM (%)</th>
<th>σ (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Truth⁴</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>91.39</td>
<td>1.3</td>
</tr>
<tr>
<td>Ground Truth⁴</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>62.45</td>
<td>3.61</td>
</tr>
<tr>
<td>Ground Truth⁴</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>59.07</td>
<td>3.81</td>
</tr>
<tr>
<td>ATISS [18]</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>90.84</td>
<td>1.03</td>
</tr>
<tr>
<td>ATISS [18]</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>91.06</td>
<td>1.11</td>
</tr>
<tr>
<td>Ours</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>90.83</td>
<td>1.07</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>91.04</td>
<td>1.08</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>91.1</td>
<td>0.99</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>91.3</td>
<td>1</td>
</tr>
<tr>
<td>Ours⁴</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>94.68</td>
<td>0.36</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>91.02</td>
<td>0.95</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>90.84</td>
<td>1.13</td>
</tr>
<tr>
<td>Ours⁴</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>93.29</td>
<td>0.57</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>90.71</td>
<td>0.6</td>
</tr>
<tr>
<td>Ours⁴</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>94.43</td>
<td>0.66</td>
</tr>
<tr>
<td>Ours⁴</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>98.58</td>
<td>0.78</td>
</tr>
</tbody>
</table>

**Table 4.2:** MSSM values and its standard deviation for the ground truth living room test scenes and the generated ones from ATISS [18] and our models.

1. Style features have extracted using CLIP image encoder
2. Style features have extracted using our image autoencoder
3. Style features have extracted using our voxel grid autoencoder
4. In these cases in order to calculate MSSM we use directly the predicted style features.
compute other two probabilities. The first one is the probability the generated objects not to be aligned with the floor layout. The second one is the probability of generating overlapping objects inside a scene. The first one is computed by comparing the coordinate of the centroid of the bounding box of each object, except lamps, along the vertical axis with the half of its height. The optimal situation is these values to be equal in order objects being perfectly aligned with the floor layout. In the case of bounding boxes which contain lamps we do not want this condition to be applied, as lamps must be higher from the floor layout. Therefore, we do not examine this condition for the bounding boxes that correspond to lamps inside a scene. The perfect alignment condition between objects and floor is described by the following equation:

\[ t_z = \frac{s_z}{2} \]  

(4.3)

In Figure 4.12 we give an example of a bounding box, which satisfies the condition of Eq. (4.3). The plane xOy corresponds to the floor layout.

![Figure 4.12: An example of perfect alignment between a bounding box and xOy plane, with the latter corresponding to the floor layout.](image)

In order to calculate the second probability we must first define mathematically the overlapping condition between two objects. Let us assume
Figure 4.13: Both (a) and (b) are cases where the overlapping condition does not hold. Particularly in (a) the inequality described by Eq. (4.4) does not hold for any axis while in (b) holds only along axis $x$ and $y$. 
we have two bounding boxes $b_1$ and $b_2$. The coordinates of their centroids are $t_{x_1}$, $t_{y_1}$, $t_{z_1}$ and $t_{x_2}$, $t_{y_2}$, $t_{z_2}$ respectively. Furthermore the dimensions of these bounding boxes are $s_{x_1}$, $s_{y_1}$, $s_{z_1}$ and $s_{x_2}$, $s_{y_2}$, $s_{z_2}$. These values could be the predictions of our model for two objects, regarding the size and the translation of their bounding boxes. In order these objects to be overlapped there is one condition, which must be satisfied along all three axis $x, y, z$ and is described by the following inequality.

$$|t_{i_1} - t_{i_2}| < \frac{s_{i_1} + s_{i_2}}{2}, i = x, y, z \quad (4.4)$$

If inequality (4.4) holds only for one or two axis we do not have an overlapping condition. For example, if we imagine two bounding boxes one above the other with some vertical distance between them, if the projections of the two boxes in the $xOy$ plane do not intersect the boxes do not overlap. If these projections are overlapped, and as a result inequality (4.4) is satisfied along axis $x$ and $y$, the boxes again do not overlap, as (4.4) is not satisfied along the vertical axis $z$. In Figure 4.13 we present two examples where the two bounding boxes do not overlap. In Figure 4.13 (a) the inequality described by Eq. (4.4) does not hold for any axis, while in Figure 4.13 (b) the inequality holds only along $x$ and $y$ axis.

Apart from these probabilities in order to evaluate even further our models we compute also FID score. In order to compute the FID score we are using images from our ground truth test scenes and images from our generated scenes. In Tables 4.3 and 4.4 we have included all these probabilities and the FID scores for bedroom and living rooms respectively.

By observing Tables 4.3 and 4.4 our conclusions are similar with these that have arisen from the MSSM values of Tables 4.1 and 4.2. Moreover, from these tables we can confirm our statement that we can extract the style features using our autoencoders in order to train our models. In some cases, when we use our autoencoders for the extraction of style features, we surpass in style consistency our variation which utilizes CLIP [20].

Furthermore, as we can see by using the predicted sizes we achieve almost zero probability of misalignments between objects and floor, while by using the retrieved sizes the percentage is higher. As this probability depends only from the height of the bounding boxes and the coordinates of their centroids along the vertical axis, its low value proves that our model has learnt very well to consider the location of an object along the vertical axis in order to predict the distribution for its height. Instead when we use the retrieved sizes of the bounding boxes we violate this predicted relation and as a result the number of misalignments across the scenes raises. The probability
of overlapped objects across scenes is a very representative indicator of the performance of our models, as if we have overlapped objects the scene generation task has failed. Regarding this probability we surpass ATISS in some cases, both for bedrooms and living rooms, by using our models. In the case of living rooms the task is more difficult as the model has to predict locations in bigger and more complex scene layouts, as we can see from Figure 4.9. This is the reason that the probabilities of overlapped objects across the generated living rooms are higher. However, the low probability of misalignment with the floor layout indicates that our model even in this case learns taking into account object position along the vertical axis in order to decide about its height. In Tables 4.3 and 4.4 we have also include the corresponding values for our test set. As we can see in the case of living rooms overlaps also exist across our ground truth scenes, and as a result the learning process becomes more difficult. As far as the FID is concerned, the fact that we achieve better scores in the case of living rooms with much higher probabilities of overlaps indicates that it is not a reliable metric in order to evaluate our experiments.

**Table 4.3:** Calculation of probabilities $P_1$, $P_2$, $P_3$ and FID score for the ground truth bedroom test scenes and the generated ones from ATISS [18] and our models.

<table>
<thead>
<tr>
<th>sizes</th>
<th>variation</th>
<th>style retrieval</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieved</td>
<td>predicted</td>
<td>style labels</td>
<td>ungAE</td>
</tr>
<tr>
<td>Ground Truth</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATISS[18]</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATISS[18]</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

1 Probability of same nightstands across scenes
2 Probability of misalignment between objects and floor across scenes
3 Probability of overlapped objects across scenes
4 In these cases our network predicts the covariance matrices of the GMMs
Table 4.4: Calculation of probabilities $P_1^1$, $P_2^2$, $P_3^3$ and FID score for the ground truth living room test scenes and the generated ones from ATISS [18] and our models.

<table>
<thead>
<tr>
<th></th>
<th>retrieved</th>
<th>predicted</th>
<th>style labels</th>
<th>imgAE</th>
<th>voxelAE</th>
<th>CLIP[20]</th>
<th>simple</th>
<th>autoregressive</th>
<th>$P_1^1$ ↑</th>
<th>$P_2^2$ ↓</th>
<th>$P_3^3$ ↓</th>
<th>FID ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Truth</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>98.43</td>
<td>0.51</td>
<td>9.54</td>
<td>-</td>
</tr>
<tr>
<td>ATISS[18]</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>50.87</td>
<td>7.16</td>
<td>59.97</td>
<td>38.86</td>
</tr>
<tr>
<td>ATISS[18]</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>46.37</td>
<td>0.47</td>
<td>61.5</td>
<td>45.36</td>
</tr>
<tr>
<td>Ours</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>66.94</td>
<td>25.55</td>
<td>56.73</td>
<td>37.86</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>68.75</td>
<td>0.51</td>
<td>58.12</td>
<td>45.84</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>40.57</td>
<td>0.51</td>
<td>85.69</td>
<td>55.4</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>69.37</td>
<td>0.51</td>
<td>85.69</td>
<td>53.98</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>53.72</td>
<td>0.51</td>
<td>61</td>
<td>48.29</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>33.23</td>
<td>0.33</td>
<td>61</td>
<td>51.74</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>70.67</td>
<td>0.33</td>
<td>61</td>
<td>50.33</td>
</tr>
</tbody>
</table>

1 Probability of same nightstands across scenes
2 Probability of misalignment between objects and floor across scenes
3 Probability of overlapped objects across scenes

During scene generation using our models, we observe something worth mentioning. As we can see in Figure 4.14 we have generated scenes where the predicted dimensions of each pair of nightstands are different. However, in all cases the nightstands are the same but in different size. All the three scenes have obtained by using the autoregressive retrieval approach we introduced. A such result is impossible with the until now size based retrieval procedures, as an object is retrieved based on its size. Especially in the cases of Figure 4.14 where the nightstands have quite big difference regarding their dimensions another retrieval approach would not be able to generate a such result. Therefore, we believe that our model in combination with our autoregressive style approach, can enhance style consistency across the scene, even if the predictions of the model are not optimal.
Despite the nightstands in all scenes are the same they have different size. With the classic retrieval process which is based on the size of objects this result would be impossible. By predicting style features through our network and utilizing the autoregressive style based retrieval approach we introduced we are able to generate scenes with style consistency even if network’s predictions regarding the size of the objects are not optimal.

### 4.5 Permutation Study

In this section we want to explore how the order of the predicted distributions affects style consistency and generally the whole generation process. It is not an ablation study as we do not remove any distribution in order to compare the new performance, but instead we just permute them. That is why we gave this name on this section. We have tried different variations regarding the order of the distributions and also the way that size and translation are predicted. As we have mentioned we predict three distributions for the translation, one for each direction, and other three for the size, one for each dimension. We refer for convenience to the translation distributions as $f_{tx}, f_{ty}, f_{tz}$ and to the size distributions as $f_{sx}, f_{sy}, f_{sz}$. From Figure 3.1 where we describe our model’s architecture we can see that the prediction process of each one of the three distributions both for size and translation is independent with the others. Each one of these distributions has its own MLP linear layers, from which its parameters are calculated, but their input is the same. Therefore the linear layers of $f_{tx}, f_{ty}, f_{tz}$ take the same input $Q$ and similar the linear layers of $f_{sx}, f_{sy}, f_{sz}$ take the same input $W$. To experiment with the way, the parameters of these distributions are being calculated we create two different approaches. The basic idea was to add some dependency in the prediction of these distributions. In order to achieve that for our first approach, in each subsequent layer apart from $Q$ and $W$ we give also the corresponding target value of the previous layer. In Table 4.5 we present our whole idea. The values $s_x, s_y, t_x, t_y$ in Table 4.5 correspond to target ground truth values during training and to predicted values during inference. The second approach we used in order to make the calculation of distributions more dependent was to create one MLP for
all the $f_{tx}, f_{ty}, f_{tz}$ and one for the $f_{sx}, f_{sy}, f_{sz}$. Instead of having three MLP layers, where each takes the same input and predicts 30 parameters for the 10 logistic distributions that describing each attribute, we have now two MLP layers, which take the same input as before but instead of 30 they predict 90 parameters. In Table 4.6 we describe this implementation.

Having defined these two extra approaches we built 8 variations of our model which were trained with style features, extracted from CLIP \cite{20} image encoder in bedroom scenes. In Table 4.7 we describe analytically all these variations.

<table>
<thead>
<tr>
<th>input to MLP linear layers</th>
<th>initial approach</th>
<th>suggested approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{tx}$</td>
<td>$Q$</td>
<td>$Q$</td>
</tr>
<tr>
<td>$f_{ty}$</td>
<td>$Q$</td>
<td>$[Q, t_x]$</td>
</tr>
<tr>
<td>$f_{tz}$</td>
<td>$Q$</td>
<td>$[Q, t_x, t_y]$</td>
</tr>
<tr>
<td>$f_{sx}$</td>
<td>$W$</td>
<td>$W$</td>
</tr>
<tr>
<td>$f_{sy}$</td>
<td>$W$</td>
<td>$[W, s_x]$</td>
</tr>
<tr>
<td>$f_{sz}$</td>
<td>$W$</td>
<td>$[W, s_x, s_y]$</td>
</tr>
</tbody>
</table>

Table 4.5: Instead of having the same input in order to calculate the parameters of each distribution, we concatenate to the input, the target values of the corresponding previous layer in the case of training or the predictions of the previous layer in the case of inference.

In Figure 4.15 we present the learning curves for all these variations. From the curves we observe that generally the negative log likelihood loss for the distribution of any attribute is smaller when the distribution prediction of another attribute has preceded. The most difficult distribution to be learnt from our model is that of the translation. However, despite its difficulty it is
Table 4.6: Instead of having three MLP layers for the translation and other three for the size, we use one for the whole size and one for the whole translation distribution but with triple output length than before.

<table>
<thead>
<tr>
<th>initial approach</th>
<th>suggested approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP input length</td>
<td>MLP output length</td>
</tr>
<tr>
<td>Q</td>
<td>30</td>
</tr>
<tr>
<td>Q</td>
<td>30</td>
</tr>
<tr>
<td>Q</td>
<td>30</td>
</tr>
<tr>
<td>Q</td>
<td>90</td>
</tr>
<tr>
<td>Q</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 4.7: Distribution prediction order for our 8 variations.

---

1. In this variation we use Table 4.5 for the translation.
2. In this variation use Table 4.5 for both size and translation.
3. In this variation we use Table 4.5 for the size and 4.6 for the translation.
4. In this variation we use Table 4.6 for the size and 4.6 for the translation.

very important for the scene generation task, as it decides where the objects are placed in the scene. The failure scenes involve most of the times overlaps between objects, which occur due to the incorrect translation predictions. It is more rare to have problems with the orientation of the objects or with their sizes. That is why in the most variations we have placed translation prediction in the end, in order to obtain as much possible information from the preceding distributions and its learning process to be facilitated. However, we can not come to conclusions only from the curves and for that reason in Table 4.8 we present all the already mentioned metrics for these 8 variations. We have not included FID score as it is not an
appropriate metric in order to evaluate style consistency and generally the scene generation process, as we have discussed before. For each variation we have included only the results from the autoregressive style based retrieval as it gave us the best results in its case. Furthermore, we have included the MSSM values for the cases where we use the predicted style features.

As we can see from Table 4.8 the MSSM values are all over 91(%) and its standard deviation across the generated scenes is relatively small, which indicates high style consistency. Moreover, the probability of obtaining a bedroom with same nightstands is also high and higher than the corresponding probabilities of our models, which are described in Table 4.3. It is impressive that all the variations have probability of same nightstands over 86(%) and five of them over 90(%). We believe that the approaches we have used in order to add some dependency between the translation and size distributions facilitate style consistency. However, as far as the general scene generation task is concerned, by looking probabilities $P_2$ and $P_3$ we can say that these approaches do not perform as well than before. As we can see the probability of overlapped objects across generated scenes is higher than our variations in Table 4.3. For this fact we believe is responsible that we use in all variations style features obtained from CLIP [20] image encoder. Even in Table 4.3

Figure 4.15: Learning curves for all the variations we have trained for (a) class, (b) translation, (c) angle, (d) size, (e) style, (f) total loss.
<table>
<thead>
<tr>
<th>MSSM</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>(%)</td>
<td>σ ↓</td>
</tr>
<tr>
<td>V₁</td>
<td>92.37</td>
</tr>
<tr>
<td>V₁</td>
<td>97.49</td>
</tr>
<tr>
<td>V₂</td>
<td>91.88</td>
</tr>
<tr>
<td>V₂</td>
<td>97.42</td>
</tr>
<tr>
<td>V₃</td>
<td>91.75</td>
</tr>
<tr>
<td>V₃</td>
<td>97.33</td>
</tr>
<tr>
<td>V₄</td>
<td>91.82</td>
</tr>
<tr>
<td>V₄</td>
<td>97.28</td>
</tr>
<tr>
<td>V₅</td>
<td>92.31</td>
</tr>
<tr>
<td>V₅</td>
<td>97.5</td>
</tr>
<tr>
<td>V₆</td>
<td>92</td>
</tr>
<tr>
<td>V₆</td>
<td>97.5</td>
</tr>
<tr>
<td>V₇</td>
<td>92.18</td>
</tr>
<tr>
<td>V₇</td>
<td>97.57</td>
</tr>
<tr>
<td>V₈</td>
<td>92.61</td>
</tr>
<tr>
<td>V₈</td>
<td><strong>97.66</strong></td>
</tr>
</tbody>
</table>

**Table 4.8:** Calculation of probabilities $P₁$, $P₂$, $P₃$ and MSSM for the generated bedrooms using the 8 variations we describe in 4.7. All these variations have been trained with style features extracted from CLIP [20] image encoder. During scene generation we use only the predicted sizes in all variations.

1. Probability of same nightstands across scenes
2. Probability of misalignment between objects and floor layout across scenes
3. Probability of overlapped objects across scenes
we can see that the corresponding approach gives us the highest probability of overlapped objects. The probability of misalignment between objects and floor, which indicates the performance of our network regarding the predicted relation between size and translation, is at low levels except from variation $V_6$. Some variations appear with higher $P_2$ probability than the corresponding CLIP variation of Table 4.3, but these probabilities remain also relatively small. We must stand however in the case of variation $V_6$ where the probability $P_2$ is very high. This number indicates that the predicted relation between the translation of the bounding boxes along the vertical axis and their height is incorrect. This results at flying objects, or objects under the floor layout. In the case where we use the retrieved sizes, as ATISS does, such results are more often as we do not respect the predicted relation between $t_z$ and $s_z$. This is also evident in Table 4.3 for the cases of ATISS and our variation where we also use the retrieved sizes. However, even in that cases the probability $P_2$ was not so high as in $V_6$ variation. Generally, we believe that the approach we describe in Table 4.5 is not suitable for scene generation tasks, as during inference a wrong prediction affects directly the subsequent predictions. Something also worth mentioning, is that variation $V_5$ where we use in both translation and size, the approach of Table 4.5 for their distributions, do not present similar behaviour. In the beginning we believe that this one was going to give us the worst results. Maybe we should generate more scenes in order to recompute our metrics and get a clearer explanation.

As a final conclusion from this study we believe that style is better to placed in the end in the prediction order, especially when we use style based retrievals and we want the quality of the predicted style features to be the optimal possible. As we have mentioned from the second chapter we believe that class must always placed first in the prediction order. For the distributions of the other three attributes, the distribution which is placed after the class has the least information in order to learn. For our experiments we have seen that size distribution can be placed first after the class without having difficulties during the learning process. Furthermore while the angle distribution is the easiest to learn, its learning process is facilitated when the translation distribution has preceded and derives information from it. This is reasonable to us as in a real life scenario in order to decide the orientation of an object we must specify first its location. As a result the order we propose for this task is **class, size, translation, angle, style**. If we choose not to use style based retrieval approaches, style distribution do not need to be placed in the end of the prediction order. In that cases
we propose the alternative variation `class, style, size, translation, angle`. Finally after our experiments we conclude that translation and size can be described either with our initial independent approach or the approach we describe in Table 4.6.

![Bedrooms](image1)

**Figure 4.16**: Failure cases obtained from our models for (a) bedrooms and (b) living rooms

### 4.6 Failure Cases

In this last section we present some failure cases both from our generative models and the dataset. The failure cases as we can see from Figure 4.16 involve mainly wrong orientation of objects, overlaps between them, absence of objects while other directly connected with them are still present and finally positioning of objects outside the floor layout. However, as we can see for almost each one failure case in Figure 4.16 there is its corresponding in Figure 4.17. For example a misoriented dressing table and an absent dining table appear in the left bedroom and living room respectively of both Figures 4.16 and 4.17. Before training we filtered the whole dataset in order to reject such scenes and not include them in our training set. However because 3D FRONT [7] and 3D FUTURE [8] datasets are relatively small we do not have the luxury for a stricter filtering. Unfortunately there are not
elegant, carefully created and detailed datasets for indoor scene generation. We believe that our method is able to achieve even better results and achieve greater performance if it is applied in a neater and more elegant dataset.

**Figure 4.17:** Failure cases obtained from the dataset for (a) bedrooms and (b) living rooms
5 CONCLUSIONS

Having conducted this Master thesis we refer to our overall conclusions. First of all as far as our contributions are concerned we are the first to incorporate style prediction during the scene generation task. As we see from the previous chapters by utilizing style in order to train our architectures we obtained the advantage to be able to generate more realistic scenes. Style is a factor which we take into account in order in real life to populate a scene with objects. Furthermore, by utilizing four methods in order to model the style we manage to conclude, which is the optimal type in order to model the style parameters. Our style labels variation is for sure an extension from previous approaches but the different styles we can predict with this variation are limited. On the other hand, style features are more promising and can alleviate incredibly the generation of scenes with style consistency. The idea of modelling style features by using a Gaussian Mixture Model, which parameters are learnable from the network, was successful and as we saw by utilizing this method we were able to surpass ATISS [18] in the metrics we introduced. Regarding the parameters of GMMs we use constant and learnable covariance matrices and obtain similar results with both approaches. Moreover, style features have not to be obtained from huge deep networks. As we saw by extracting style features from our shallow autoencoders we not only surpass ATISS in style consistency, but also in some cases our own model in which we use CLIP [20] image encoder. Comparing our two autoencoders we believe that the style features extracted from our voxelgrid autoencoder alleviated the scene generation task in the case of living rooms, while these from our image autoencoder facilitated the scene generation in the case of bedrooms. Specifically, we believe that in the case of bedrooms the optimal model is that which uses our image autoencoder style features and in the case of living rooms the variation which uses features from our voxelgrid autoencoder. We believe, that because a voxelgrid autoencoder encodes information from the shape of an object, the extracted style features accommodate the generation process in more complex and bigger layout such as living rooms. As contribution we must also refer our autoergressive style based retrieval. By using this approach during retrieval we were able to
generate scenes with style consistency even in cases, where the model’s predictions regarding size make it impossible for other approaches. From our permutation study we were able to conclude to an optimal prediction order regarding the distributions of objects attributes and to reject other alternative methods in order to predict the distributions of their size and translation.

Generally we believe that style based scene generation is much promising and has much to offer. However the retrieval process even in the case of our own autoregressive version is a limiting factor. A combination of a transformer based scene generation approach, which incorporates also style prediction, with an approach which uses radiance fields and does not need retrieval, will be revolutionary and get the scene generation task to a whole new level. We aim to implement this idea in the near future.
REFERENCES


