

NANTES UNIVERSITÉ

MASTER THESIS

Multimodal generative model for spatial graph

Author:

Robin POURTAUD 

Supervisor:

Hideaki TAKEDA 

Fabrice GUILLET 


*A thesis submitted in fulfillment of the requirements
for the degree of Master of Data Science*

in the

Polytech Nantes

March 26, 2024

Declaration of Authorship

I, Robin POURTAUD , declare that this thesis titled, “Multimodal generative model for spatial graph” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

NANTES UNIVERSITÉ

Abstract

National Institute of Informatics
Polytech Nantes

Master of Data Science

Multimodal generative model for spatial graph

by Robin POURTAUD 

In recent years, the adoption of 3D models has increased across various fields, including entertainment, medical imaging, and virtual reality. Concurrently, generative AI models have demonstrated significant success in generating diverse media types, such as text, images, and videos. Inspired by the principles of DALL-E 2 and Stable Diffusion models, which generate high-quality 2D images from textual inputs, this thesis aims to investigate the feasibility of extending these principles to generate 3D mesh structures from graph representations. To achieve this, we explore suitable Graph Neural Networks (GNNs) for graph embedding and adapt diffusion methods for 3D mesh data. Our proposed approach contributes to the understanding of generative models for 3D mesh structures and serves as a stepping stone towards generating more complex networks, such as knowledge graphs, which have wide-ranging applications in semantic web services, information extraction, and knowledge management.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Hideaki Takeda, for his invaluable guidance, support, and encouragement throughout the development of this paper. His expertise and insights have significantly contributed to the quality of the research presented in this work.

I am also grateful to the Takeda-Sensei's team and my fellow interns for the stimulating discussions and collaborative environment, which have inspired and enriched my research endeavors.

Special thanks go to my tutor, Fabrice Guillet, for his continuous support and advice during the course of my studies. His dedication to my academic progress has played a pivotal role in shaping my research interests and capabilities.

Lastly, I would like to acknowledge my master's supervisor, Hoel Le Capitaine, for his guidance throughout my years of master. His mentorship and expertise have been instrumental in the successful completion of this project.

I am sincerely grateful to all who have contributed to my research, both directly and indirectly, and for the opportunities that have been provided to me.

Contents

Declaration of Authorship	iii
Abstract	v
Acknowledgements	vii
1 Introduction	1
2 Digital Representation of 3D objects	3
2.1 Manifold	3
2.2 Simplicial Complexes	4
2.3 Meshes	5
2.4 Voxels	6
2.5 Point Clouds	6
2.6 Other Representations	7
3 Graph and Hypergraph	9
3.1 Graph: Definition	9
3.2 Hypergraph	10
3.3 Mesh to Hypergraph for 3D Object Representation	11
3.3.1 The .obj File Format	11
3.3.2 Transforming .obj Files to Hypergraphs	12
4 Literature Review: Graph Generation	13
4.1 Challenges	13
4.2 Embedding	14
4.2.1 Text Embedding	14
4.2.2 Geometric 3D object Embedding	15
Point clouds and Voxels	15
Mesh	16
4.2.3 Graph Embedding	17
4.2.4 Embedding Evaluation	19
Intrinsic Evaluation	19
Extrinsic Evaluation	19
4.3 Data Generation Task	19
4.3.1 Euclidean Data Generation	20
Generative Adversarial Networks	20
Diffusion	21
Variational AutoEncoder	21
4.3.2 Graph generation	24
4.4 Contrastive Learning and CLIP	24
4.4.1 Text Encoder	25
4.4.2 Image Encoder	25
Vision Transformer (ViT)	25
Residual Network (ResNet)	26
4.4.3 Shared Embedding Space	26
4.5 Stable-Diffusion	27

5 MeshGNN: Model proposition	29
5.1 Fine Tuning of Clip witout compatibility layer	29
5.2 Fine Tuning of Clip with compatibility layer	30
5.3 Generation	31
5.4 Limitations	32
6 Experiment: CLIP Generation of 3D model	33
6.1 Datasets and Datapreparation	33
6.1.1 ShapenetSem	33
6.1.2 MNISTSuperpixel	34
6.2 Implementation Details	34
6.3 Results and comments	34
6.3.1 Graph Variational AutoEncoder	34
6.3.2 Clip	36
6.3.3 Graph CLIP and Full Model	36
7 Conclusion and Future Directions	39
A NII Introduction	41
Bibliography	43

List of Figures

2.1	Sphere Manifold	3
2.2	Parametric representation of a circle in the Euclidean plane	4
2.3	Simplicial complexes of dimension 1, 2, 3, and 4	5
2.4	Mesh Representation of a table	5
2.5	A 3D object made of smaller cubes (voxels) representing a voxel-based structure	6
2.6	Point-cloud representation of face	7
2.7	Octree	7
3.1	A simple undirected graph with four vertices and four edges	9
3.2	A simple hypergraph with four vertices, one edge and one hyperedge	10
3.3	Mesh of a table 2.4 transform into a Graph Object	12
4.1	PointNet++ Qi et al., 2017 ("ours") compared to PointNet Qi et al., 2016 and the ground truth	16
4.2	A: Zachary Karate Club Graph, B: Zachary Karate Club Embedding in 2D (with ACP reduction) Source: Guo and Zhao, 2020	17
4.3	Diagram of a Generative Adversarial Network (GAN). The Generator (red) produces Fake Data (blue) from a Random Input (blue). The Discriminator (green) receives either Real Data (blue) or Fake Data (blue) and tries to distinguish between them.	20
4.4	Diffusion Process	21
4.5	Variational Autoencoder with graph data	22
4.6	Classification of deep generative models for graph generation problems, Source: Guo and Zhao, n.d.	24
4.7	Contrastive pre-training	25
4.8	Stable diffusion high quality image generation example, Source: AI, 2022	27
4.9	Stable diffusion architecture, Source: AI, 2022	28
5.1	Clip Naive, without compatibility layer	29
5.2	Clip training with compatibility layer	30
5.3	MeshGraph Full Architecture	31
6.1	Top MNISTSuperpixel Loss for VGAE	35
6.2	Sample results of generation	35
6.3	Shapenet Graph Variational Auto Encoder Results	36
6.4	Result of the model to generate some example from text input, good examples	37
6.5	Result of the model to generate some examples from text input, bad examples	37

Chapter 1

Introduction

The increasing adoption of 3D models can be observed across a wide number of fields, from entertainment and medical imaging, to virtual reality. Due to their versatility, 3D mesh models have emerged as the most used approach for representing these kind of objects. Composed of vertices, edges, and faces, 3D mesh models can be effectively represented as topological structures or as multi-graph configurations, offering adaptability and ease of manipulation in various applications. Concurrently, the growing utilization of generative AI for diverse media types, including text, music, images, slides, videos, and even websites, demonstrates a strong interest from both the public and corporate sectors in this area. Two recent breakthroughs in generative AI models for 2D images are DALL-E 2 and Stable Diffusion. DALL-E 2, developed by OpenAI, is an advanced version of the initial DALL-E, which was capable of generating novel, often surreal images from textual descriptions. DALL-E 2 built on this capability, providing even more coherent and high-quality 2D image outputs. The Stable Diffusion model takes text and images as inputs, employing CLIP encoding for the text and using UNet networks for diffusing and reconstructing the images. The unique strength of this model lies in its ability to generate coherent and high-quality outputs.

Given the advancements in both 3D models and generative AI, it is timely and pertinent to investigate the feasibility of employing generative models for 3D mesh structures. The research question of this thesis is: Can the principles of DALL-E 2 and Stable Diffusion be extended to generate 3D mesh structures from a graph representation? The implications of this question span various domains, from 3D modeling and rendering to more advanced applications in virtual reality and augmented reality.

In this paper, I employed a distinct methodology to explore the intersection of generative AI and 3D mesh models. Utilizing OpenAI's CLIP model, a sophisticated language-image multi-modal model, I engineered a unique bridge between textual labels and the corresponding graph representation of a 3D model. CLIP's ability to understand and connect images and text served as an effective instrument in our design, creating a semantic link that guides the generative process. Further, I introduced a generative model, conditioned on the CLIP model's output, which is capable of producing images that correspond to the desired 3D mesh structures. These results, although preliminary, hint at the immense potential of this approach and underscore the need for further exploration and fine-tuning.

In addition, this document incorporates an extensive survey on the broader field of data generation, offering a comprehensive look at the various methodologies, techniques, and applications involved. The survey provides a deep dive into generative models, their development over the years, and their impact across a multitude of domains. It aims to provide a holistic understanding of the landscape of data generation and the strides being made to expand its potential. This broad scope provides a robust foundation for our explorations in applying generative AI principles to 3D mesh structures, and underscores the significance of our preliminary results in this emerging area.

Chapter 2

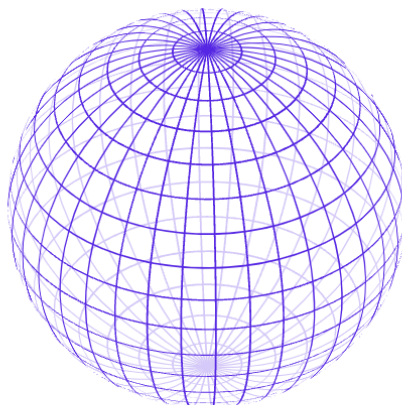
Digital Representation of 3D objects

This section provides a comprehensive introduction to the fundamental concepts of Manifolds, Simplicial Complexes, and meshes, establishing the necessary background for understanding the 3D objects employed in this thesis. Other 3D objects representations are also discussed (being a central part of visual representation in research), to have an overview of the full spectrum of the field.

2.1 Manifold

In our three-dimensional world, I are surrounded by a multitude of 3D objects every day. These objects can be considered as manifold structures Bronstein et al., 2016, that possess specific properties that make them more advantageous to work with compared to discrete objects. A manifold M is a topological space that locally resembles Euclidean space near each point. More precisely, each point in M has a neighborhood that is homeomorphic to an open subset of \mathbb{R}^n , where n is the dimension of the manifold. This characteristic of manifolds allows for the application of concepts and tools from calculus and differential geometry, enabling the exploration of transformations and dynamics in ways that are not feasible with discrete objects.

In 3D representation, I will deal with 2-manifold, manifold of dimension 2 that locally re-assembles 2 2-dimensional Euclidean plane around each point. Using the same description, every point in a 2-manifold has a neighborhood that is homeomorphic to an open subset of \mathbb{R}^2 . A first example of a 2-manifold could be the surface of a sphere^{2.1}



$$r = x^2 + y^2 + z^2$$

FIGURE 2.1: Sphere Manifold

Manifold structure is however quite strict. While offering many advantages, does impose certain constraints due to its inherent properties. One such limitation is that a manifold must be continuous, which means that not all objects or collections of objects can be considered as a single manifold. For instance, a disjoint collection of objects cannot be treated as a single continuous manifold because there is no continuous transformation between the separate components. In our three-dimensional world, as human, I may consider many objects as single instances and not a collection of many manifolds (for example a bike) Bronstein et al., 2016.

To describe the global structure of a manifold, I make use of transition maps and atlases. Transition maps provide a means to smoothly relate the local coordinate systems of overlapping charts, while an atlas is a collection of compatible charts that together cover the entire manifold Zomorodian, 2005. This concept allows us to represent complex structures, such as a bike, as a single manifold by combining local coordinate systems with the help of transition maps.

Digitally, it would be hard do define this infinite space without having memory issues. Representing manifolds as mathematical formulas, such as parametric or implicit equations, can offer precise descriptions, enabling smooth transformations and better analysis of manifold structures. Parametric representations use equations to describe the coordinates of each point on the manifold in terms of multiples parameters, which can provide a continuous representation and be easily manipulated mathematically Xu, Tong, and Stilla, 2021. However, this approach can be computationally intensive for complex manifolds and may not cover all manifolds with simple parametric representations. The choice of representation often depends on the specific problem domain and the desired balance between precision and computational efficiency. An example of this representation would be a circle of radius r centered at (h, k) in the Euclidean plane can be represented parametrically by $x(t) = h + r \cos t$ and $y(t) = k + r \sin t$ with parameter t varying from 0 to 2π , or implicitly by $(x - h)^2 + (y - k)^2 = r^2$ 2.2. Working with manifolds often leads to calculus problems that can benefit from the extensive body of research in this domain. However, addressing these problems may require the expertise of skilled theoretical researchers who possess a deep understanding of the underlying mathematical principles and techniques.

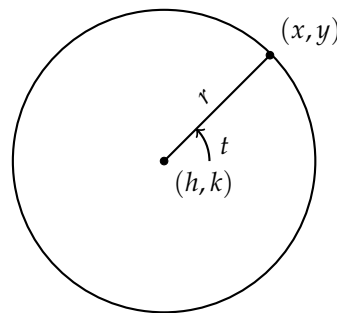


FIGURE 2.2: Parametric representation of a circle in the Euclidean plane

2.2 Simplicial Complexes

Considering these difficulties. Many approximations of these 3D objects has been considered to represent object digitally. The most popular method of manifold approximation in memory would be the use of simplicial complexes. A simplicial complex is a finite collection of simplices (vertices, edges, triangles, etc.2.3) that satisfy certain properties. These simplices can be combined to approximate the geometry and topology of a manifold.Carmo, n.d.

Simplicial complexes are particularly useful for approximating manifolds in computer-based applications because they allow for a discrete representation of the continuous manifold structureZomorodian, 2005. This representation can be used in various fields, such as computer graphics, computational geometry, and topological data analysis, to study and manipulate the underlying manifold objects efficiently.

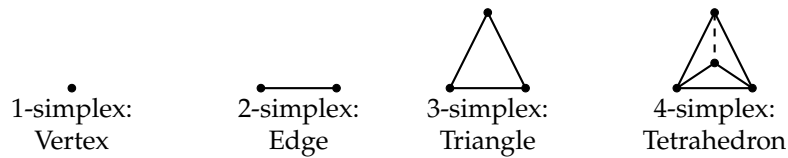


FIGURE 2.3: Simplicial complexes of dimension 1, 2, 3, and 4

2.3 Meshes

Meshes, which are derived from simplicial complexes, are widely used in computer graphics as a discrete representation of continuous manifold structures. A mesh is a collection of geometric elements such as vertices, edges, and faces (usually triangles or quadrilaterals) that are interconnected to form the shape and surface of a 3D object [2.4](#). The use of meshes enables efficient rendering, manipulation, and analysis of complex 3D objects in various applications, such as video games and animations.

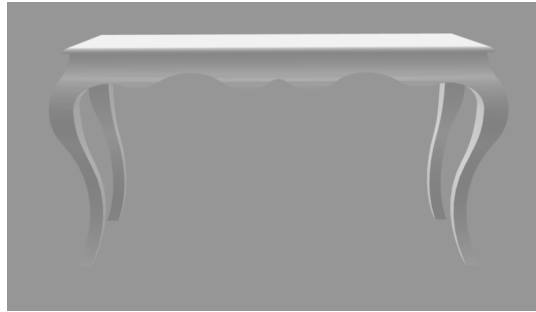


FIGURE 2.4: Mesh Representation of a table

Meshes are highly versatile and widely used in various fields, particularly in computer graphics and video games. Many algorithms have been developed to handle meshes and enable them to simulate real-life objects and environments with remarkable accuracy. While meshes may not perfectly replicate the manifold properties of real-life objects, this is not necessarily a problem as their discrete representations can be used effectively for simulation and rendering purposes. One such algorithm is tessellation, which enables the merging of simplices based on the render distance to form a continuous representation of the mesh. This technique allows for the efficient use of computational resources, ensuring that the mesh remains visually appealing and accurate to the viewer without overburdening the system. While tessellation does not perfectly preserve the manifold properties of the original object, it can still provide a practical and effective solution for representing 3D objects in digital form [Luebke et al., 2002](#), for example:

- **Efficient memory usage:** By reducing the number of nodes and edges in the graph, I can save memory resources, which is crucial for large-scale applications and real-time processing.
- **Visualization and interpretability:** A reduced graph can be more easily visualized and understood by humans, as it focuses on the most important structural components and relationships.
- **Noise reduction:** Pruning and graph reduction can help filter out noise in the data, enhancing the overall quality of the information represented in the graph.

Textures can be added to meshes to provide intrinsic information and enhance their visual appearance. Textures are images that are mapped onto the surface of a mesh to simulate the appearance of real-life materials, such as wood, metal, or fabric. These images can be generated from photographs or created digitally to achieve the desired effect. The process of adding textures to a mesh involves UV mapping, which is a technique for projecting the 2D image onto the 3D mesh.

The versatility and flexibility of meshes, coupled with their ability to incorporate various properties such as textures, is a key factor contributing to their widespread popularity in many fields.

2.4 Voxels

Voxels, or volumetric pixels, are another popular representation of 3D objects that differ from meshes. Unlike meshes, which represent objects using a collection of interconnected geometric elements, voxels represent objects as a grid of small cubic units called voxels. Each voxel contains information about the physical properties of the object, such as its color, texture, and density. This makes voxels particularly useful for applications such as medical imaging and scientific simulations Lorensen and Cline, 1987Fedorov et al., 2012Li et al., 2012, where the focus is on the internal properties of the object rather than its surface appearance. However, voxels are generally less computationally efficient than meshes, as they require more memory and processing power to represent the same object. As a result, the choice between voxels and meshes often depends on the specific requirements of the application, such as the need for high-resolution internal imaging or detailed surface rendering. 2.5

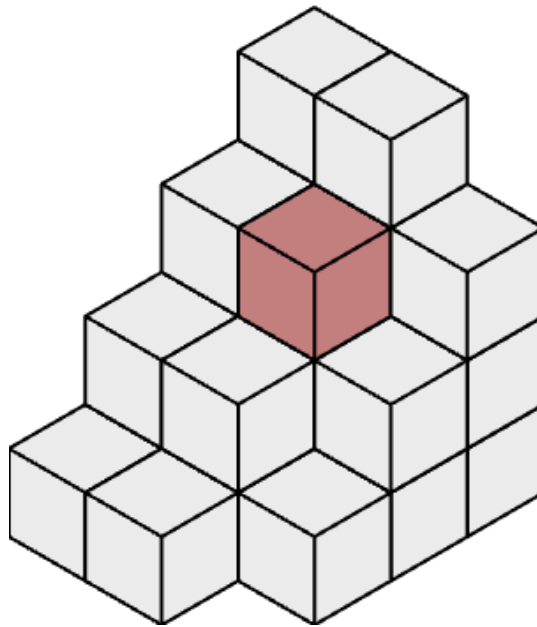


FIGURE 2.5: A 3D object made of smaller cubes (voxels) representing a voxel-based structure

2.5 Point Clouds

Point clouds are another popular representation of 3D objects that differ from meshes and voxels. A point cloud is a collection of points in 3D space that represent the surface or internal structure of an objectZhou, Zhang, and Foroosh, n.d. Unlike meshes, point clouds do not explicitly define the connectivity between the points, and therefore lack information about the object's topology. However, point clouds are particularly useful for applications such as 3D scanning and LidarZhou, Zhang, and Foroosh, n.d., where accurate and detailed representation of the object's surface or environment is required. Point clouds can also be used as an intermediate representation for generating meshes or voxelsMatthew Bolitho, n.d. One major advantage of point clouds is their relatively low memory usage, which makes them suitable for applications with limited computational resources. For example, photogrammetry Remondino and El-Hakim, n.d. is a technique that often involves mapping physical space using point clouds as a starting point, which can then be used to generate meshes or

other representations. However, point clouds also present challenges in terms of processing and analysis due to their lack of structure and topology. [2.6](#)



FIGURE 2.6: Point-cloud representation of face

2.6 Other Representations

In addition, there are several other representations of 3D objects that exist: Octrees [2.7](#) that divide the 3D space into smaller cubes allowing for hierarchical representation of the object's structure. Each of these representation has this pros and cons that will depends on the specific requirements of the application.

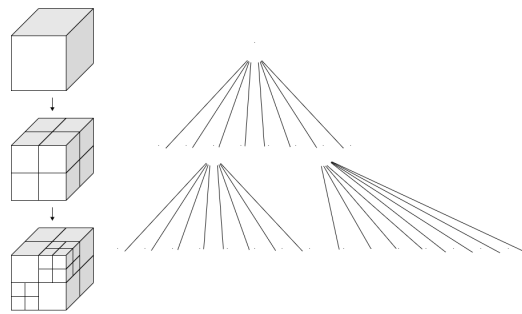


FIGURE 2.7: Octree

In this paper, I will primarily focus on meshes as a representation of 3D objects for several reasons. Firstly, meshes offer computationally efficient methods of representation and manipulation. Secondly, algorithms for simplification, such as tessellation (in the context of level of detail), are widely available and easy to use. Thirdly, meshes are a popular representation for 3D objects in various fields, including computer graphics and scientific simulations. Finally, while point clouds and voxels have been explored extensively in the context of machine learning tasks, meshes have received less attention. In the next section, I will introduce graph structures as a means of analyzing mesh representations of 3D objects.

In the next section, I will introduce the concept of graphs and multigraphs, which can be used to analyze the complex connectivity of mesh structures.

Chapter 3

Graph and Hypergraph

In the previous chapter, I introduced simplicial complexes and meshes as the most common representations of 3D objects. A simplicial complex is a collection of simplices. For this paper, I will define with more constraints that a simplicial complex is a finite set of simplices. In this way, I don't allowed multiple edges or faces between multiple vertices. In other term, I can define the following: $S = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, a simplicial complex S is a triple composed of a set of vertices V , a set of edges E and a set of faces F .

3.1 Graph: Definition

In graph theory, an undirected graph¹ is define as $G = (\mathcal{V}, \mathcal{E})$ with \mathcal{V} the set of vertices and \mathcal{E} the set of edges 3.1. For example, the following graph representation can be used to model complex structures that are not necessarily Euclidean in nature: $G = (\mathcal{V}, E)$ with $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ and $E = \{\{v_i, v_j\} \mid v_i, v_j \in \mathcal{V} \wedge i \neq j\}$. Graph can be used to represent a wide range of real-world problems, from social networks and transportation system to knowledge graph *Introducing the Knowledge Graph: things, not strings n.d.* Their non-Euclidean nature allows them to model relationships and interactions between entities in a more flexible and abstract way than traditional Euclidean structures.

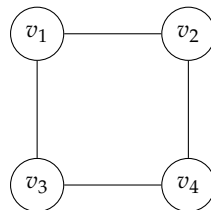


FIGURE 3.1: A simple undirected graph with four vertices and four edges

Graph Theory has been vastly studied in many fields of computer science. Other the years, graph theory has evolved into a vital area of mathematics and computer science with a multitude of problem resolutions:

- Connectivity: Studying how a graph is structured, searching for connected components and bridges in a graph (network).
- Graph traversals: Algorithm for searching in a graph such as depth-first search or Dijkstra's.
- Spectral graph theory: Analyzing the properties of graphs using linear algebra techniques.
- Many other methods: Community detection, graph generation, graph coloring...

In the digital realm, a common way to represent a graph is by using an adjacency matrix. This method provides a concise and efficient means of encoding the information about the connections between vertices in the graph.

¹I will now use the word graph to talk about undirected graph

An adjacency matrix is a square matrix of size $n \times n$, where n is the number of vertices in the graph. The entry at position (i, j) in the matrix indicates the presence or absence of an edge between vertex i and vertex j . For an undirected graph, the adjacency matrix is symmetric, meaning that the matrix is equal to its transpose.

For an undirected graph $G = (V, E)$ with vertices $V = v_1, v_2, \dots, v_n$, the adjacency matrix A is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

The adjacency matrix representation offers several advantages, such as constant-time access to the information about the presence or absence of an edge between any two vertices. However, it also has some drawbacks, especially when dealing with sparse graphs, as it requires $O(n^2)$ space, which may be inefficient for graphs with relatively few edges. Alternative representations, such as adjacency lists, can be more efficient in these cases.

The main limitation of graph is probably the fact that it strictly allowed binary relation. In the next section, I introduce Hypergraph that remove this restriction.

3.2 Hypergraph

A hypergraph is an extension of the traditional graph concept, allowing for a more generalized and flexible representation of relationships between entities. In a standard graph, an edge connects exactly two vertices, representing a pairwise relationship (binary relations). In contrast, a hypergraph allows edges, called hyperedges, to connect any number of vertices, representing more complex relationships (n-ary relations) among multiple entities.

Formally, a hypergraph H 3.2 is defined as a pair (V, E) , where V is a finite set of vertices and E is a set of hyperedges. Each hyperedge $e \in E$ is a non-empty subset of V . Unlike standard graphs, there is no restriction on the cardinality of a hyperedge, which means it can connect two or more vertices.

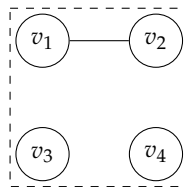


FIGURE 3.2: A simple hypergraph with four vertices, one edge and one hyperedge

Several concepts and algorithms from traditional graph theory have been extended and adapted for hypergraphs, including connectivity, coloring, and traversals. However, due to the increased complexity of hypergraphs, many problems become more challenging to solve in this generalized setting.

The adjacency matrix concept can be extended for hypergraphs using higher-dimensional structures such as hypercubes or tensors. For a hypergraph $H = (V, E)$ with n vertices, I can define an n -dimensional hypercube or tensor T to represent the relationships between vertices within the hyperedges.

Each element of the tensor T_{i_1, i_2, \dots, i_n} is assigned a value of 1 if the vertices $v_{i_1}, v_{i_2}, \dots, v_{i_n}$ form a hyperedge in H , and 0 otherwise. This representation allows for the encoding of complex relationships involving multiple vertices simultaneously.

However, it is important to note that this tensor-based representation of hypergraphs suffers from the curse of dimensionality, as the size of the tensor grows exponentially with the number of vertices. As a result, it becomes impractical to use this representation for relations that exceed 3 vertices due to the rapidly increasing memory requirements and computational complexity.

In such cases, alternative representations and data structures, such as incidence matrices or edge lists, may be more appropriate for efficiently representing and processing hypergraphs.

As seen previously, Hypergraph and Simplicial complex are two different structures. However, their topology and properties shares common concepts. In fact, hypergraphs and simplicial complexes can both be used to represent 3D objects, and it is possible to create a lossless bijection between them, preserving all information during the transformation process. In the next section, I will explore the unique characteristics of hypergraphs and simplicial complexes when applied to 3D objects and examine the ways to establish a lossless bijection between the two representations.

3.3 Mesh to Hypergraph for 3D Object Representation

The representation of 3D objects in computer files can be achieved through various formats, with the .obj format being one of the most prevalent. In this section, I delve into the unique features of hypergraphs that make them suitable for representing 3D objects and their topological properties.

3.3.1 The .obj File Format

First, to define the transformation from .obj file to simplicial complex, I need to define what is an .obj file formally.

An .obj file represents an object with absolute positions. It contains information about the vertices, texture coordinates, and vertex normals of a 3D object. The vertices are defined as 3D coordinates, the texture coordinates are 2D coordinates, and the vertex normals are 3D vectors. In addition to these, the .obj file format also specifies the faces of the object, which are formed by connecting vertices together. A face can be described as a polygon that is defined by an ordered list of three or more vertices.

The following is a brief overview of the main components of an .obj file:

- **Vertices:** These are defined using the `v` keyword followed by the X, Y, and Z coordinates of the vertex. For example, `v 1.0 2.0 3.0` would define a vertex with coordinates (1.0, 2.0, 3.0). Mathematically, I can represent the set of vertices as $M_{\mathcal{V}}$, where $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$.
- **Texture coordinates:** These are defined using the `vt` keyword followed by the U and V coordinates of the texture. For example, `vt 0.5 0.5` would define a texture coordinate with coordinates (0.5, 0.5). Mathematically, I can represent the set of texture coordinates as $M_{\mathcal{T}}$, where $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$.
- **Vertex normals:** These are defined using the `vn` keyword followed by the X, Y, and Z components of the normal vector. For example, `vn 0.0 1.0 0.0` would define a vertex normal pointing in the positive Y direction. Mathematically, I can represent the set of vertex normals as $M_{\mathcal{N}}$, where $\mathcal{N} = \{n_1, n_2, \dots, n_n\}$.
- **Faces:** These are defined using the `f` keyword followed by the indices of the vertices, texture coordinates, and vertex normals that form the face. For example, `f 1/1/1 2/2/1 3/3/1` would define a face formed by the vertices with indices 1, 2, and 3, texture coordinates with indices 1, 2, and 3, and vertex normals with index 1. Mathematically, I can represent the set of faces as $M_{\mathcal{F}}$, where $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$.

Having defined the structure of an .obj file, I can now explore the process of transforming this representation into an hypergraph.

3.3.2 Transforming .obj Files to Hypergraphs

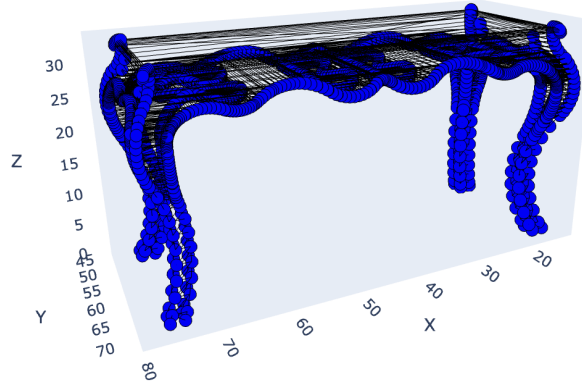


FIGURE 3.3: Mesh of a table 2.4 transform into a Graph Object

To represent a mesh file as a graph 3.3, I have several options, which could be either lossless or lossy. First, I need to decide which features from the mesh file will be used as graph vertices, edges, etc. Some possible approaches include:

- Mapping the vectors from vertices, texture coordinates, and vertex normals to G_V (graph vertices) and splitting faces into three separate edges. Mathematically, this can be defined as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = M_V \cup M_T \cup M_N$, and $\mathcal{E} = \{e | e \subset \mathcal{V} \wedge |e| = 2\}$.
- Computing the distance matrix between points and adding features of distances between edges, while ignoring vertex positions. Let D be the distance matrix between points in M_V . I can define the graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = M_V$, and $\mathcal{E} = \{e | e \subset \mathcal{V} \wedge |e| = 2 \wedge d(e) \in D\}$.
- Adding information about face normals and incorporating it into the hypergraph. Let F_N be the set of face normals. I can define the hypergraph as $H = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = M_V \cup F_N$, and $\mathcal{E} = \{e | e \subset \mathcal{V} \wedge |e| \geq 2\}$.
- Using only relative positions of vertices. Let R_V be the set of relative vertex positions. I can define the graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = R_V$, and $\mathcal{E} = \{e | e \subset \mathcal{V} \wedge |e| = 2\}$.

These approaches allow for different levels of detail and complexity when transforming a mesh file into a graph or hypergraph representation.

In the next section, I will now delve into the state of the art regarding the graph generation and data generation in general.

Chapter 4

Literature Review: Graph Generation

Comment: This paper is complementary to the following slides. https://docs.google.com/presentation/d/1f4hcvEmzsDWrFlVHd_kteLGmQHmnJ1y8HiYpUDU1Lvg/ They follow a similar but different structure and some paper or part are more details on the slides.

The development of generative models capable of creating realistic and coherent data has garnered significant attention in recent years, particularly with the success of models like GPT-3 Brown et al., 2020, Stable Diffusion AI, 2022, and DALL-E 2 OpenAI, 2022. These models have demonstrated remarkable capabilities in generating high-quality outputs for various data types, such as text and 2D images. Extending the principles of these generative models to handle 3D mesh structures and graph representations presents an intriguing research question, with potential implications across various domains, such as computer graphics, virtual reality...

To address this question, it is essential to have a comprehensive understanding of the current state of the art in graph and mesh generation, as well as the techniques employed in graph and mesh embedding and data generation tasks. Graph and mesh embeddings play a crucial role in this context, as they provide a means to represent complex 3D structures and relationships in a lower-dimensional space, which can then be fed into generative models for synthesis. Inspired by recent advancements in generative models, this literature review aims to provide an overview of the most relevant research in the fields of graph and mesh generation. The focus is on the methods and models that could inform the development of a generative model for 3D mesh structures from graph representations.

The chapter is structured into sections that discuss graph and mesh embedding, text embedding (necessary for generating graphs from text inputs), and data generation tasks for a variety of data types, such as 2D images, 3D voxels, point clouds, and meshes. A particular focus will be on the intersection of joint-embedding techniques and contrastive learning as exemplified by models like CLIP Radford et al., 2021. CLIP represents a significant advancement in multimodal learning, bridging the gap between text and images. This opens the door to promising research avenues, exploring how its principles can be applied to other forms of data like graphs and 3D mesh structures. By delving into these areas, I aim to acquire insights into the existing techniques and strategies that can be adapted or extended to conceive a generative model for 3D mesh structures grounded in graph representations.

4.1 Challenges

Before discussing the generation of graphs from textual input (e.g., labeled data), it is essential to consider the challenges associated with the development of standard generative models for graph-to-graph transformations Zhu et al., 2022 Li et al., 2018b:

- **Non-Euclidean aspect:** Graphs are non-Euclidean structures, making it challenging to directly apply algorithms designed for Euclidean data, such as images represented by grids, to graph data. This necessitates the development of specialized models and techniques to effectively process and generate graph data. This issue can be somewhat mitigated in our context, as I am primarily dealing with 3D meshes. These mesh structures, despite being represented as graphs, inherently relate to Euclidean space given

their geometric nature Bronstein et al., 2016. As a result, certain techniques developed for Euclidean data might be more readily applicable or adaptable to 3D mesh data Monti et al., 2016.

- **Large output spaces:** Graph generation models need to handle large output spaces, especially when dealing with hypergraphs. This can increase the computational complexity and memory requirements of the models. Li et al., 2018b
- **Scalability:** As the size and complexity of graphs increase, generative models must be able to scale accordingly to handle larger input sizes while maintaining reasonable computational efficiency. Li et al., 2018b
- **Multirelationality of graphs:** Graphs often contain multiple types of relationships between nodes, which adds complexity to the generation process and requires models to accurately capture and reproduce these relationships. Li et al., 2018b
- **Black Box aspect:** Deep learning-based generative models are often considered "black boxes" due to their complex architectures and lack of interpretability, which makes it difficult to understand the underlying processes that generate the desired output. Li et al., 2018b

This number of challenges has led to the development of many graph generation algorithms Zhu et al., 2022. In their paper, the authors present a comprehensive review of the existing deep graph generation literature, covering a wide range of emerging methods and application areas. The paper proposes the following classifications for graph generation methods :

- **Variational autoencoders** Kingma and Welling, 2013 Kipf and Welling, 2016c Tan et al., 2017
- **Generative adversarial networks** Goodfellow et al., 2014 Wang et al., 2017a
- **Diffusion models** Liu et al., 2023

The three topics will be described later in this chapter in the graph generation subsection.

4.2 Embedding

Embedding techniques have become an essential aspect of machine learning and deep learning applications. These methods aim to represent high-dimensional, complex data in a lower-dimensional space, which can significantly improve the performance of various tasks, such as classification, clustering, and visualization. By capturing the inherent relationships and structures within the data, embeddings provide a more manageable and meaningful representation that can be effectively used by machine learning algorithms. In this section, I will explore different embedding techniques and their applications on different data types.

4.2.1 Text Embedding

Text embedding is a crucial aspect of this research, as the generation of 3D mesh structures from graph representations requires the effective conversion of textual inputs into numerical representations that can be processed by generative models. Text embedding techniques have evolved over the years, with traditional methods giving way to deep learning-based approaches.

Traditional text embedding methods, such as Bag-of-Words (BoW) Mikolov et al., n.d. and Term Frequency-Inverse Document Frequency (TF-IDF) Salton and Buckley, 1988, represent text as vectors based on the frequency of the words. However, these methods can fail to capture the semantic relationships between words and suffer from high dimensionality and sparsity issues Li et al., 2020. Limitation of word frequency can be described by their indifference to word order and their inability to represent idiomatic phrases. For example

the semantic representation of "Air" and "Canada" are hard to combine to form the semantic representation of "Air Canada" Mikolov et al., n.d. The Bag-of-Concepts (BoC) model Li et al., 2020 enriches text representation by incorporating semantic concepts from external knowledge bases, addressing the limitations of traditional methods like BoW in capturing semantic relationships and reducing previously mentioned problems like high dimensionality or sparsity.

In response to these limitations, distributed representations of words emerged, with the introduction of word2vec Mikolov et al., n.d. and doc2vec Le and Mikolov, 2014. Word2vec can usually rely on two algorithms, Continuous Bag-of-Words (CBOW) and Skip-gram Mikolov et al., n.d., which learn dense, low-dimensional word embeddings capable of capturing semantic and syntactic relationships. Doc2vec, an extension of word2vec, generates document-level embeddings by incorporating paragraph vectors. CBoW has been improved multiple times Wang et al., 2017b during the past and now can be considered now as a standard in NLP.

Further advancements in text embedding led to the development of contextualized word embeddings, such as ELMo Peters et al., 2018, which leverages bidirectional LSTMs to generate context-dependent embeddings. BERT Devlin et al., 2018 and GPT Brown et al., 2020 are two other prominent contextualized word embedding models, with BERT utilizing masked language modeling and GPT employing a unidirectional transformer architecture. Many extension of these models have been releases during the last 10 years Liu et al., 2019 Sanh et al., 2019 Lan et al., 2019

More recently, large-scale language models (LLMs) like GPT-3 Brown et al., 2020, GPT-4OpenAI, 2023, BARD AI, n.d.(a), PaLM AI, n.d.(b) and LLaMA AI, n.d.(c) have demonstrated impressive capabilities in text generation by leveraging the power of transformers Vaswani et al., 2017 and unsupervised pretraining. These models have laid the groundwork for the proposed approach to generate 3D mesh structures from graph representations.

In conclusion, text embedding methods have evolved from traditional, sparse representations to dense, deep learning-based embeddings, and now to large-scale language models. This research aims to leverage these advancements to create a generative model for 3D mesh structures using Graph Neural Networks, adapted diffusion methods, and therefore state of the art text embedding.

4.2.2 Geometric 3D object Embedding

3D object embedding involves the transformation of 3D objects, typically represented as point clouds, meshes, or volumetric data, into lower-dimensional continuous vector representations. These compact representations facilitate efficient processing and analysis of 3D data by machine learning algorithms for tasks such as object recognition, segmentation, and generation.

Point clouds and Voxels

Point clouds and voxels are two common representations of 3D data that have been extensively studied for their embedding definition, without going into too much detail in this discussion. Both representations offer unique advantages and challenges when it comes to defining embeddings and applying machine learning techniques.

Various approaches have been developed to define embeddings for point cloud data, with a primary focus on capturing local and global geometric information. Notable techniques include PointNet Qi et al., 2016 and PointNet++Qi et al., 2017, which have demonstrated success in tasks such as object recognition, segmentation, and generation. PointNet++ present itself has a net improvement compare to PointNet and other model.

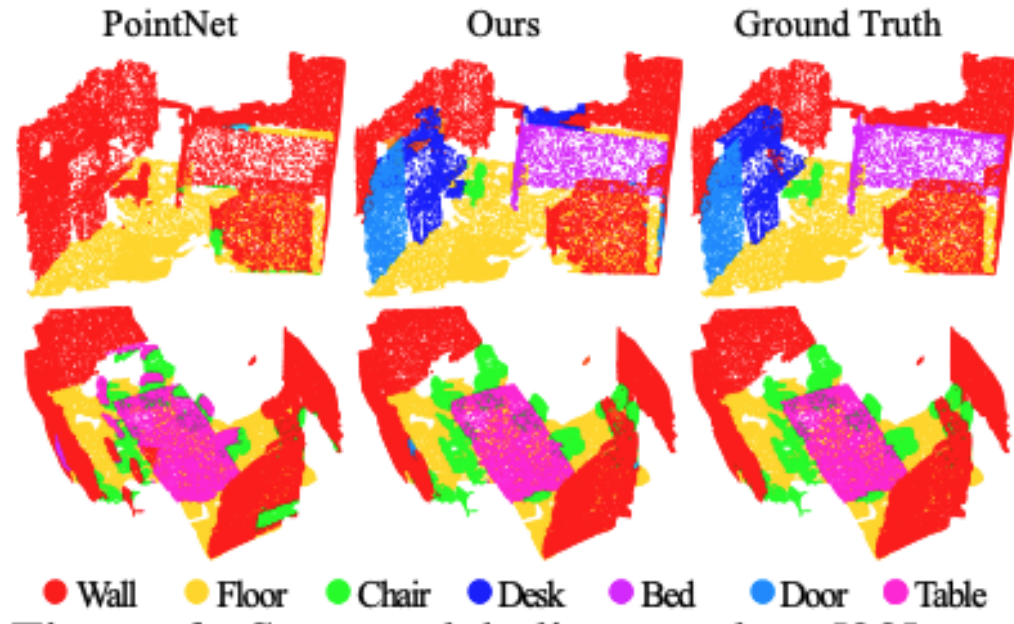


FIGURE 4.1: PointNet++ Qi et al., 2017 ("ours") compared to PointNet Qi et al., 2016 and the ground truth

Voxel representations discretize the continuous 3D space into a fixed grid, making it easier to apply conventional machine learning techniques, such as convolutional neural networks (CNNs) O'Shea and Nash, 2015. A number of methods have been proposed to define embeddings for voxel data, including 3D CNNs Tran et al., 2017, which extend the concepts of 2D CNNs to handle the added dimensionality. The original purpose of 3D CNN is to embedded videos but this model has been applied multiple times to voxels Qi et al., n.d. PapersWithCode, n.d.

Both point clouds and voxels have been widely studied for their embedding definition, and various techniques have been developed to capture the complex geometric and semantic information contained within these 3D representations. Each representation comes with its own set of advantages and challenges, and the choice of representation often depends on the specific application and the availability of data.

Mesh

3D objects can be considered as manifolds, which are topological spaces locally resembling Euclidean spaces. Manifold learning techniques 2.2. *Manifold learning — scikit-learn 1.2.2 documentation n.d.*, such as Isomap Tenenbaum, Silva, and Langford, 2000, Locally Linear Embedding Morris et al., 2011, Low-Rank Isomap Mehrbani, Mohammad, and Kahaei, 2021 and Laplacian Eigenmaps Belkin and Niyogi, 2003, have been used to find low-dimensional embeddings of high-dimensional data that preserve the intrinsic geometric structure of the objects. Some researcher also tried to tackle the problem considering these structures as simplicial complexes Hajij et al., 2021. However, these methods usually lack the ability to capture complex geometric structures and semantic information.

Laplacian Eigenmaps Belkin and Niyogi, 2003 is a widely-used nonlinear dimensionality reduction technique that aims to preserve local geometric structures of the data. Given a set of data points x_1, x_2, \dots, x_n in a high-dimensional space, the goal is to find a low-dimensional embedding y_1, y_2, \dots, y_n that preserves the local relationships among the data points. The steps to compute the Laplacian eigenmaps are as follows:

- **Construct a similarity graph:** Create an undirected graph with the data points as vertices. Two vertices x_i and x_j are connected by an edge if they are considered "close" Belkin and Niyogi, 2003. Either in the ϵ -neighborhoods ($\|x_i - x_j\| < \epsilon$ with $\epsilon \in \mathbb{R}$) or n nearest neighbors (with $n \in \mathbb{N}$). The relationship being symmetric, the adjacency

matrix will also be symmetric. The weight can be either binary (1 if connected, 0 otherwise) or using the commonly used Heat Kernels formula:

$$W_{ij} = \begin{cases} e^{-\frac{\|x_i - x_j\|^2}{t}}, & \text{if } \|x_i - x_j\| < \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

- **Create the Embedding** The graph Laplacian L is computed as $L = D - W$, where D is the degree matrix (a diagonal matrix with $d_{ii} = \sum_j w_{ij}$) and W is the adjacency matrix containing the edge weights w_{ij} . Find the eigenvectors corresponding to the smallest non-zero eigenvalues of the generalized eigenvalue problem: $Ly = \lambda Dy$. Typically, the first k eigenvectors are used to construct the low-dimensional embedding, where k is the desired dimensionality of the reduced space. The low-dimensional coordinates y_i for each data point x_i can be formed by concatenating the corresponding components of the first k eigenvectors.

The Laplacian eigenmaps method, which uses the heat kernel to define the weights of the graph, can be linked to Graph Convolutional Networks (GCNs) Kipf and Welling, 2016a. GCNs are a class of deep learning models specifically designed for handling graph-structured data. They perform convolution-like operations on graph-structured data, such as 3D object embeddings, to learn local and global patterns in the data. This leads us to the next section, **Graph Embedding**

4.2.3 Graph Embedding

Graph embedding concentrates on devising techniques to represent graph-structured data as low-dimensional vectors. These vector representations facilitate the application of machine learning algorithms to graph data, enabling tasks such as node classification, link prediction, graph clustering, and graph generation.

Various graph embedding techniques have been proposed in the literature, including spectral methods, matrix factorization approaches, and random walk-based methods. Spectral methods, such as Laplacian Eigenmaps Belkin and Niyogi, 2003 (as mentioned in the previous section about mesh embedding), leverage graph Laplacian properties to generate low-dimensional embeddings. Matrix factorization approaches, like GraRep Cao, Lu, and Xu, 2015, factorize graph-related matrices to obtain node embeddings. Random walk-based methods, such as DeepWalk Perozzi, Al-Rfou, and Skiena, 2014 and node2vec Grover and Leskovec, 2016, use random walks to capture local and global graph structures and apply the skip-gram model Mikolov et al., n.d. to learn embeddings.

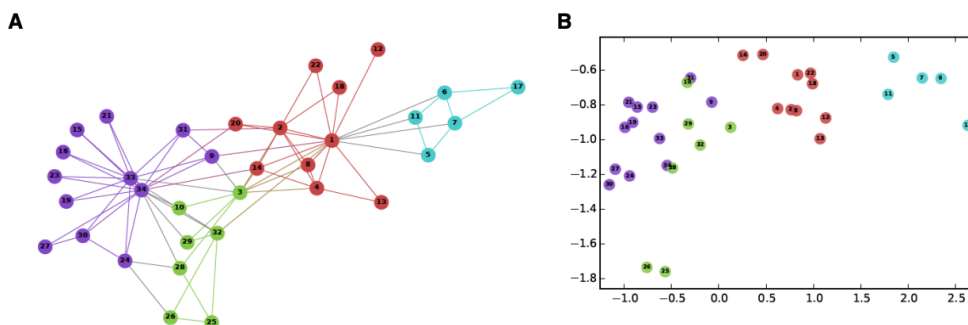


FIGURE 4.2: A: Zachary Karate Club Graph, B: Zachary Karate Club Embedding in 2D (with ACP reduction) Source: Guo and Zhao, 2020

Graph Neural Networks (GNNs) Scarselli et al., 2009 were first introduced as a class of deep learning models specifically designed to handle graph-structured data. GNNs have the ability to process and learn from graph data by leveraging the inherent relationships between nodes and edges in a graph. Since their introduction, GNNs have become a widely

studied area in machine learning, with numerous architectures and methods being proposed to address various graph-based tasks.

Current GNN architectures can be broadly categorized into three main classes: Graph Convolutional Networks (GCNs) Kipf and Welling, 2016b, Graph Attention Networks (GATs) Veličković et al., 2017, and GraphSAGE Hamilton, Ying, and Leskovec, 2017, among others.

Graph Convolutional Networks (GCNs) Kipf and Welling, 2016b employ a first-order approximation of spectral graph convolutions, an approach inspired by the Fourier transform's ability to convert a signal from the time domain to the frequency domain. These spectral convolutions take into account the adjacency and degree matrices of a graph, allowing GCNs to capture local neighborhood information.

The GCN model operates by propagating information between neighboring nodes, resulting in each node being represented by an embedding that captures its local graph structure. This procedure is often compared to the convolutional operation in Convolutional Neural Networks (CNNs), but applied to graph data.

However, the localized nature of GCNs can be both a strength and a weakness. On the one hand, it allows the model to capture local relationships and structures effectively. On the other hand, it also limits the model's ability to handle global graph information and structures, and it struggles with larger graphs due to the increased computational complexity.

Graph Attention Networks (GATs) Veličković et al., 2017 introduced attention mechanisms to the domain of GNNs. Like GCNs, GATs also operate based on a node's local neighborhood. However, instead of giving equal importance to all neighbors, GATs use attention Vaswani et al., 2017 mechanisms to assign different weights to different nodes based on their relationships.

This mechanism allows GATs to capture more nuanced, context-dependent information, making them highly flexible and adaptable. They can model a wide range of relationships and are particularly suitable for graphs with complex, non-homogeneous structures.

However, depending on the type of attention computation, a general drawback of GATs lies in their computational cost. Computing attention weights for each node and its neighbors can be resource-intensive, making GATs less suitable for large-scale graph data.

GraphSAGE Hamilton, Ying, and Leskovec, 2017 is a general inductive framework that generates embeddings for previously unseen data. It does this by sampling and aggregating features from a node's local neighborhood. Unlike GCNs and GATs, which are transductive and require access to the entire graph structure, GraphSAGE is capable of generating embeddings for new nodes or graphs during inference.

While this inductive approach makes GraphSAGE versatile and adaptable to unseen data, it does come with its limitations. In particular, GraphSAGE might not capture higher-order relationships or global graph properties effectively, as it primarily focuses on local neighborhood information.

A comparative summary of these three models is provided in Table 4.1. Note that the "Performance" column in this table is an oversimplification. The actual performance of these models can vary significantly depending on the specific task, the graph structure, and the nature of the node and edge features.

TABLE 4.1: Comparison of GNN Architectures

Model	Limitations	Performance
GCN	Struggles with global structures	Good for smaller graphs
GAT	High computational cost	Excellent for complex structures
GraphSAGE	Struggles with global properties	Versatile across multiple tasks height

In addition to these primary architectures, there have been numerous other GNN variants proposed for specific applications, such as Graph Isomorphism Networks (GINs) Xu et al., 2018 for graph classification, and Dynamic Graph Convolutional Networks (DGCNs) Wang et al., 2018 for dealing with dynamic graphs where the topology changes over time.

Finally, it's important to highlight our initial hypothesis that Graph Convolutional Networks (GCNs) might be particularly well-suited for our task of generating 3D mesh structures from graph representations. Our reasoning stems from the spectral nature of GCNs, which relies on the spectral domain for graph convolution operations. This mechanism

leverages the properties of the graph Laplacian, an approach that has parallels with Laplacian Eigenmaps, a technique I know to perform well on geometric structures. Consequently, I anticipate that the spectral approach of GCNs could be beneficial when working with meshes, as they are essentially geometric structures represented as graphs. I aim to validate or refute this hypothesis with our empirical results in the later sections of this work.

4.2.4 Embedding Evaluation

Evaluating the quality of graph and mesh embeddings is a crucial aspect of developing generative models for 3D mesh structures. Various evaluation methods have been proposed to assess the performance of embeddings with respect to their ability to capture structural and relational properties of the original data. This subsection is essentially based on Guo and Zhao, [n.d.](#) and Hailu et al., [2020](#)

Intrinsic Evaluation

Intrinsic evaluation methods focus on measuring the extent to which embeddings preserve the inherent properties of the graph, such as local and global structure, and the relationships between entities.

- **Graph Reconstruction:** Assess the ability of the embeddings to reconstruct the original graph, by measuring the similarity between the adjacency matrix of the original graph and that of the reconstructed graph.
- **Preservation of Distances:** Measure the correlation between pairwise distances in the original graph and the embedding space, to assess the preservation of global structure.
- **Graph-based Metrics:** Evaluate embeddings using graph-specific metrics such as clustering coefficient, modularity, and centrality measures, to examine how well the embeddings capture the structural properties of the original graph.

Extrinsic Evaluation

Extrinsic evaluation methods involve using the embeddings as input for downstream tasks, assessing the performance of the embeddings in the context of specific applications, most of these techniques produces accuracy that

- **Node Classification:** Assess the performance of the embeddings in classifying nodes based on their attributes or roles within the graph, which can be an indicator of how well the embeddings capture local information.
- **Link Prediction:** Evaluate the embeddings' ability to predict missing or future links in the graph, which can be used to test the embeddings' capacity to capture relational information.

It is important to notice that for Node Classification or Link Prediction, the used classifier has an impact on the accuracy. It is necessary to try multiple classifier to maximise the accuracy.

4.3 Data Generation Task

This section provides an overview of the state of the art in generative models for a variety of data types, including 2D images, 3D voxels, 3D point clouds, and 3D meshes. The goal is to understand the characteristics and advantages of working with different data types compared to graph analysis. A comprehensive understanding of this section could lead to better insights for generating and processing data.

In this section, I will focus on data generation within the same data type (e.g., image-to-image). The following section will discuss generating different data types from textual information and existing methods. Text generation will not be covered here.

4.3.1 Euclidean Data Generation

Euclidean data generation typically deals with data that resides in a regular grid structure, such as pixels in 2D images or voxels in 3D volumes. Generative models like Generative Adversarial Networks (GANs) Goodfellow et al., 2014 and Variational Autoencoders Kingma and Welling, 2013 have shown remarkable success in generating realistic and high-quality images.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) Goodfellow et al., 2014 are a class of generative models introduced by Goodfellow et al. in 2014. They consist of two neural networks trained in competition: the generator, which creates synthetic data instances, and the discriminator, which attempts to differentiate between real and synthetic instances. The generator and discriminator are typically trained in an adversarial game, where the generator aims to fool the discriminator into believing that the generated instances are real, while the discriminator tries to correctly classify instances as real or synthetic.

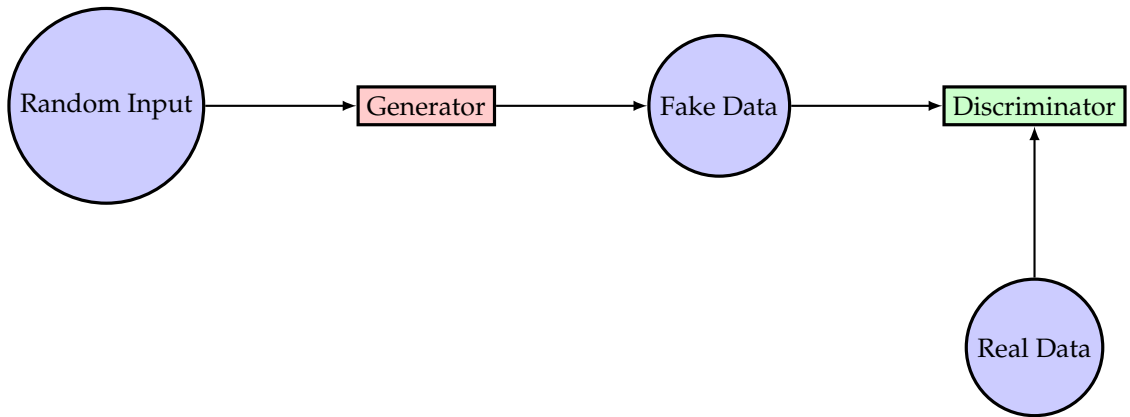


FIGURE 4.3: Diagram of a Generative Adversarial Network (GAN). The Generator (red) produces Fake Data (blue) from a Random Input (blue). The Discriminator (green) receives either Real Data (blue) or Fake Data (blue) and tries to distinguish between them.

To illustrate how the GAN training process works, let's first discuss the loss function for GANs. The training process involves optimizing a min-max game between the generator and discriminator. The generator aims to produce data that the discriminator cannot distinguish from real data, and the discriminator aims to correctly classify real and fake data. This process can be expressed by the following objective function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (4.2)$$

In this equation, G is the generator, D is the discriminator, \mathbf{x} represents real data samples from the true data distribution $p_{\text{data}}(\mathbf{x})$, and \mathbf{z} is the noise input to the generator, coming from a noise distribution $p_{\mathbf{z}}(\mathbf{z})$.

The first term of the equation calculates the expected log-likelihood of the discriminator correctly classifying real data samples. The second term calculates the expected log-likelihood of the discriminator correctly identifying fake samples.

By optimizing this objective function, the discriminator and the generator are trained simultaneously: the discriminator is trained to maximize its ability to discriminate real data from fake, and the generator is trained to minimize the discriminator's ability to identify its generated data as fake. Thus, the generator improves its ability to generate realistic data, while the discriminator becomes better at distinguishing between real and generated data. This adversarial training process continues until a Nash equilibrium Nash, n.d. is reached where the generator produces data indistinguishable from the real data, and the discriminator cannot distinguish real data from generated data, outputting a probability of 0.5 for all inputs.

GANs have shown impressive results in generating realistic images and have been widely used in various fields such as image synthesis, super-resolution, and image-to-image translation. The foundational work of GANs has inspired numerous variants and improvements, including but not limited to Conditional GANs (cGANs) Mirza and Osindero, 2014, Deep Convolutional GANs (DCGANs) Radford, Metz, and Chintala, 2015, and Wasserstein GANs (WGANs) Arjovsky, Chintala, and Bottou, 2017, each with their unique strengths and applications.

However, training GANs can be challenging due to issues like mode collapse [Common Problems | Machine Learning | Google for Developers n.d.](#), where the generator learns to produce a limited variety of samples, and the difficulty of balancing the training of the generator and discriminator to avoid one overpowering the other.

Diffusion

Diffusion models, a class of generative models, are built upon the principle of stochastic processes known as diffusion processes. In these models, data is imagined to undergo a Markovian diffusion process, which transforms it from a simple noise distribution to the complex data distribution I wish to model.

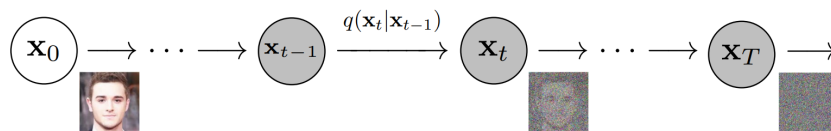


FIGURE 4.4: Diffusion Process

A simple form of a diffusion process can be expressed as follows:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \sqrt{2D}\mathbf{N}(0, 1) + \frac{1}{2}b(\mathbf{x}_t)^2 \quad (4.3)$$

where \mathbf{x}_{t+1} and \mathbf{x}_t are the states of the process at time steps $t + 1$ and t , respectively, D is the diffusion coefficient, and $\mathbf{N}(0, 1)$ is a standard normal distribution. The term $b(\mathbf{x}_t)$ denotes a drift function that determines the direction and speed of the diffusion process at each time step.

Originally, diffusion models were not efficient enough to compete with other generative models like GANs or VAEs, mainly due to their slow mixing times. However, recent advancements like Denoising Diffusion Probabilistic Models (DDPMs) Ho, Jain, and Abbeel, 2020 and Stable Diffusion Models Stability.ai, n.d. that diffused on embedding instead of the original data have significantly improved the generation quality and speed of diffusion models, making them a competitive choice for generating high-quality samples in various domains.

Variational AutoEncoder

Variational Autoencoders (VAEs) Kingma and Welling, 2013 4.5 are another class of generative models that have been widely used in various applications. They belong to the family of latent variable models and are particularly known for their ability to learn the underlying data distribution in an unsupervised manner.

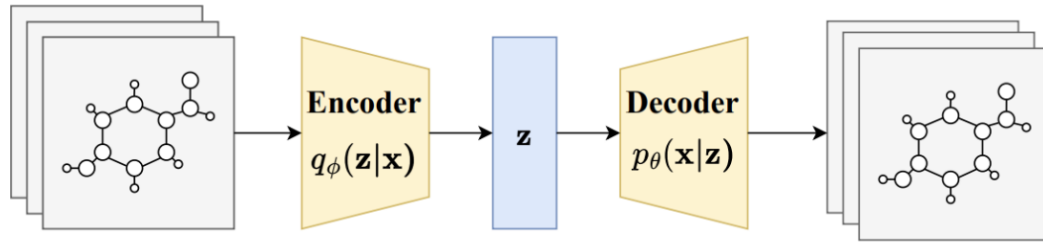


FIGURE 4.5: Variational Autoencoder with graph data

VAEs consist of two main components: an encoder and a decoder. The encoder network maps the input data to a latent space, and the decoder network maps points in the latent space back to the data space. VAEs impose a specific structure on the latent space, typically assuming that the latent variables follow a multivariate Gaussian distribution.

The objective function of VAEs consists of two parts: a reconstruction loss that encourages the decoded samples to resemble the original data, and a regularization term that encourages the latent variables to follow the assumed distribution. The VAE objective can be written as follows:

$$\mathcal{L}_{\text{VAE}}(\theta, \phi; \mathbf{x}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p_{\theta}(\mathbf{x}|\mathbf{z})] - \text{KL}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \quad (4.4)$$

where $q_{\phi}(\mathbf{z}|\mathbf{x})$ is the encoding distribution, $p_{\theta}(\mathbf{x}|\mathbf{z})$ is the decoding distribution, and $\text{KL}(\cdot||\cdot)$ denotes the Kullback-Leibler divergence. The Kullback-Leibler (KL) divergence is a measure of how one probability distribution diverges from a second, expected probability distribution.

Mathematically, the KL divergence of two continuous probability distributions is defined as:

$$D_{\text{KL}}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \quad (4.5)$$

This formula is of course declined for discrete probability with a sum.

The first term in the objective encourages the model to reconstruct the input data well, while the second term encourages the latent space to follow a predefined distribution, typically a standard normal distribution.

VAEs have several advantages such as being able to perform both generation and inference tasks, producing smooth latent spaces, and having a sound theoretical foundation. However, VAEs are also known for producing blurrier samples compared to GANs and might require more complex architectures or additional techniques to generate high-quality samples.

The following table compare the 3 models:

	GAN	Diffusion	Variational Autoencoder
Training Complexity	Resource-intensive	Medium resource demand	Lower resource demand
Result Quality	High fidelity, sharp details	High fidelity, less sharp details	Lower fidelity, blurred details
Stability of Training	Potential for instability due to adversarial nature	Stable due to explicit noise model	Stable due to deterministic encoding-decoding structure
Latent Space Interpolation	Smooth transitions, less coverage	Less smooth transitions, adequate coverage	Smooth transitions, excellent coverage
Mode Collapse	Prone due to adversarial training	Less prone due to denoising process	Least prone due to regularization in latent space

TABLE 4.2: Comparison between GAN, Diffusion, and Variational Autoencoder models

The properties are defined by:

- **Training Complexity:** Refers to the computational resources and time required to train the model. GANs generally require more resources and have a higher computational complexity compared to Diffusion and VAE models.
- **Sample Quality:** Refers to the quality of the samples generated by the model. GANs are generally capable of generating high-fidelity samples with sharp details, Diffusion models also create high-fidelity samples but with less sharp details, while VAEs typically produce samples of lower fidelity with more blurred details.
- **Stability of Training:** GANs can potentially exhibit instability due to their adversarial nature, while Diffusion models and VAEs provide more stable training due to their explicit noise model and deterministic encoding-decoding structure respectively.
- **Latent Space Interpolation:** Refers to the model's ability to generate novel samples by interpolating between points in the latent space. GANs and VAEs are known for their capability of creating smooth transitions between different points in the latent space, with VAEs typically having better coverage of the space, while Diffusion models may have less smooth transitions.
- **Mode Collapse:** Refers to the situation where the model produces similar or identical samples, failing to capture the diversity in the training data. GANs are prone to mode collapse due to the adversarial nature of their training, Diffusion models are less prone due to their denoising process, and VAEs are least prone due to the regularization imposed in the latent space.

Extensions of these models to 3D data, such as 3D-GANs Chan et al., 2021 and 3D-VAEs Foti et al., 2021, have been developed to generate 3D voxel-based representations.

In the case of point cloud data, deep learning models like PointNet Qi et al., 2016 and PointNet++ Qi et al., 2017 have been adapted for generative tasks, leading to Point Cloud GANs Li et al., 2018a and Point Cloud VAEs Guo, Du, and Zhao, n.d. that generate realistic 3D point cloud representations of objects. These models play a critical role in our quest to generate 3D mesh structures, as they provide valuable insights into dealing with non-Euclidean data and 3D representations.

4.3.2 Graph generation

Graph generation focuses on creating new graphs that resemble a given set of input graphs, capturing structural and relational properties. Generative models for graph data include GraphRNN You et al., 2018, GraphVAE Simonovsky and Komodakis, 2018, and GraphGAN Wang et al., 2017a. These models are designed to generate realistic graphs with varying sizes, structures, and properties.

For 3D mesh data, MeshGAN Cheng et al., 2019 and MeshVAE Tan et al., 2017 have been proposed, which extend the principles of GANs and VAEs to generate 3D meshes. These models are capable of capturing complex geometric structures and generating high-quality 3D mesh representations.

In "A systematic Survey on Deep Generative Models for Graph Generation Guo and Zhao, n.d.", they propose a taxonomy structure of different generative method of graph generation:

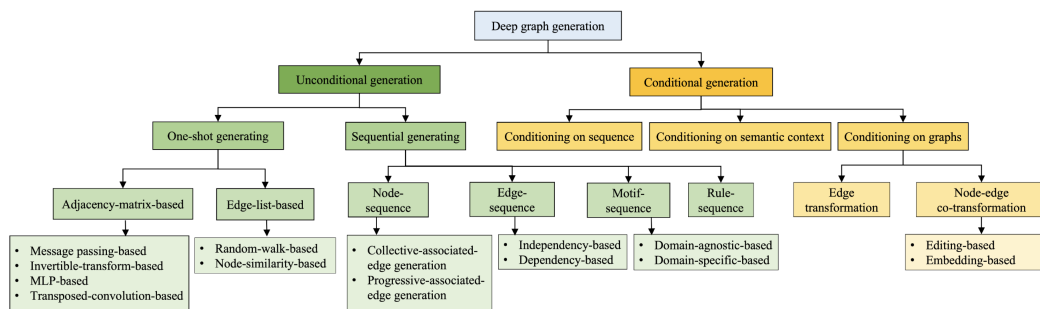


FIGURE 4.6: Classification of deep generative models for graph generation problems, Source: Guo and Zhao, n.d.

In the hierarchy depicted in Figure 4.6, the two main classes are unconditional generation and conditional generation. This master thesis focuses on the task of generating graphs given an input text. This falls under the umbrella of conditional generation methods. More specifically, the task can be seen as a form of conditioning on semantic context. This is a relatively new topic in the field of generative models for graph-structured data.

Given the relative novelty of this subject, it is crucial to leverage insights from other branches of the domain, particularly from our current understanding of technologies such as stable diffusion, which served as the initial inspiration for this work.

In the following section, I will first examine the utilization of CLIP in stable diffusion - a novel model that bridges image and text representations. This understanding will enable us to identify the most suitable model for our ultimate objective.

4.4 Contrastive Learning and CLIP

Joint-Embedding is a technique where a model learns to map images and their corresponding textual descriptions into a shared embedding space, enabling better understanding and manipulation of the relationship between the visual and textual data. By training on a large dataset of images and their associated captions, the model can generate high-quality images from text prompts and vice versa.

One approach to achieve Joint-Embedding is through **Contrastive Learning** Xie et al., 2021. Contrastive Learning leverages the principle of maximizing agreement between positive pairs (images and their associated captions) while minimizing agreement with negative pairs (images and bad captions). This learning objective encourages the model to project visually similar images and semantically related captions closer in the embedding space, while pushing visually dissimilar images and bad captions farther apart.

1. Contrastive pre-training

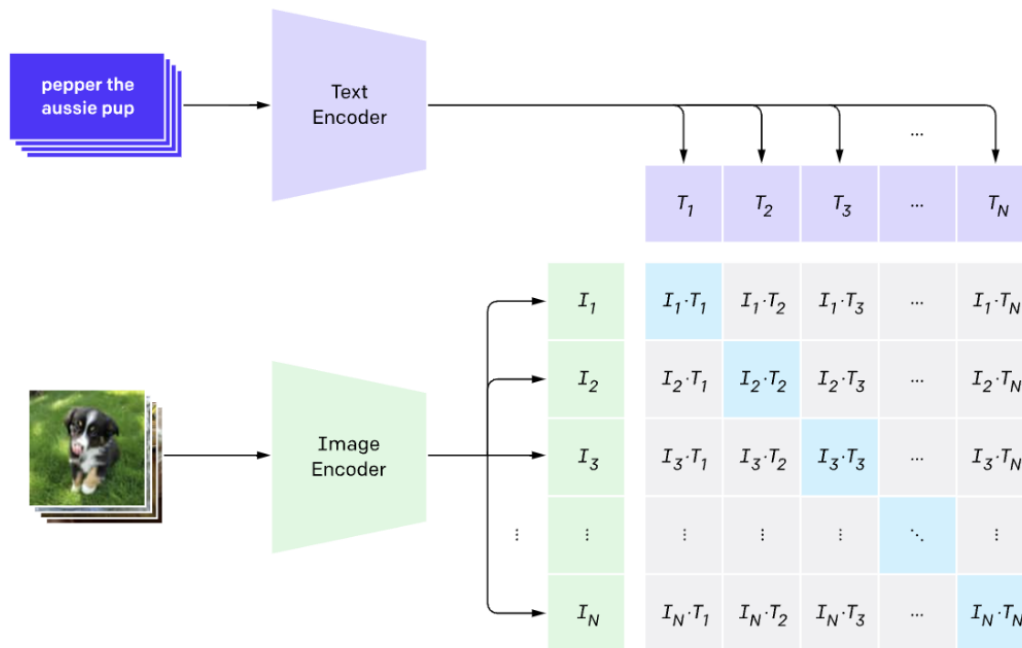


FIGURE 4.7: Contrastive pre-training

The figure 4.7, is an official illustration from clip that illustrate how it is pre-trained using contrastive learning. It is divided into 3 parts, a text encoder, an image encoder, and a shared embedding. Lets start by talking about the text encoder.

4.4.1 Text Encoder

The Text Encoder is the component of the CLIP model responsible for converting input text data into a high-dimensional vector representation, known as an embedding. This is done through a transformer model, a type of deep learning model that has been proven to be highly effective in Natural Language Processing (NLP) tasks.

In the context of CLIP, the text encoder takes a sentence (or any string of text) as input, processes it through several layers of transformer blocks, and outputs an embedding that represents the semantic content of the input text. Each block in the transformer model contains mechanisms to capture the dependencies between the words in the input sentence and to understand the contextual meaning of each word.

This text embedding can then be used to perform tasks like text classification, text generation, or, in the case of CLIP, to be compared with image embeddings in the shared embedding space.

In the next section, I will discuss the Image Encoder of the CLIP model.

4.4.2 Image Encoder

The Image Encoder in the CLIP model is responsible for transforming input images into a high-dimensional vector representation, similar to the text encoder. The image encoder can be one of two architectures: Vision Transformer (ViT) or a variant of the Residual Network (ResNet).

Vision Transformer (ViT)

The Vision Transformer is a recent model architecture that applies the transformer mechanism, traditionally used in NLP tasks, to computer vision tasks. In ViT, the image is first divided into a fixed number of patches. Each patch is then flattened and linearly transformed

into a sequence of embeddings. These embeddings are passed through a transformer model to capture the relationships between different parts of the image, resulting in a final image embedding.

Residual Network (ResNet)

ResNet is a popular model architecture in the field of computer vision. It is characterized by its use of skip connections, or shortcuts to jump over some layers. These shortcuts help solve the vanishing gradient problem, enabling the training of very deep networks. In the context of CLIP, a ResNet-based image encoder transforms an input image through a series of convolutional layers to generate a final image embedding.

Regardless of the architecture used, the image encoder outputs a vector that represents the semantic content of the input image. This image embedding can then be compared with text embeddings in the shared embedding space.

In the next section, I will delve into the shared embedding space, the area where both the text and image embeddings interact.

4.4.3 Shared Embedding Space

The shared embedding space is where the magic happens. In this space, the image and text embeddings interact, and CLIP learns to associate the semantic content of images and text. The embeddings from the image and text encoders are normalized and then compared using a cosine similarity function, which measures the cosine of the angle between the two vectors. This yields a similarity matrix S , with entries s_{ij} representing the similarity score between image i and text snippet j .

CLIP is then trained using a contrastive loss function, which encourages the model to increase the similarity of positive pairs (correct image-text pairs) and decrease the similarity of negative pairs (incorrect image-text pairs). In other words, the model is trained to match each image with its correct caption and mismatch it with other captions.

The loss function is formulated as follows:

$$l_i^{(u \rightarrow v)} = -\log \frac{\exp(s_{uv}/\tau)}{\sum_{k=1}^N \exp(s_{uk}/\tau)} \quad (4.6)$$

Here, $l_i^{(u \rightarrow v)}$ denotes the loss for image u given caption v , s_{uv} represents the similarity score (in this case cosine similarity) between image u and caption v , and τ is a temperature parameter that controls the concentration of the distribution. It's used to control the sharpness of the probability distribution calculated by the softmax function in the loss equation.

The temperature parameter influences the concentration of the probability mass of the softmax output:

- A high temperature value (i.e., $\tau \gg 1$) makes the distribution more uniform. In other words, the probabilities for all classes become closer to equal. This encourages exploration, as the model is less certain about its predictions.
- A low temperature (i.e., $\tau \ll 1$) makes the distribution more "sharp," with the probability mass concentrated on the class with the highest logits (before softmax). This makes the model more confident in its predictions, encouraging exploitation.

In CLIP, the temperature parameter τ helps in controlling how hard or soft the assignments of the text-image pairs are. The optimal value of τ is usually found through experimentation or hyperparameter tuning.

The loss for the reverse direction, that is, the loss for caption v given image u is calculated in a similar manner:

$$l_i^{(v \rightarrow u)} = -\log \frac{\exp(s_{vu}/\tau)}{\sum_{k=1}^N \exp(s_{vk}/\tau)} \quad (4.7)$$

Finally, the overall loss function, denoted by L , is computed as the sum of the losses for all image-caption pairs in the dataset:

$$L = \sum_{u,v} [l_i^{(u \rightarrow v)} + l_i^{(v \rightarrow u)}] \quad (4.8)$$

By minimizing this loss function, CLIP learns to map semantically similar images and text snippets close to each other in the shared embedding space.

Unlike standard classification tools, which are typically trained to map inputs to a fixed set of output classes, CLIP is designed to associate inputs from one modality (e.g., text) to inputs from another modality (e.g., images), in a way that preserves semantic relationships. This joint embedding approach enables CLIP to understand a much wider range of inputs than a typical classifier, since the potential range of inputs includes all possible texts and images, not just a pre-defined set of classes. In addition, CLIP's ability to understand the semantic content of both images and text allows it to leverage a vast amount of loosely labeled data from the web, which is something that standard classifiers often struggle with. CLIP's training process, which involves contrasting positive pairs (matching images and texts) against negative pairs (non-matching images and texts), also tends to produce more robust and generalizable representations than standard classification training. These advantages make CLIP an excellent tool for tasks involving cross-modal understanding and generation, such as the Stable Diffusion process I are considering.

In the next section, I will finally see the full architecture of stable diffusion that will be the main source of inspiration for my model presented in the next section.

4.5 Stable-Diffusion

Stable Diffusion AI, 2022 is a state of the art generative model that employs diffusion-based techniques to generate high-quality visual outputs, offering potential applications in domains such as computer vision and art^{4.8}



FIGURE 4.8: Stable diffusion high quality image generation example, Source: AI, 2022

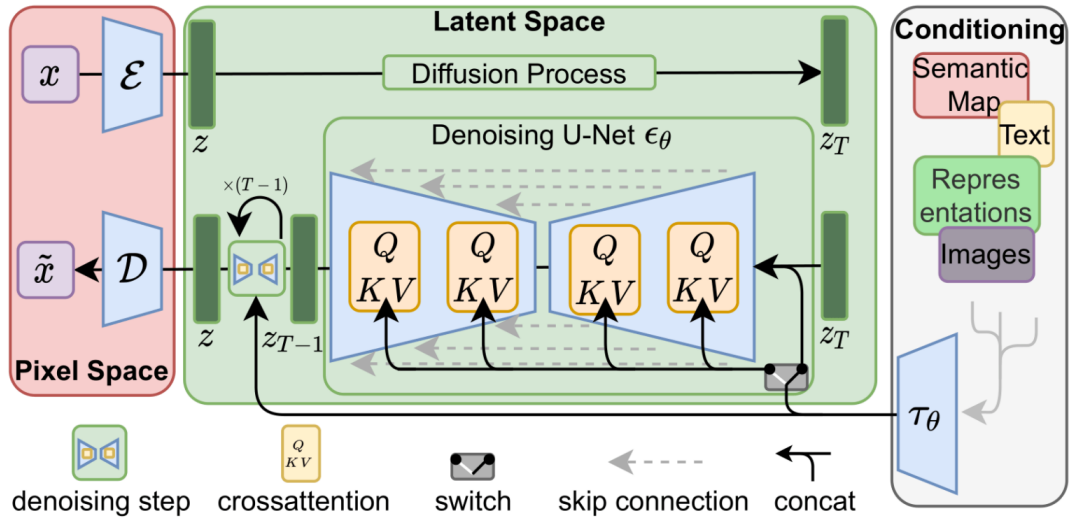


FIGURE 4.9: Stable diffusion architecture, Source: AI, 2022

Stable diffusion is a recent development in generative modeling, yet its fundamental principles have been explored in previous research. These principles typically involve setting up a generative process as a Markov chain, where each step evolves the state of the system (i.e., the generated data) according to certain rules, usually involving a form of random perturbation. Over time, the system converges to a stable distribution that is the target of the generative process.

The key novelty introduced by Stable Diffusion is the application of the CLIP model as a "switch". Instead of using a standard Markov chain process, Stable Diffusion applies the CLIP model to control the generative process. CLIP is a model developed by OpenAI that can learn to associate images and text by mapping them into a shared embedding space. By utilizing the ability of CLIP to understand both visual and textual data, Stable Diffusion uses this model to guide the Markov chain process, nudging it towards states that are more consistent with a given text prompt.

Moreover, the Stable Diffusion approach introduces an additional layer of abstraction by performing the diffusion process in a latent space. This latent space is obtained through a Variational Autoencoder (VAE). The VAE is a type of generative model that can learn to compress data into a lower-dimensional latent space in a way that is useful for generative tasks. The latent space captures the key factors of variation in the data, allowing the generative process to focus on these main factors, making the diffusion process more efficient and effective.

Therefore, the major innovation of Stable Diffusion comes from the combination of a Markov chain-based generative process with a powerful joint embedding model (CLIP), and the use of a latent space obtained from a VAE for the diffusion process. This combination results in a robust and versatile generative model that can generate high-quality data from complex and diverse prompts.

Chapter 5

MeshGNN: Model proposition

Given the current state-of-the-art, the highest quality generation models are typically based on diffusion models. However, these models can be computationally expensive and complex to train. Furthermore, the area of graph diffusion is relatively nascent, attracting high levels of interest and competitive research.

Taking into consideration these challenges and computational constraints, it quickly became apparent that Graph Variational Autoencoders (Graph VAEs) offer an attractive alternative. While they have been extensively used for generating various types of graph data, their potential for spatial data generation remains largely untapped.

A Graph VAE, like a traditional autoencoder, consists of an encoder and a decoder. For the encoder, I could utilize architectures like Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs). The decoder would be a simple cross product decoder.

To generate high-quality 3D meshes, the plan is to leverage the power of CLIP, a model developed by OpenAI. I propose two different architectures: one that includes a compatibility layer, which employs a Multilayer Perceptron (MLP) to bridge the gap between the image space and the graph space, and a second one where the CLIP image encoder model natively ingests graph embeddings. In the following sections, I will elaborate on these proposed architectures and discuss their potential benefits and drawbacks.

5.1 Fine Tuning of Clip without compatibility layer

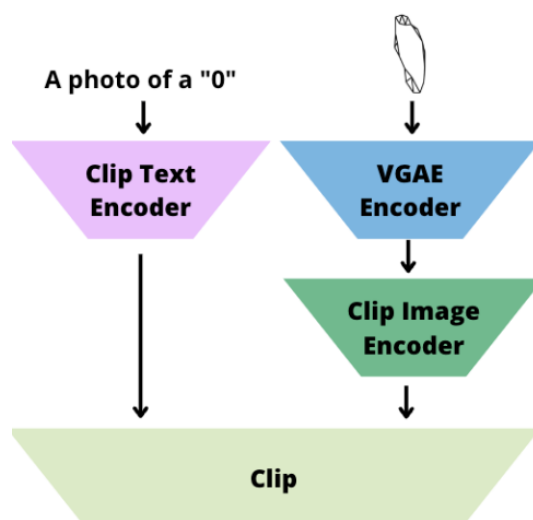


FIGURE 5.1: Clip Naive, without compatibility layer

In our proposed model, I aim to fine-tune CLIP without the compatibility layer. This involves two primary paths of information flow.

1. **Text-to-Clip:** In this path, I feed the input text to the CLIP text encoder. The resulting text embedding is then fed into the CLIP model.

2. **Graph-to-VAE-to-Clip:** In this path, I first feed the graph into the Graph VAE encoder, which transforms the graph into a low-dimensional embedding. This graph embedding is then passed into the CLIP image encoder, and the resulting image embedding is fed into the CLIP model.

The aim is to make the graph embedding (after being transformed to image form by the CLIP image encoder) as similar as possible to the concept described by the input text. During training, I optimize the model to minimize the distance between these two embeddings. This fine-tuning process allows us to generate graphs that closely match the descriptions provided in the input text.

The absence of a compatibility layer in our model design allows for a direct comparison between the embeddings generated from the text input and the graph input, potentially leading to a more efficient model. This proposed model leverages the strengths of Graph VAEs and CLIP, providing a novel approach for generating graph data from text inputs.

It is important to notice that the image encoder will create an image that will not be present in the data! For this method, the fine tuning is essential for it to work.

Another point to take into account is that, while it seems possible that passing from graph embedding to the shared embedding without passing through an autoencoder seems possible, I would argue that the training should start from scratch instead of finetuning. This would require a lot more training.

5.2 Fine Tuning of Clip with compatibility layer

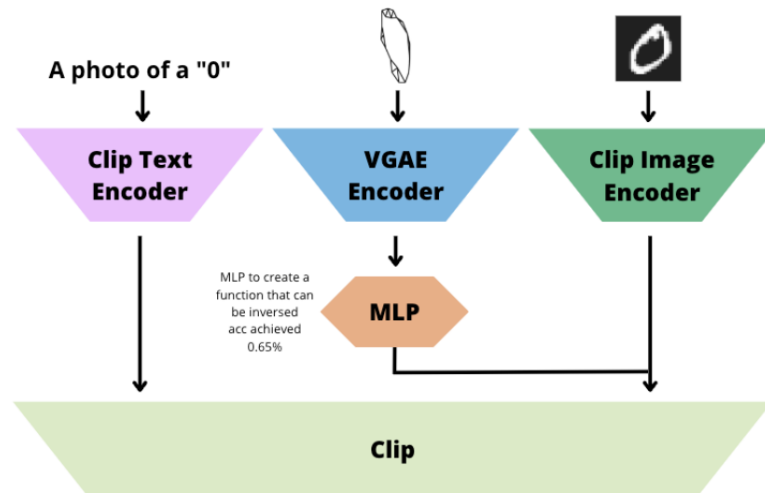


FIGURE 5.2: Clip training with compatibility layer

In an alternative fine-tuning approach, I can introduce a compatibility layer to connect the image space and the graph encoding space. The compatibility layer, implemented as a Multi-Layer Perceptron (MLP), is designed to transform the graph embeddings generated by the VAE encoder into a format compatible with the CLIP image encoder.

The information flow in this setup involves the following steps:

1. **Text-to-Clip:** As in the previous setup, I feed the input text directly into the CLIP text encoder and pass the resulting text embedding to the CLIP model.
2. **Graph-to-Image-to-Clip:** The input graph is projected into multiple images. These images are passed into the CLIP image encoder to generate image embeddings.
3. **Graph-to-VAE-to-MLP-to-Clip:** In parallel with the above step, the input graph is also passed into the Graph VAE encoder, which generates graph embeddings. These graph

embeddings are transformed by the MLP compatibility layer into a format that can be processed by the CLIP image encoder. The transformed embeddings are then passed into the CLIP model.

During training, the aim is to minimize the distance between the text embeddings and the embeddings generated from the MLP-transformed graph embeddings. The compatibility layer allows us to directly generate CLIP-compatible embeddings from the graph data, potentially enhancing the model's ability to generate graphs that closely match the descriptions provided in the input text.

This proposed approach effectively bridges the gap between the image and graph encoding spaces, leveraging the strengths of Graph VAEs, CLIP, and an MLP compatibility layer to provide a novel methodology for generating graph data from text inputs.

It is important to notice that the quality of the result will not depend on the quality of the image as input but the quality of the learned MLP. The Image has only here a role of classification.

5.3 Generation

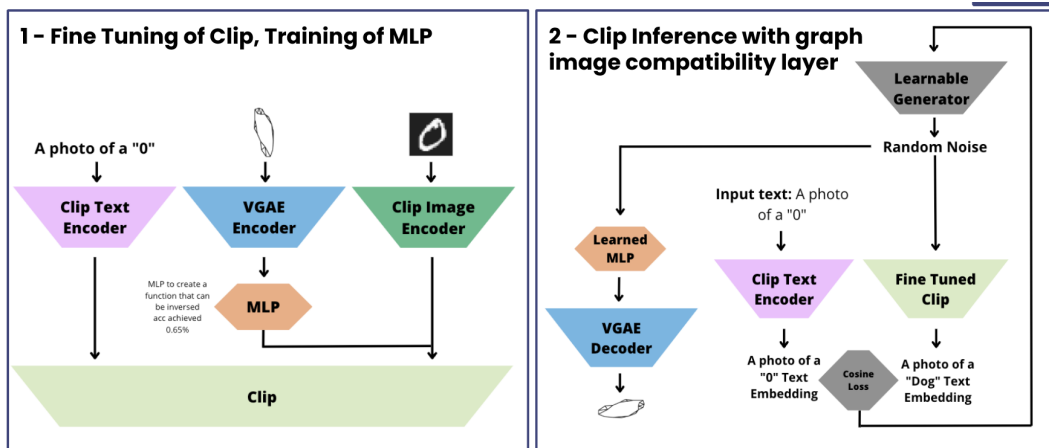


FIGURE 5.3: MeshGraph Full Architecture

In order to use CLIP for data generation, I modify its original use as a classifier and repurpose it as a generative model. This is accomplished using a learned Multi-Layer Perceptron (MLP) compatibility layer and a variational graph autoencoder (VAE).

The proposed generative approach involves the following steps:

1. **Noise Generation:** I initialize a learnable generator to produce a random noise vector. This noise vector is seen as the latent variable which would be used for generating the graph.
2. **MLP Transformation:** I feed the generated noise vector into the learned MLP, which transforms the noise into a format compatible with the image encoder of CLIP.
3. **Decoding to Graph:** The MLP-transformed noise vector is then fed into the Graph VAE decoder, which generates a graph representation from the noise vector.
4. **Prediction by CLIP:** The MLP-transformed noise vector is also fed into the fine-tuned CLIP model, resulting in an initial prediction of the graph's textual description.
5. **Comparison with Text Encoder:** In parallel with the above steps, the input text is passed through the CLIP text encoder, which generates an embedding representing the desired graph.

6. **Loss Calculation and Backpropagation:** A loss is calculated using cosine divergence between the prediction from the fine-tuned CLIP model and the embedding from the text encoder. This loss is backpropagated through the learnable generator, which adjusts its parameters to reduce the loss.
7. **Iteration until Convergence:** The above steps are repeated until the loss converges to a minimum. At this point, the learnable generator is capable of producing noise vectors that, when transformed by the MLP and decoded by the VAE, yield graph representations closely matching the descriptions provided in the input text.

In this way, I hope to achieve the generation of graphs that are in close alignment with the textual descriptions, utilizing the strengths of CLIP, Graph VAEs, and MLPs for generative tasks.

5.4 Limitations

As with all models, our proposed MeshGNN approach has its own set of limitations that need to be acknowledged:

1. **Training Complexity:** Training such a complex generative model could be a daunting task, as the interaction between different parts of the model such as Graph VAE, CLIP, and the MLP might lead to a high-dimensional, non-convex optimization problem. Therefore, training may be computationally intensive, and finding optimal solutions could be challenging.
2. **Graph Quality:** The quality of the generated graphs heavily depends on the capabilities of the VAE decoder. If the decoder fails to generate high-quality graph representations, the overall quality of the generated data might be low, regardless of the performance of other components in the system.
3. **Textual Description Dependence:** The generation process heavily relies on the textual description provided as input. Hence, the quality and specificity of this description could significantly affect the output. A vague or generic description might lead to poorly defined or ambiguous graphs.
4. **Data Requirement:** The model requires significant amounts of training data to effectively learn the intricate relationships between the text descriptions and the corresponding graph structures. This might pose a challenge if such specific training data is not readily available.
5. **Computational Resources:** The proposed architecture is likely to be resource-intensive, requiring significant computational power and memory for training and inference. This might limit its applicability in low-resource settings.
6. **Data Complexity:** The model might struggle with complex graphs or those that incorporate high-level abstract concepts that are difficult to encode in a graph structure.

Despite the limitations mentioned, the proposed MeshGNN architecture demonstrates the potential of leveraging a compatibility layer to bridge the gap between the image and graph spaces, enabling the generation of intricate 3D mesh structures. While it doesn't fully exploit the capabilities of the powerful CLIP model, it provides a more efficient approach that avoids the need for extensive retraining.

Future work could focus on refining this compatibility layer, exploring different types of transformations or architectures, or developing novel training strategies to better align the graph and image spaces. Additionally, investigating the application of MeshGNN in different domains, or extending it to handle more complex graph structures could also provide interesting avenues for future research.

Ultimately, the goal is to advance the state of the art in 3D mesh generation and contribute to the broader efforts in understanding and harnessing the complex interplay between textual descriptions and visual or graph-structured data.

Chapter 6

Experiment: CLIP Generation of 3D model

6.1 Datasets and Datapreparation

6.1.1 ShapenetSem

In this experiment, I will use the ShapeNetSem dataset released in June 2015 (v0) Savva, Chang, and Hanrahan, [n.d.](#) This dataset is a richly annotated subset of ShapeNet containing 3D models with various physical attributes. The dataset includes 3D models in multiple formats such as OBJ, COLLADA (DAE), and binary voxelizations (binvox).

The metadata associated with each model is stored in a metadata.csv file, containing information such as unique model ID, manually annotated categories, WordNet synset offsets and lemmas, normalized vectors indicating semantic orientation, scale unit, dimensions, container-like properties, volume, surface area, weight, and friction force. Other metadata files include material priors, material densities, and a taxonomy of manual categories.

I will preprocess the ShapeNetSem dataset by transforming the 3D models from mesh format to graph format. Several possible settings for this transformation can be considered:

- Directly convert the mesh to a graph by considering vertices as nodes and edges as connections.
- Use a nearest-neighbor approach to connect vertices within a certain distance threshold.
- Apply a clustering algorithm to group vertices and create nodes representing each cluster, followed by establishing connections based on cluster proximity.
- Compute a set of feature points on the mesh and connect them based on geometric or topological similarity, creating a sparse graph representation.

The choice of transformation settings will depend on the specific requirements of the stable diffusion process and the desired level of detail for the graph representation. By converting the 3D models into graphs, I aim to leverage the power of geometric deep learning techniques.

For the text embedding process, I will utilize an open source LLM [TheBloke/LLaMA-7b-GPTQ · Hugging Face n.d.](#) that transform raw input into english sentences definitions. This state-of-the-art model provides high-quality text embeddings, capturing both semantic and syntactic information in the input text. This metadata includes physical properties of the material, descriptions, and categories of objects. By converting this metadata into a continuous vector representation, I can efficiently incorporate this information into the stable diffusion process for 3D model generation. Utilizing this additional information can potentially improve the quality and accuracy of the generated 3D models by providing context-aware features.

6.1.2 MNISTSuperpixel

On the other hand, MNISTSuperpixel [torch_geometric.datasets.MNISTSuperpixels n.d.](#) is a relatively simpler dataset that contains graphs generated from MNIST digits by using a superpixel segmentation algorithm. Each graph node corresponds to a superpixel, and edges are formed between adjacent superpixels. The node features include the position and color information of the corresponding superpixel.

Despite its simplicity, this dataset allows us to explore the capabilities of my model in a controlled setting. In my experiments, I use the full dataset, comprising 60,000 graph representations of MNIST digits.

6.2 Implementation Details

My implementation is based on Python and makes use of the PyTorch <https://pytorch.org/> for the MeshGNN model and the PyTorch version of the CLIP model provided by OpenAI <https://openai.com>.

I make use of Google Colab Pro [Colab Subscription Pricing n.d.](#) for my experiments due to its affordability and the availability of powerful GPUs.

My code and models are publicly available on GitHub at <https://github.com/RobinPourtaud/meshgraph>.

For model tracking and version control, I use Weights and Biases (wandb) <https://wandb.ai/>. It provides a comprehensive suite of tools for tracking experiment metadata, including input data, hyperparameters, output data, and a fully versioned history of code. This allows us to keep track of different models and experiment runs, and compare their performances easily.

In the following sections, I present the results of my experiments and discuss their implications.

6.3 Results and comments

6.3.1 Graph Variational AutoEncoder

The proposed architecture utilizes a Graph Variational Autoencoder to facilitate the generation of diverse graphs derived from the dataset. This methodology forms the crux of the initial experimentation stage.

An extensive set of hyperparameters was scrutinized to ascertain the optimal configuration for the system. The specific details regarding the hyperparameters and the grid search process have been thoroughly documented on GitHub. The hyperparameters that emerged as most effective were a learning rate of 0.01, and the employment of the Adam optimizer for a span of 10 epochs. Although extending the epochs could potentially enhance the performance, it was observed that this incurred a prohibitively long computational duration. The batch size was set to 64.

The GCNConv, as anticipated, manifested as the superior convolution method across all subsequent experiments. The encoder hidden channels were configured to 64, and an output configuration of 8 proved adequate for the MNISTSuperpixel dataset. The system achieved a loss score of 1.008 [6.1](#)

The following sections illustrate some of the results derived from the aforementioned setup.

encoder_out	lr	optimizer	encoder_hidden	batch_size	encoder_label	loss
8	0.01	adam	64	128	GATConv	1.008
8	0.01	adam	64	64	GATConv	1.011
8	0.001	adam	64	64	GATConv	1.029
16	0.01	adam	64	128	GATConv	1.029
16	0.01	adam	128	128	GATConv	1.034
16	0.001	adam	128	64	GATConv	1.042
8	0.01	adam	128	64	GATConv	1.045
4	0.01	adam	128	64	GATConv	1.054

FIGURE 6.1: Top MNISTSuperpixel Loss for VGAE

Although the loss metric cannot be relied upon as an absolute measure of performance, it serves a crucial role in comparing different models during the training phase. Specifically, it enables a quick evaluation of whether or not the model is being effectively trained.

The experimental results indicated that the encoder output dimensionality was a paramount factor influencing the performance. Particularly for the MNIST dataset, it was observed that a lower encoder output dimensionality resulted in better performance. This could potentially be attributed to the inherent simplicity of the MNIST dataset.

The ensuing section presents the results derived from these experiments.

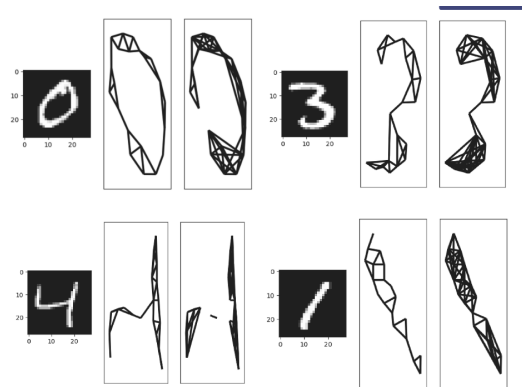


FIGURE 6.2: Sample results of generation

In Figure 6.2, for each numeral from left to right, we have the original MNIST image, its equivalent in the MNIST Superpixel dataset, and the reconstruction produced by the Variational Autoencoder.

The results obtained are encouraging, given that Variational Autoencoders aim to produce structures that are similar but not identical to the input data.

A similar process was conducted for the ShapeNet dataset, which only includes two classes: Tables and Chairs. Given the increased complexity of this dataset, the training process was substantially more challenging and time-consuming, which limited the scope for hyperparameter tuning and the number of epochs. These constraints may account for the observable quality of the results. The best model achieved a loss of 18.2. While it is difficult to interpret this loss value in isolation, the quality of the output models suggests that further analyses might not yield significant improvements. Figure 6.3 provides an overview of the results.

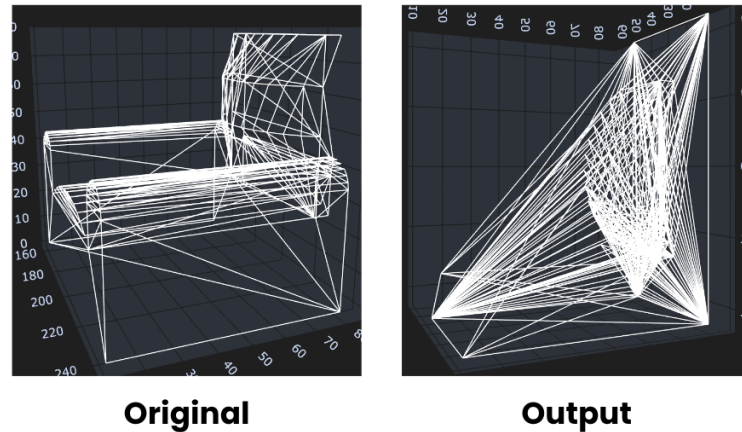


FIGURE 6.3: Shapenet Graph Variational Auto Encoder Results

For both experience, an hyperparameter has been used to choose the probability of link between nodes. It has been set to 0.7 arbitrarily, given its only visual impact.

6.3.2 Clip

Unfortunately, our attempt to utilize CLIP without a compatibility layer did not yield satisfactory results, primarily due to the computational limitations encountered when retraining such an extensive model on Google Colab Pro.

Therefore, the subsequent results presented here pertain exclusively to experiments conducted using CLIP with a compatibility layer.

Below are the top three image models utilized by CLIP that produced the most promising results:

	RN50x64	RN50x16	VIT-L/14
1	0.33	0.60	0.32
2	0.60	0.72	0.59
3	0.70	0.76	0.71
4	0.73	0.81	0.77
5	0.83	0.84	0.81

TABLE 6.1: CLIP with compatibility layer results for MNISTSuperpixel

In this context, "top" is defined by the global accuracy of the model in assigning the correct label within the top 1, top 2, etc., predictions.

OpenAI claims a CLIP accuracy rate of 66.6 percent, yet the highest I achieved was 60 percent. This discrepancy is likely due to the inherent inaccuracies of the Multilayer Perceptron (MLP) applied on top of the model.

6.3.3 Graph CLIP and Full Model

The complexity of comparing graph structures prevented us from using an automated process for evaluation. Therefore, the results were evaluated manually. Here are some examples with and without the compatibility layer:

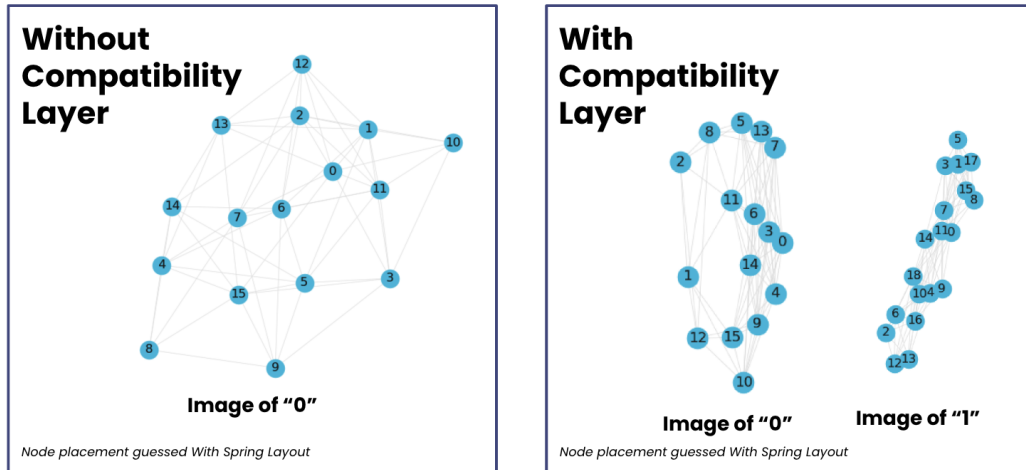


FIGURE 6.4: Result of the model to generate some example from text input, good examples

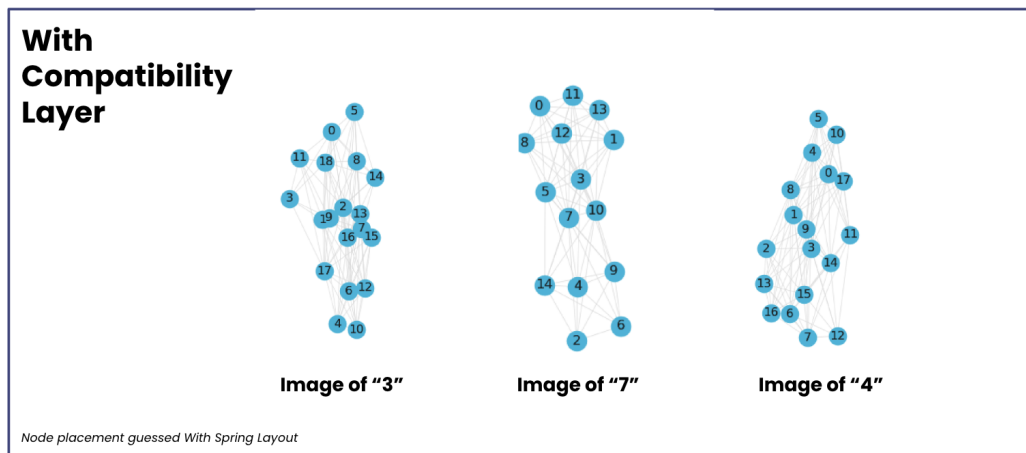


FIGURE 6.5: Result of the model to generate some examples from text input, bad examples

As anticipated, the experimental outcomes, although not ideal, demonstrate the functioning of the proposed system. In theory, enhanced performance can be achieved with more training.

There are four crucial reasons accounting for the current results:

1. The MLP, which is an integral part of the process, exhibits an accuracy of merely 65 percent.
2. The Variational Autoencoder (VAE) generates data that is similar to the input, but not identical, thereby adding an unpredictability element.
3. In the final experiment, the node embeddings were not computed, leaving the node placement task to the spring layout algorithm.
4. Despite these challenges, the model was able to generate correct outputs for some numbers, indicating that further training could potentially improve its performance.

The experiments undertaken in this thesis have demonstrated the potential for generating 3D mesh structures from textual inputs using a compatibility layer between graph space and image space without the need for retraining CLIP. Despite certain limitations and challenges, such as an accuracy level of 65 percent for the MLP, unpredictability in VAE's output,

and the computation of node embeddings, the model displayed promising results in some instances.

Though the results were not perfect, they showed that the system functions as intended and gave credence to the idea that, with more training, the performance could improve. The successful generation of correct outputs for certain numbers further reinforces this belief.

Chapter 7

Conclusion and Future Directions

In this research, the potential of generating 3D mesh structures, represented as graphs or hypergraphs, from textual inputs was thoroughly explored. The promising results obtained suggest numerous avenues for future research and applications.

One of the intriguing possibilities is the adaptation of the CLIP architecture to natively generate graphs, a significant shift from its current functionality, which predominantly involves image generation. This would not only open up a new realm of applications but would also contribute to the fundamental understanding of how various data types can be interconnected and used for generation tasks.

In a broader sense, the application of such a model could extend beyond 3D modeling and computer graphics. For instance, it might be employed in fields such as computational biology for the generation of protein structures or in social network analysis for the generation of community structures, based on textual descriptions or properties.

In conclusion, the field of graph generation, especially from diverse inputs such as textual data, is an emerging and exciting domain with vast potential for interdisciplinary research and applications. This thesis serves as a starting point for this exploration, and it is hoped that future work will continue to push the boundaries of what is possible in this area.

Appendix A

NII Introduction

The National Institute of Informatics (NII) is a leading research institute in Japan, focusing on the development and advancement of informatics and its applications. Established in 2000, the NII is dedicated to conducting cutting-edge research in various fields of information science, including artificial intelligence, data science, computer science, and information systems.

As an interdisciplinary research institute, the NII aims to provide solutions to real-world problems through the collaboration of experts from different domains. The institute is committed to fostering the growth of informatics and nurturing the next generation of researchers and professionals in the field. The NII offers a wide range of academic programs and research opportunities, from undergraduate and graduate level education to postdoctoral research positions.

The NII is actively involved in international collaborations and partnerships, working closely with universities, research institutions, and industries around the world. By leveraging its expertise and resources, the NII contributes to the global scientific community and helps address the challenges of today's rapidly evolving digital society.

With a strong focus on both theoretical and applied research, the NII continually pushes the boundaries of informatics, driving innovation and shaping the future of information science and technology.

<https://www.nii.ac.jp/>

Bibliography

- 2.2. *Manifold learning* — *scikit-learn 1.2.2 documentation* (n.d.). URL: <https://scikit-learn.org/stable/modules/manifold.html>.
- AI, Google (n.d.[a]). *Google AI updates: Bard and new AI features in Search*. URL: <https://blog.google/technology/ai/bard-google-ai-search-updates/>.
- (n.d.[b]). *Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance* – *Google AI Blog*. URL: <https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html>.
- AI, Meta (n.d.[c]). *Introducing LLaMA: A foundational, 65-billion-parameter language model*. URL: <https://ai.facebook.com/blog/large-language-model-llama-meta-ai/>.
- AI, Stability (2022). *Stable Diffusion*. Accessed: 2023-04-27.
- Arjovsky, Martin, Soumith Chintala, and Léon Bottou (Jan. 2017). “Wasserstein GAN”. In: URL: <https://arxiv.org/abs/1701.07875v3>.
- Belkin, Mikhail and Partha Niyogi (June 2003). “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural Computation* 15 (6), pp. 1373–1396. ISSN: 08997667. DOI: [10.1162/089976603321780317](https://doi.org/10.1162/089976603321780317).
- Bronstein, Michael M. et al. (Nov. 2016). “Geometric deep learning: going beyond Euclidean data”. In: *IEEE Signal Processing Magazine* 34 (4), pp. 18–42. ISSN: 10535888. DOI: [10.1109/msp.2017.2693418](https://doi.org/10.1109/msp.2017.2693418). URL: <https://arxiv.org/abs/1611.08097v2>.
- Brown, Tom B. et al. (May 2020). “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 2020-December*. ISSN: 10495258. URL: <https://arxiv.org/abs/2005.14165v4>.
- Cao, Shaosheng, Wei Lu, and Qionikai Xu (Oct. 2015). “GraRep: Learning graph representations with global structural information”. In: *International Conference on Information and Knowledge Management, Proceedings 19-23-Oct-2015*, pp. 891–900. DOI: [10.1145/2806416.2806512](https://doi.org/10.1145/2806416.2806512). URL: <https://dl.acm.org/doi/10.1145/2806416.2806512>.
- Carmo, Mantredo P. do (n.d.). *Differential Geometry of Curves and Surfaces*. URL: <http://www2.ing.unipi.it/griff/files/dC.pdf>.
- Chan, Eric R. et al. (Dec. 2021). “Efficient Geometry-aware 3D Generative Adversarial Networks”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2022-June*, pp. 16102–16112. ISSN: 10636919. DOI: [10.1109/CVPR52688.2022.01565](https://doi.org/10.1109/CVPR52688.2022.01565). URL: <https://arxiv.org/abs/2112.07945v2>.
- Cheng, Shiyang et al. (Mar. 2019). “MeshGAN: Non-linear 3D Morphable Models of Faces”. In: URL: <https://arxiv.org/abs/1903.10384v1>.
- Colab Subscription Pricing (n.d.). URL: <https://colab.research.google.com/signup>.
- Common Problems | Machine Learning | Google for Developers (n.d.). URL: <https://developers.google.com/machine-learning/gan/problems>.
- Devlin, Jacob et al. (Oct. 2018). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference 1*, pp. 4171–4186. URL: <https://arxiv.org/abs/1810.04805v2>.
- Fedorov, Andriy et al. (Nov. 2012). “3D Slicer as an image computing platform for the Quantitative Imaging Network”. In: *Magnetic resonance imaging* 30 (9), pp. 1323–1341. ISSN: 1873-5894. DOI: [10.1016/J.MRI.2012.05.001](https://doi.org/10.1016/J.MRI.2012.05.001). URL: <https://pubmed.ncbi.nlm.nih.gov/22770690/>.
- Foti, Simone et al. (Nov. 2021). “3D Shape Variational Autoencoder Latent Disentanglement via Mini-Batch Feature Swapping for Bodies and Faces”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2022-June*, pp. 18709–18718. ISSN: 10636919. DOI: [10.1109/CVPR52688.2022.01817](https://doi.org/10.1109/CVPR52688.2022.01817). URL: <https://arxiv.org/abs/2111.12448v5>.

- Goodfellow, Ian et al. (June 2014). “Generative Adversarial Networks”. In: *Communications of the ACM* 63 (11), pp. 139–144. ISSN: 15577317. DOI: [10 . 1145 / 3422622](https://doi.org/10.1145/3422622). URL: [https : // arxiv . org / abs / 1406 . 2661v1](https://arxiv.org/abs/1406.2661v1).
- Grover, Aditya and Jure Leskovec (July 2016). “node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 13-17-August-2016, pp. 855–864. ISSN: 2154-817X. DOI: [10 . 1145 / 2939672 . 2939754](https://doi.org/10.1145/2939672.2939754). URL: [https : // arxiv . org / abs / 1607 . 00653v1](https://arxiv.org/abs/1607.00653v1).
- Guo, Xiaojie, Yuanqi Du, and Liang Zhao (n.d.). “PROPERTY CONTROLLABLE VARIATIONAL AUTOEN-CODER VIA INVERTIBLE MUTUAL DEPENDENCE”. In: (). URL: [https : // github . com / xguo7 / PCVAE . .](https://github.com/xguo7/PCVAE)
- Guo, Xiaojie and Liang Zhao (n.d.). “A Systematic Survey on Deep Generative Models for Graph Generation”. In: ().
- (July 2020). “A Systematic Survey on Deep Generative Models for Graph Generation”. In: URL: [http : // arxiv . org / abs / 2007 . 06686](http://arxiv.org/abs/2007.06686).
- Hailu, Tulu Tilahun et al. (Feb. 2020). “Intrinsic and Extrinsic Automatic Evaluation Strategies for Paraphrase Generation Systems”. In: *Journal of Computer and Communications* 8 (2), pp. 1–16. ISSN: 2327-5219. DOI: [10 . 4236 / JCC . 2020 . 82001](https://doi.org/10.4236/JCC.2020.82001). URL: [http : // www . scirp . org / journal / PaperInformation . asp x ? PaperID = 98203](http://www.scirp.org/journal/PaperInformation.aspx?PaperID=98203)[http : // www . scirp . org / Journal / Paperabs . asp x ? paperid = 98203](http://www.scirp.org/Journal/Paperabs.aspx?paperid=98203).
- Hajij, Mustafa et al. (Mar. 2021). “Simplicial Complex Representation Learning”. In: URL: [http : // arxiv . org / abs / 2103 . 04046](http://arxiv.org/abs/2103.04046).
- Hamilton, William L., Rex Ying, and Jure Leskovec (June 2017). “Inductive Representation Learning on Large Graphs”. In: *Advances in Neural Information Processing Systems* 2017-December, pp. 1025–1035. ISSN: 10495258. URL: [https : // arxiv . org / abs / 1706 . 02216v4](https://arxiv.org/abs/1706.02216v4).
- Ho, Jonathan, Ajay Jain, and Pieter Abbeel (June 2020). “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems* 2020-December. ISSN: 10495258. URL: [https : // arxiv . org / abs / 2006 . 11239v2](https://arxiv.org/abs/2006.11239v2).
- Introducing the Knowledge Graph: things, not strings* (n.d.). URL: [https : // blog . google / products / search / introducing - knowledge - graph - things - not /](https://blog.google/products/search/introducing-knowledge-graph-things-not/).
- Kingma, Diederik P. and Max Welling (Dec. 2013). “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. URL: [https : // arxiv . org / abs / 1312 . 6114v11](https://arxiv.org/abs/1312.6114v11).
- Kipf, Thomas N. and Max Welling (Sept. 2016a). “Semi-Supervised Classification with Graph Convolutional Networks”. In: URL: [http : // arxiv . org / abs / 1609 . 02907](http://arxiv.org/abs/1609.02907).
- (Sept. 2016b). “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. URL: [https : // arxiv . org / abs / 1609 . 02907v4](https://arxiv.org/abs/1609.02907v4).
- (Nov. 2016c). “Variational Graph Auto-Encoders”. In: URL: [https : // arxiv . org / abs / 1611 . 07308v1](https://arxiv.org/abs/1611.07308v1).
- Lan, Zhenzhong et al. (Sept. 2019). “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: URL: [https : // arxiv . org / abs / 1909 . 11942v6](https://arxiv.org/abs/1909.11942v6).
- Le, Quoc and Tomas Mikolov (May 2014). “Distributed Representations of Sentences and Documents”. In: *31st International Conference on Machine Learning, ICML 2014* 4, pp. 2931–2939. URL: [https : // arxiv . org / abs / 1405 . 4053v2](https://arxiv.org/abs/1405.4053v2).
- Li, Chun Liang et al. (Oct. 2018a). “Point Cloud GAN”. In: *Deep Generative Models for Highly Structured Data, DGS@ICLR 2019 Workshop*. URL: [https : // arxiv . org / abs / 1810 . 05795v1](https://arxiv.org/abs/1810.05795v1).
- Li, Jian Peng et al. (Feb. 2012). “A meta-analysis of voxel-based morphometry studies of white matter volume alterations in Alzheimer’s disease”. In: *Neuroscience and biobehavioral reviews* 36 (2), pp. 757–763. ISSN: 1873-7528. DOI: [10 . 1016 / J . NEUBIOREV . 2011 . 12 . 001](https://doi.org/10.1016/J.NEUBIOREV.2011.12.001). URL: [https : // pubmed . ncbi . nlm . nih . gov / 22192882 /](https://pubmed.ncbi.nlm.nih.gov/22192882/).
- Li, Pengfei et al. (Apr. 2020). “Bag-of-Concepts representation for document classification based on automatic knowledge acquisition from probabilistic knowledge base”. In: *Knowledge-Based Systems* 193, p. 105436. ISSN: 0950-7051. DOI: [10 . 1016 / J . KNOSYS . 2019 . 105436](https://doi.org/10.1016/J.KNOSYS.2019.105436).
- Li, Yujia et al. (Mar. 2018b). “Learning Deep Generative Models of Graphs”. In: URL: [https : // arxiv . org / abs / 1803 . 03324v1](https://arxiv.org/abs/1803.03324v1).
- Liu, Chengyi et al. (Feb. 2023). “Generative Diffusion Models on Graphs: Methods and Applications”. In: URL: [https : // arxiv . org / abs / 2302 . 02591v2](https://arxiv.org/abs/2302.02591v2).

- Liu, Yinhan et al. (July 2019). “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: URL: <https://arxiv.org/abs/1907.11692v1>.
- Lorensen, William E. and Harvey E. Cline (Aug. 1987). “Marching cubes: A high resolution 3D surface construction algorithm”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*, pp. 163–169. DOI: [10.1145/37401.37422](https://doi.org/10.1145/37401.37422). URL: https://www.researchgate.net/publication/202232897_Marching_Cubes_A_High_Resolution_3D_Surface_Construction_Algorithm.
- Luebke, David. et al. (2002). “Level of Detail for 3D Graphics : Application and Theory.” In: p. 431.
- Matthew Bolitho, Hugues Hoppe Michael Kazhdan (n.d.). *Poisson surface reconstruction | Proceedings of the fourth Eurographics symposium on Geometry processing*. URL: <https://dl.acm.org/doi/10.5555/1281957.1281965>.
- Mehrbani, Eysan, · Mohammad, and Hossein Kahaei (Mar. 2021). “Low-Rank Isomap Algorithm”. In: URL: <https://arxiv.org/abs/2103.04060v1>.
- Mikolov, Tomas et al. (n.d.). “Distributed Representations of Words and Phrases and their Compositionality”. In: ().
- Mirza, Mehdi and Simon Osindero (Nov. 2014). “Conditional Generative Adversarial Nets”. In: URL: <https://arxiv.org/abs/1411.1784v1>.
- Monti, Federico et al. (Nov. 2016). “Geometric deep learning on graphs and manifolds using mixture model CNNs”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-January, pp. 5425–5434. DOI: [10.1109/CVPR.2017.576](https://doi.org/10.1109/CVPR.2017.576). URL: <https://arxiv.org/abs/1611.08402v3>.
- Morris, Meaghan et al. (2011). “Review The Many Faces of Tau”. In: *Neuron* 70 (3), pp. 410–426. ISSN: 1063-6692. URL: <http://dx.doi.org/10.1016/j.neuron.2011.04.009>.
- Nash (n.d.). “<https://www.cs.upc.edu/~ia/nash51.pdf>”. In: ().
- OpenAI (2022). *DALL-E 2*. Accessed: 2023-04-27.
- (Mar. 2023). “GPT-4 Technical Report”. In: URL: <https://arxiv.org/abs/2303.08774v3>.
- O’Shea, Keiron and Ryan Nash (Nov. 2015). “An Introduction to Convolutional Neural Networks”. In: *International Journal for Research in Applied Science and Engineering Technology* 10 (12), pp. 943–947. DOI: [10.22214/ijraset.2022.47789](https://doi.org/10.22214/ijraset.2022.47789). URL: <https://arxiv.org/abs/1511.08458v2>.
- PapersWithCode (n.d.). *3D Object Recognition*. URL: <https://paperswithcode.com/task/3d-object-recognition>.
- Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena (Mar. 2014). “DeepWalk: Online Learning of Social Representations”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710. DOI: [10.1145/2623330.2623732](https://doi.org/10.1145/2623330.2623732). URL: <http://arxiv.org/abs/1403.6652><http://dx.doi.org/10.1145/2623330.2623732>.
- Peters, Matthew E. et al. (Feb. 2018). “Deep contextualized word representations”. In: *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference 1*, pp. 2227–2237. DOI: [10.18653/v1/n18-1202](https://doi.org/10.18653/v1/n18-1202). URL: <https://arxiv.org/abs/1802.05365v2>.
- Qi, Charles R et al. (n.d.). “Volumetric and Multi-View CNNs for Object Classification on 3D Data”. In: ().
- Qi, Charles R. et al. (Dec. 2016). “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-January, pp. 77–85. DOI: [10.1109/CVPR.2017.16](https://doi.org/10.1109/CVPR.2017.16). URL: <https://arxiv.org/abs/1612.00593v2>.
- Qi, Charles R. et al. (June 2017). “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *Advances in Neural Information Processing Systems 2017-December*, pp. 5100–5109. ISSN: 10495258. URL: <https://arxiv.org/abs/1706.02413v1>.
- Radford, Alec, Luke Metz, and Soumith Chintala (Nov. 2015). “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. URL: <https://arxiv.org/abs/1511.06434v2>.
- Radford, Alec et al. (Feb. 2021). “Learning Transferable Visual Models From Natural Language Supervision”. In: URL: <https://arxiv.org/abs/2103.00020v1>.
- Remondino, Fabio and Sabry El-Hakim (n.d.). “IMAGE-BASED 3D MODELLING: A REVIEW”. In: ().

- Salton, Gerard and Christopher Buckley (Jan. 1988). "Term-weighting approaches in automatic text retrieval". In: *Information Processing Management* 24 (5), pp. 513–523. ISSN: 0306-4573. DOI: [10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0).
- Sanh, Victor et al. (Oct. 2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: URL: <https://arxiv.org/abs/1910.01108v4>.
- Savva, Manolis, Angel X Chang, and Pat Hanrahan (n.d.). "Semantically-Enriched 3D Models for Common-sense Knowledge". In: (). URL: <http://graphics.stanford.edu/projects/semgeo/>.
- Scarselli, Franco et al. (Jan. 2009). "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20 (1), pp. 61–80. ISSN: 10459227. DOI: [10.1109/TNN.2008.2005605](https://doi.org/10.1109/TNN.2008.2005605).
- Simonovsky, Martin and Nikos Komodakis (Feb. 2018). "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11139 LNCS, pp. 412–422. ISSN: 16113349. DOI: [10.1007/978-3-030-01418-6_41](https://doi.org/10.1007/978-3-030-01418-6_41). URL: <https://arxiv.org/abs/1802.03480v1>.
- Stability.ai (n.d.). *Stable Diffusion Public Release - Stability.ai*. URL: <https://stability.ai/blog/stable-diffusion-public-release>.
- Tan, Qingyang et al. (Sept. 2017). "Variational Autoencoders for Deforming 3D Mesh Models". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 5841–5850. ISSN: 10636919. DOI: [10.1109/CVPR.2018.00612](https://doi.org/10.1109/CVPR.2018.00612). URL: <https://arxiv.org/abs/1709.04307v3>.
- Tenenbaum, J. B., V. De Silva, and J. C. Langford (Dec. 2000). "A global geometric framework for nonlinear dimensionality reduction". In: *Science (New York, N.Y.)* 290 (5500), pp. 2319–2323. ISSN: 0036-8075. DOI: [10.1126/SCIENCE.290.5500.2319](https://doi.org/10.1126/SCIENCE.290.5500.2319). URL: <https://pubmed.ncbi.nlm.nih.gov/11125149/>.
- TheBloke/LLaMA-7b-GPTQ · Hugging Face (n.d.). URL: <https://huggingface.co/TheBloke/LLaMA-7b-GPTQ>.
- `torch_geometric.datasets.MNISTSuperpixels` (n.d.). URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.MNISTSuperpixels.html.
- Tran, Du et al. (Nov. 2017). "A Closer Look at Spatiotemporal Convolutions for Action Recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 6450–6459. ISSN: 10636919. DOI: [10.1109/CVPR.2018.00675](https://doi.org/10.1109/CVPR.2018.00675). URL: <https://arxiv.org/abs/1711.11248v3>.
- Vaswani, Ashish et al. (June 2017). "Attention Is All You Need". In: URL: <http://arxiv.org/abs/1706.03762>.
- Veličković, Petar et al. (Oct. 2017). "Graph Attention Networks". In: URL: <http://arxiv.org/abs/1710.10903>.
- Wang, Hongwei et al. (Nov. 2017a). "GraphGAN: Graph Representation Learning with Generative Adversarial Nets". In: URL: <https://arxiv.org/abs/1711.08267v1>.
- Wang, Qi et al. (June 2017b). "Two improved continuous bag-of-word models". In: *Proceedings of the International Joint Conference on Neural Networks 2017-May*, pp. 2851–2856. DOI: [10.1109/IJCNN.2017.7966208](https://doi.org/10.1109/IJCNN.2017.7966208).
- Wang, Yue et al. (Jan. 2018). "Dynamic Graph CNN for Learning on Point Clouds". In: *ACM Transactions on Graphics* 38 (5), p. 13. ISSN: 15577368. DOI: [10.1145/3326362](https://doi.org/10.1145/3326362). URL: <https://arxiv.org/abs/1801.07829v2>.
- Xie, Zhongwei et al. (2021). "Learning Text-Image Joint Embedding for Efficient Cross-Modal Retrieval with Deep Feature Engineering". In: DOI: [10.1145/3490519](https://doi.org/10.1145/3490519). URL: <https://doi.org/10.1145/3490519>.
- Xu, Keyulu et al. (Oct. 2018). "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019*. URL: <https://arxiv.org/abs/1810.00826v3>.
- Xu, Yusheng, Xiaohua Tong, and Uwe Stilla (June 2021). *Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry*. DOI: [10.1016/j.autcon.2021.103675](https://doi.org/10.1016/j.autcon.2021.103675).
- You, Jiaxuan et al. (Feb. 2018). "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models". In: *35th International Conference on Machine Learning, ICML 2018* 13, pp. 9072–9081. URL: <https://arxiv.org/abs/1802.08773v3>.

- Zhou, Zixiang, Yang Zhang, and Hassan Foroosh (n.d.). "Panoptic-PolarNet: Proposal-free LiDAR Point Cloud Panoptic Segmentation". In: (). URL: <https://github.com/edwardzhou130/Panoptic-PolarNet>.
- Zhu, Yanqiao et al. (Mar. 2022). "A Survey on Deep Graph Generation: Methods and Applications". In: URL: <https://arxiv.org/abs/2203.06714v3>.
- Zomorodian, Afra J. (Jan. 2005). "Topology for Computing". In: *Topology for Computing*. DOI: 10.1017/CB09780511546945. URL: <https://www.cambridge.org/core/books/topology-for-computing/1171035B570105A57865CEA390BA5E74>.