Pretraining Graph Transformers - Bachelor's Thesis

Alexander Krauck
Department of Machine Learning
Johannes Kepler University Linz
Upper Austria, Austria
alexander.krauck@gmail.com

Abstract

Recently, Graph Transformers (GT) have surfaced as a leading model in machine learning, outperforming previous benchmarks set by methods like Graph Neural Networks (GNN), particularly in tasks like molecular property prediction. Despite their remarkable success, the potential merits of pretraining with GTs have not been extensively studied, leaving a gap in the literature. This study addresses this gap by conducting a series of experiments on two different pretraining methodologies, one of which is a novel approach, and fine-tuning across multiple datasets. Firstly, the study aims to explore the possibilities for pretraining GTs. Secondly, it strives to determine the benefits that can be expected of pretraining GTs for downstream tasks. The results suggest that pretraining can considerably enhance the performance and efficiency of GTs. Alongside these findings, this study provides a comprehensive discussion on the correct application of GTs within existing frameworks. It also highlights the present challenges and limitations in the field, and proposes a novel GT approach, a modification of the Graphormer3D.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

As the frontiers of deep learning continue to expand, novel paradigms such as Graph Transformers (GT) are emerging and warrant comprehensive study. Incorporating methodologies from the well-established Graph Neural Networks (GNN), GTs demonstrate robust capabilities for encapsulating complex properties across a graph. A corpus of recent research, including works by Rampasek et al. [2022], Kim et al. [2022], and Ying et al. [2021], has conclusively illustrated that GTs frequently outperform GNN approaches, underpinning their growing relevance.

In light of this innovative class of machine learning models, there is a compelling necessity to conduct rigorous studies around pretraining and thereby fortify both theoretical and practical foundations of GTs. While initial research on pretraining methods for GTs exists, the depth of exploration into the underlying differences among these methods and the comprehensive experimental assessment of their impacts remain relatively unaddressed in the current literature. In addition, the scarcity of publicly available documentation on pretraining GT models, such as Graphormer by [Ying et al., 2021], hinders reproducibility and transparency in the field.

Despite the considerable potential of GTs, their implementation often necessitates the construction of the accompanying pipeline by the users. This can be attributed to the limitations of the fairseq framework by [Ott et al., 2019], the original platform for Graphormer implementation. While fairseq is a powerful tool for sequence-to-sequence learning, it's not particularly user-friendly, especially for beginners trying to get the system up and running. This high learning curve can be a barrier to its wider adoption. Moreover, fairseq's preprocessing capabilities are limited, often requiring the use of additional external libraries for complex tasks, adding another layer of complexity for users. Additionally, its design is primarily focused on sequence-to-sequence tasks, and it may not be as flexible or easy to adapt for tasks beyond this domain. These constraints, along with the relative novelty of other frameworks like Huggingface Transformers and DeepChem in relation to GTs,

contribute to the overall challenges of implementing GTs, underscoring the need for more accessible and robust platforms.

While some GNN pretraining method surveys exist and might be adapted for GTs, the literature lacks inclusive publications that incorporate comprehensive experimentation. This gap impedes the generation of pretrained models for a diverse range of downstream tasks, such as molecular property prediction.

1.2 Objectives

The ambition of this research is two-fold. Firstly, this study aims to explore the range of possibilities for pretraining GTs. Secondly, I intend to experimentally measure the impact of using pretraining methods on GTs for later downstream tasks, such as molecular property prediction.

From a practical perspective, the research strives to offer a transparent and robust implementation of pretraining methods for GTs with the aim of democratizing access to state-of-the-art machine learning methods[1]. The intention is to stimulate advanced research in graph data analytics, specifically in the realm of molecular studies utilizing deep learning techniques. As an essential part of this practical application, this study aims to contribute enhancements and additions to the widely-used Huggingface Transformers library on GitHub. These proposed contributions will encompass the pretraining methods developed and applied throughout this research.

1.3 Thesis Structure

The remainder of this thesis is organized as follows:

- **Section 2: Notation** – This section establishes the notation used throughout the thesis.
- **Section 3: Related Work** – This part provides an overview of GTs, pretraining methodologies, and related studies in the field. It also points out existing gaps in the research and practical landscape that this thesis seeks to address.
- **Section 4: Methodology** – This section presents the proposed pretraining methods for GTs and explains my approach to their implementation. Additionally, this chapter provides an extensive discussion on data selection and processing, including addressing specific challenges encountered while using the Huggingface Transformers library.
- **Section 5**: This chapter begins with a detailed explanation of the experimental setup, providing the context for the presented results. It then proceeds to present the outcomes of my experiments and provides a comprehensive analysis. The effects of various pretraining methods on the performance of GTs are discussed in depth.
- **Section 6: Conclusion and Future Work** – I summarize my findings and contributions in this section, discussing their implications for the broader field of machine learning. I also outline potential directions for future research.

Through this roadmap, I aim to enhance the clarity and accessibility of my research, facilitating its replication and further development by other researchers in the field.

2 Notation

In this thesis, specific data elements, functions or classes in the code are denoted using the typewriter font, \texttt, such as `edge_input`. This provides a direct reference to the actual variable or data structure names in the code base, enhancing the readability and concrete understanding of the underlying algorithms and processes.

The input graph $\mathcal{G}$ is comprised of two major components. Firstly, the `node_feat` matrix of $\mathbb{N}^{n \times m}$ with n nodes, with each node encapsulating m node-features. Secondly, an adjacency matrix $\mathbf{A} \in \mathbb{B}^{n \times n}$, which is always symmetric for the graphs discussed in this thesis, represents the connections between nodes. Furthermore, from a sparse perspective, there is the `edge_attr` matrix

[1]The code used in the course of this research is available under https://github.com/alexanderkrauck/pretraining-graph-transformers

$\mathbf{F} \in \mathbb{N}^{l \times j}$ where l is the number of edges in the graph and j is the number of edge features for each edge. In particular, a "degree" node-feature $\mathbf{D} \in \mathbb{N}^n$ which can be calculated by summing $\mathbf{A}$ over one axis is also relevant for later sections in this work.

Moreover, the `path` matrix $\mathbf{P} \in \mathbb{N}^{n \times n}$ and the `spatial_pos` matrix $\mathbf{S} \in \mathbb{N}^{n \times n}$, both calculated from $\mathbf{A}$ via the Floyd–Warshall algorithm exist. With $\mathbf{P}$ it is possible to determine a shortest path between two arbitrary nodes in $\mathcal{G}$, while with $\mathbf{S}$ it can directly seen how far apart two nodes are in a graph when following a shortest path.

Lastly, the `input_edges` tensor of $\mathbb{N}^{h \times n \times n \times j}$, where h is equal to $\max(\mathbf{S})$, which models the edge features $\mathbf{F}$ in `edge_attr` along a shortest path between any two nodes in $\mathcal{G}$. Its importance is further explained in Section 3.3.2.

3 Background and Related Work

3.1 Transformer

the Transformer architecture invented by [Vaswani et al., 2017] as depicted in Figure 1 consists of a variable amount of inputs, which are embedded initially. Those embeddings of variable size depending on a hyperparameter are passed through a fixed number of Transformer layers in order to refine the embeddings. The refinement in each block takes place by the use of a multi-head-attention block, where practically each embedding is communicating with each other embedding to exchange information, followed by a two-layer feed-forward neural network for each embedding separately for refinement. Furthermore, often a similar decoder block is employed to produce sequential output data like text in NLP tasks.

3.2 Graph Transformers

GTs are a recent variation of the Transformer architecture discussed briefly in Section 3.1 that are built for processing graph data. Different approaches exist to make use of all the data a graph has to offer, while still being reasonably efficient [Ying et al., 2021, Zhang et al., 2022, Rampasek et al., 2022, Kim et al., 2022]. The big two advantages of GTs, especially compared to GNNs, are that firstly in each layer of the Transformer encoder each node has access to information from each other node and not only the neighbors. Secondly, GTs are more expressive than the Weisfeiler-Lehman graph isomorphism (WL) test, which heavily depends on the exact implementation of the GT, but some implementations even have full expressivity [Kim et al., 2022, Rampasek et al., 2022] which is backed up by the theory from [Kreuzer et al., 2021].

However, those advantages come at the cost of more required system resources. In particular, while most GNN architectures scale with $\mathcal{O}(n \cdot \max(\mathbf{D}))$, GTs require $\mathcal{O}(n^2)$. It is worth noting that there is a hybrid version by [Rampasek et al., 2022] termed GraphGPS between GNN and GT that is less costly. However, the scope of this paper lies on pure Transformer models.

One of the most interesting parts of constructing a machine learning model for graphs is how the edge information is utilized. In GNNs information is only exchanged between neighboring nodes where the neighborhood is defined by the edges except some versions of GNNs that somehow attempt to increase the expressivity [Cai et al., 2023, Krauck, 2023, You et al., 2021]. The GNN approach is well established in the metier of processing graph information but it comes with the issue that the nodes will only ever get local information from its neighborhood in each layer. In contrast, as the research of GTs is still rather young, there are still many very different ideas on how to incorporate the edge information, but generally, as GTs have access to information from each other node in each layer and not only the neighbors, stronger methods should generally be possible. One approach therefore is detailed in the Section 3.3 below.

3.3 Graphormer

Graphormer proposed by [Ying et al., 2021] is a modified version of the encoder part of the well known and most successful Transformer-architecture [Vaswani et al., 2017] detailed in Section 3.1.

The modifications made from the Transformer to Graphormer, which are primarily in the node embedding and attention bias terms, endow the Transformer architecture with sufficient information

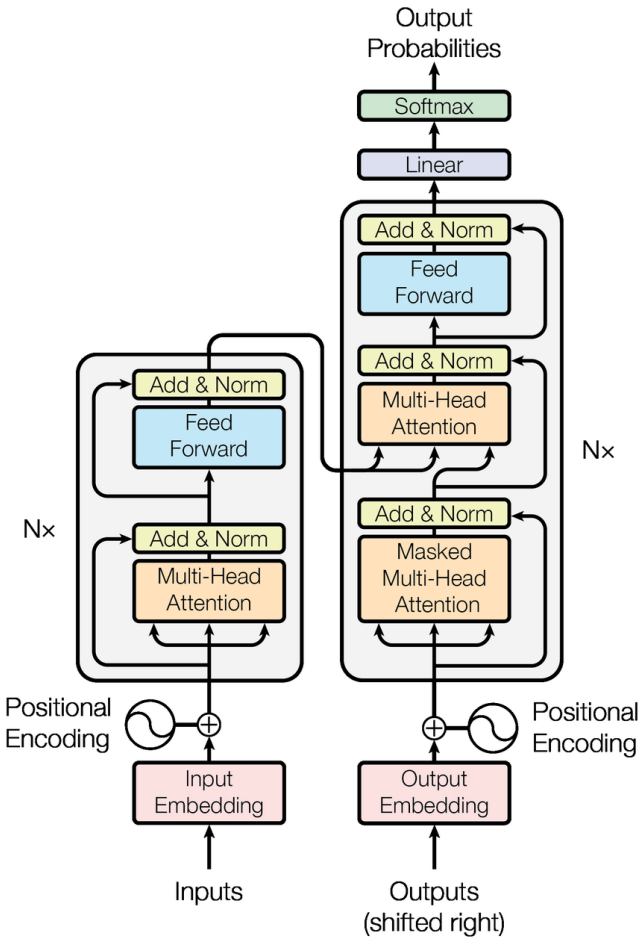

Figure 1: The architecture of the Transformer model, consisting of an input embedding layer, multiple Transformer blocks, and a final output layer. Graphics by [Vaswani et al., 2018].

and inductive bias to effectively learn from graph-based data. Here, the term 'inductive bias' refers to the set of assumptions that the Graphormer model makes to predict outputs for unseen data. These assumptions are introduced through the model architecture itself and guide the learning process towards solutions that are likely to work well with the kind of data the model will encounter. This in particular is important, as it has been evident, that using the blank Transformer model does not work sufficiently well for graph data as described by [Ying et al., 2021]. It is notable that [Kim et al., 2022] introduced a way to input graph data into a default Transformer that seems to work reasonably, albeit never reaching state of the art.

3.3.1 Node Embedding

Graphormer applies a distinctive approach to node embedding. As depicted in Figure 2, instead of using a single embedding, each feature of a graph node (e.g., atomic number) is embedded separately. These feature embeddings are then combined for each node to form the final feature representation. This strategy offers two key advantages. Firstly, it reduces the number of required embeddings. Secondly, it allows loss information to propagate more uniformly into the embeddings, as each node feature's embedding is updated whenever that feature is present, regardless of other node attributes. This technique is also applied to edge features, as discussed in Section 3.3.2.

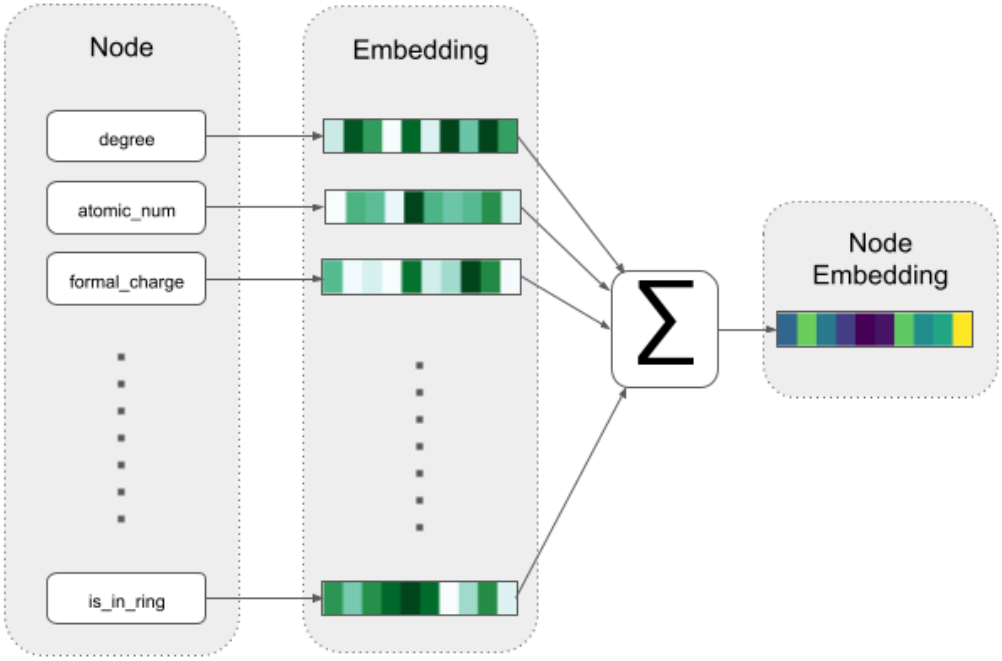

Figure 2: Creation of the node embeddings.

3.3.2 Attention Bias

The attention bias mechanism, used within all multi-head attention blocks, signifies a key alteration to the Transformer architecture in the Graphormer model. This mechanism comprises two primary elements.

The first part of the attention bias involves embedding the `spatial_pos` matrix $\mathbf{S}$, to a different floating-point number for each attention head. This provides the model with a sense of the spatial relationship between the nodes in the graph.

The second part of the attention bias depends on the choice between two methods: "single-hop" and "multi-hop". The "single-hop" method embeds each edge feature to a number for each edge, before averaging these edge feature embeddings for each edge, mirroring the process used for node embeddings as discussed in Section 3.3.1.

Conversely, the "multi-hop" method, as illustrated in Figure 3, while being more computationally demanding, allows for the consideration of multiple hops on the edge-graph from each node by using the `input_edges` tensor. This means that if a node has a neighbor that is two bonds away, an edge-feature-encoding is created for this connection as well, up to a predefined limit `multi_hop_max_dist` on the number of hops. These multi-hop edge feature embeddings are then averaged, akin to the single-hop method and the node embedding process.

The resultant tensor undergoes a linear transformation, also known as batch matrix multiplication, with a weight tensor. This operation implies that each feature embedding dimension is matrix-multiplied by a unique matrix, based on the distance between the connecting nodes for this embedding. Despite the computational demand, especially during the training phase, this method allows all gradients to propagate through the weight matrix for each distance, a particularly useful architectural property when dealing with rare edge types.

Finally, the attention bias is scaled linearly by the distance of the edges, ensuring a lower impact of edges on nodes that are further away. The attention bias values for all pairs of nodes that are farther apart than a specified threshold are set to negative infinity, effectively ruling out their influence in the multi-head attention.

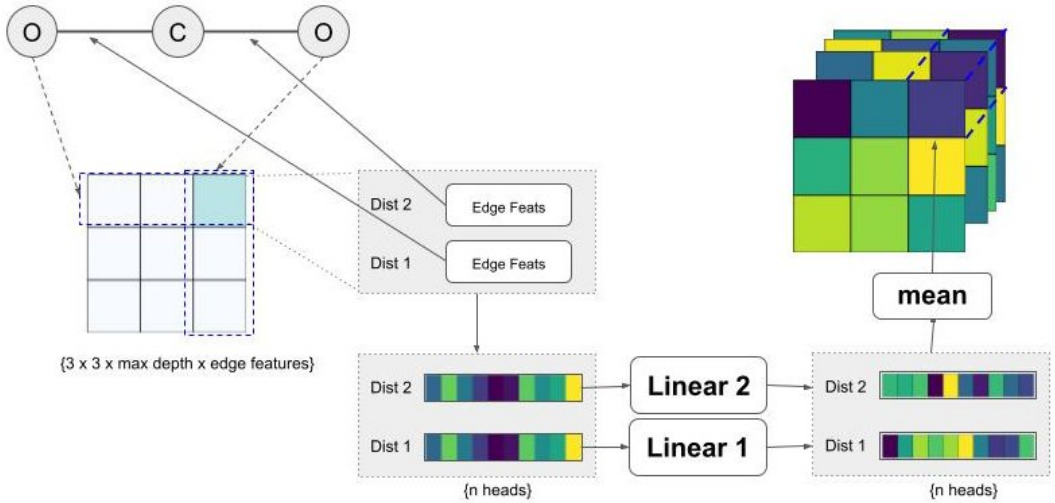

Figure 3: Process of creating the multi-hop attention bias tensor for the Carbon Dioxide molecule. Text in curly brackets stands for dimensionality. The two linear operations are without a bias term and equivalent with a batch matrix multiplication.

3.4 Graphormer3D

[Shi et al., 2023] modified the Graphormer to a version that can utilize three dimensional data to also use this architecture for problems where edges do not exist opening it to the field of point clouds. The 3D version works with the euclidean distances between the graph nodes and with the atom types to produce meaningful edge-representations and also the Graphormer specific `attn_bias` tensor. Moreover, the node embeddings are constructed from the atom types combined with information about the connection between a node to all other nodes. However, a pitfall of the 3D version is that it does not directly have the ability to utilize edge features that are known at all, so the edge features are exclusively dependent on 3D coordinates and node features.

3.5 Pretraining in Machine Learning

Pretraining is a well-established practice in many areas of machine learning where a model is first trained on a large dataset before being fine-tuned for a specific task. This technique can be particularly useful when there is a scarcity of labeled data for the specific task, as it allows the model to learn useful features from the pretraining data that can speed up convergence during the fine-tuning phase and improve performance in particular also for generalization. Pretraining can be performed in a supervised manner, using labeled data, or in an unsupervised manner, using unlabeled data. A notable subtype of unsupervised pretraining is self-supervised learning, where the data itself is used to generate labels for training. Self-supervised learning can be particularly beneficial as it leverages large amounts of unlabeled data, which is often more readily available than labeled data.

In supervised pretraining, the model is trained to predict provided labels, which can be beneficial even if the labels are not directly related to those present during fine-tuning, as long as the data distributions are similar.

In contrast, unsupervised pretraining, and more specifically self-supervised learning, does not rely on provided labels. Instead, it uses the data itself to generate labels for training. Known examples of this approach include Masked Language Modeling (MLM) and Next Sentence Prediction, popularized by [Devlin et al., 2019], as well as Contrastive Learning, Causal Language Modeling, and Replaced Token Prediction. In the domain of graph data, several works have explored self-supervised methods for pretraining, such as the survey by [Xie et al., 2023], the approach by [Zaidi et al., 2022], and the Transformer-based approach by [Rong et al., 2020].

3.5.1 Masked (Language) Modeling

In Masked Language Modeling (MLM), a percentage of tokens in a given sequence are randomly masked or replaced with mask tokens. The model's objective during pretraining is to predict the original token of the masked ones. In most cases, the model attempts to predict the index of the masked token's embedding in the embedding layer. However, there are also variations on this theme. For instance, in Vision Transformers, a method first introduced by [Dosovitskiy et al., 2020] and further developed by [Vaswani et al., 2021], the model tries to predict the true input embedding of the masked region. It's important for the case of predicting the true input embedding to apply regularization measures to prevent a phenomenon known as mode collapse, where all embeddings collapse to the same value, thereby nullifying the predictive objective. Moreover, [Rong et al., 2020] used a similar approach as MLM in their GT GROVER, where they mask a random subgraph of variable size and the model is tasked to predict contextual properties of this subgraph.

3.5.2 Denoising

Denoising is another form of self-supervised learning where random noise is added to the input of the model, and the model's objective is to either predict the noise that was added or recover the original, noise-free input. As a side note, the well known denoising autoencoder by [Vincent et al., 2008], a popular unsupervised type of neural network architecture also utilizes self-supervised learning.

In the context of 3D molecular data, a novel approach by [Zaidi et al., 2022] involves adding Gaussian noise to the coordinates of atoms within a molecule and having the model predict the added noise. Their method, applied to the Graph Network-based Simulators (GNS) architecture [Sanchez-Gonzalez et al., 2020], perturbs the positional information of the input in a rotationally invariant manner, which is possible because of how GNS are designed. The noise level is controlled by a hyperparameter. This method has been shown to correspond to modeling a molecular force field, providing a sound theoretical foundation for the task. Notably, their method achieved strong results on common benchmarks such as QM9.

3.5.3 Motif Prediction

Another self-supervised pretraining approach is the prediction of motifs, or recurring patterns, as used by [Rong et al., 2020]. In their work, motifs are extracted from the graph beforehand with tools such as RDKit, a common chemoinformatics software for working with molecule data, and the model is tasked to predict those motifs given the graph as input. This approach is different to the ones above, as instead of node-level predictions, here graph level predictions are made, making it potentially a better fit for pretraining for later graph level prediction fine tuning tasks.

3.5.4 Survey on Self-Supervised Learning in GNNs

For a more detailed review of self-supervised methods in GNNs, which share many similarities with GTs, the reader is referred to the survey by [Xie et al., 2023]. They categorize self-supervised methods for GNNs into contrastive and predictive methods, providing a comprehensive overview of the current state of the art in this area. It is reasonable to assume that many of these methods can be adapted for the framework of Transformers, given the shared focus on graph-structured data and also the fact that Transformers are actually just GNNs in a special setting as described by [Chaitanya, 2020] in a blog post.

4 Methodology

4.1 Data Collection

The datasets used for pretraining are PCQM4Mv2 and PCBA and for finetuning the Tox21, QM9 and ZINC datasets provide a good way for figuring out the approaches' performances. Tox21 is a popular dataset within the field of drug discovery and thus was an easy choice for a classification task. Furthermore, QM9 and ZINC are the two datasets with regression tasks that seem to be used the most in recent publications and thus its nicely possible to have a fair comparison. For easier comparison of the datasets Table 1 and Figure 4 show the main differences.

Table 1: Comparison of Different used Datasets

Dataset	# Mols.	Mean #	Q1 #	Q3 #	DFT	Task
Tox21	7,831	18.6	11	23	No	multilabel cl.
Tox21 Original	12,707	19.3	11	24	No	multilabel cl.
QM9	130,831	16.0	18	20	Yes	multiple reg.
ZINC	12,000	23.2	20	26	No	reg.
PCBA	437,929	26.0	22	30	No	multilabel cl.
PCQM4Mv2	3,378,606	29.5	25	34	Yes	multiple reg.

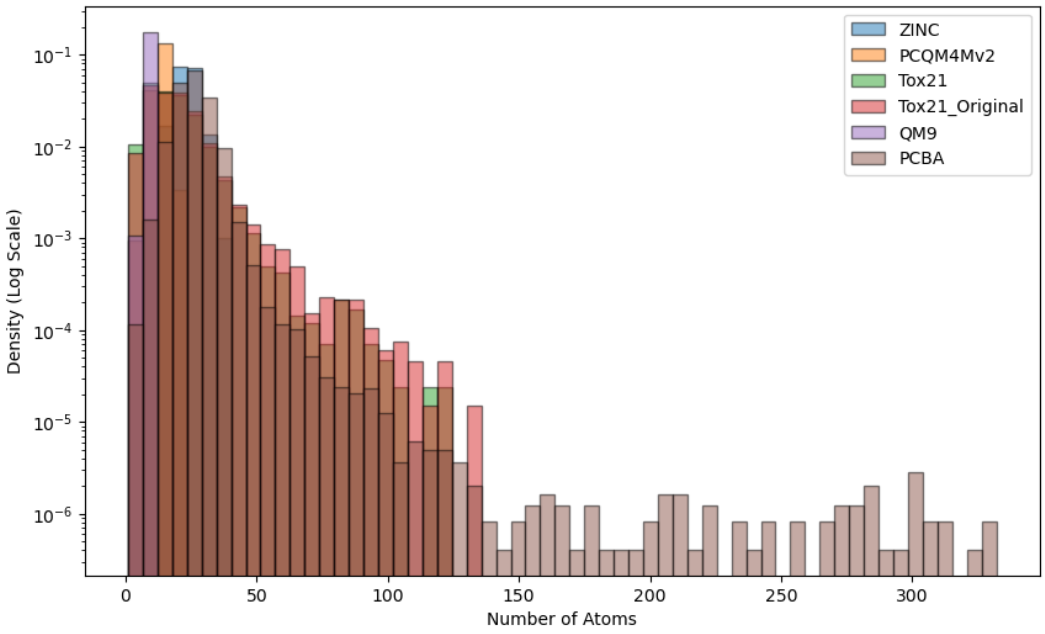

Figure 4: Histogram of the density of number of nodes n of molecules across the used datasets. Most of the datasets mostly stay well below 150 atoms per graph with the exception of PCBA which has a significant amount of molecules even with 300 atoms.

PCQM4Mv2 [Nakata and Shimazaki, 2017] is one of the biggest openly accessible datasets for molecule data with around 3.3 million molecules including 3D information for each atom that was generated using density functional theory (DFT). [Hu et al., 2020] have previously defined a meaningful machine learning task on this dataset which involves predicting the HOMO-LUMO gap molecular property based only on the basis of the smiles string. However, the dataset including 3D information for its train set has since been used by multiple authors Zaidi et al. [2022], Ying et al. [2021], Wang et al. [2022] for the purpose of pretraining a model for the reason that it is quite large and thus offers great generalization.

PCBA [Wu et al., 2018] is another quite large scale dataset containing molecule data. The dataset contains some molecules that are much larger than in the other datasets with the biggest having 332 atoms, excluding hydrogen atoms, as can be seen in Figure 4. The dataset is available for download from DeepChem by [Ramsundar et al., 2019].

Tox21 [Wu et al., 2018] is a smaller dataset for toxicity prediction on molecules with 12 distinct binary targets. Those targets include nuclear receptors and stress response pathways within the human body. There is also a bigger, original version of this dataset with around 5,000 more molecules which was used in the Tox21 challenge [Unknown, 2014]. It has been a bit ambiguous which of those two dataset versions is referred to in other works so both are used here. This orignal Tox21 dataset is in this work always referred to as "Original Tox21" and the smaller version from Molecule Net by [Wu

et al., 2018] is termed just "Tox21". It is worth mentioning that while the original dataset comes with a predefined train-test-validation split, the other version does not.

ZINC The ZINC dataset [Dwivedi et al., 2020] is a widely utilized, smaller subset of the comprehensive ZINC database [Irwin et al., 2020]. It's often employed to benchmark the performance of machine learning models that operate on graphs [Ying et al., 2021, Kim et al., 2022, Rampasek et al., 2022]. The task associated with this dataset involves the prediction of constrained solubility, represented as $y = logP - SAS - cycles$, through a regression model. In this expression, $logP$ refers to the partition coefficient between water and octanol, SAS is an abbreviation for the synthetic accessibility score, and $cycles$ signifies the count of cycles containing more than six atoms.

QM9 [Ramakrishnan et al., 2014, Ruddigkeit et al., 2012] is a molecule-dataset for predicting 19 quantum chemical properties with regression. The dataset contains, compared to the other datasets above, comparably small molecules, making it particularly favourable for GTs since the $\mathcal{O}(n^2)$ complexity has little impact here compared to GNNs.

4.2 Data Preprocessing for Pretraining Graph Transformers

The preprocessing phase ensures data uniformity by transforming all datasets, regardless of their diverse formats (e.g. sdf, csv, pickle), into the 'arrow' format compatible with the Hugging Face Transformers framework. This decision was primarily driven by the availability of established preprocessing pipelines within the Hugging Face ecosystem tailored for such data, as implemented by [Fourrier, 2023]. Moreover, during this process certain molecular properties are extracted via the help of tools from the RDKit library [Landrum, 2023] inside the `process_molecule` function. Atomic properties include atomic number, chirality, degree, formal charge, total number of hydrogen atoms, number of radical electrons, hybridization, and whether the atom is aromatic or is in a ring. Bond properties include bond type, stereo, and whether the bond is conjugated. Moreover, the number of atoms and the sparse adjacency matrix called `edge_index` are extracted.

A crucial component of the datasets used in this project is the presence of valid 3D conformers. For instance, the PCQM4Mv2 dataset comes with 3D information generated by Density Functional Theory (DFT), a quantum mechanical modelling technique described by [Hohenberg and Kohn, 1964]. DFT is a quantum mechanical modelling technique for providing a detailed picture of molecular properties. It tries to describe the electronic structure of a system. However, due to its complexity, DFT methods can be computationally expensive and time-consuming, particularly for larger molecules. The complexity of DFT scales with $\mathcal{O}(n^3)$ relative to the number of electrons in a molecule, making it feasible for smaller molecules but impractical for larger ones, particularly those containing more than 200 electrons.

To circumvent this computational challenge, this project employs a more cost-effective method available in the RDKit library for the datasets that do not already come with DFT generated 3D data, like ZINC or Tox21. The RDKit algorithm uses a two-step process: it initially generates a 3D conformation using a distance geometry algorithm, followed by further optimization via a Universal Force Field (UFF) approach as introduced by [Rappe et al., 1992]. UFF provides a simpler approximation of molecular geometry compared to DFT. It treats atoms and bonds as basic particles and springs, and is designed to be applicable to all molecules, including inorganic compounds. This makes UFF highly versatile and significantly faster than DFT. However, the trade-off is that UFF's approximations may be less accurate than the detailed picture provided by quantum mechanical methods like DFT.

For datasets not originally equipped with DFT-generated 3D data, it was considered to create two 'arrow' versions of the datasets, one with 3D and one without. This is because the local 3D generation method using RDKit cannot create conformers for all molecules. However, it was decided that it would offer better comparability between Graphormer and Graphormer3D if the same subset of the dataset would be used, so it was not implemented.

4.2.1 Employment of Target Normalization for Regression Tasks

For regression tasks in datasets like QM9 and ZINC, target normalization is generally beneficial. However, in the case of the ZINC dataset, normalization was not applied because its targets are already reasonably normal, exhibiting a mean of 0.015 and a standard deviation of 2.011. Furthermore, the

implementation of target normalization was only incorporated later during this project's progression, when many experiments of ZINC were already successfully executed.

Target normalization for the QM9 dataset was executed due to the significant difference in scales across its targets. The standard deviation of these targets spans a wide range, from a minimum of 0.02 to a staggering maximum of 1813.79. To account for this disparity and facilitate analysis, all the targets within the QM9 dataset were scaled to attain a mean of 0.0 and a standard deviation of 1.0. This method ensures a consistent standard of comparison, enhancing the accuracy of regression tasks.

4.3 Used Architecture

The decision to utilize Graphormer by [Ying et al., 2021], detailed in Section 3.3, was influenced by several factors. Primarily, it is included as a ready-to-use implementation in a renowned and extensively used framework, streamlining its integration into this project. Graphormer constitutes a significant representative of GTs, a model class that finds limited representation in extant literature. It is noteworthy for being the pioneer model that effectively processed graph data, marking a considerable advancement in the field. Despite not being the latest state-of-the-art model, Graphormer's performance continues to compete effectively, thereby striking a balance between performance and ease of use.

The utilized Graphormer implementation is grounded in the Huggingface Transformers framework [Wolf et al., 2020], potentially the most comprehensive machine learning library for Transformer models, albeit primarily for NLP applications. The library's recent release (version 4.27.2) incorporates the Graphormer variation as its inaugural GT model.

Nevertheless, the application of Huggingface Transformer's Graphormer implementation was not without challenges. Replicating the results from Ying et al. [2021] was initially arduous due to some significant yet easily overlooked bugs in the current Huggingface Transformers implementation. Furthermore, certain sections of the code, including parts of the original implementation by [Ying et al., 2021], were suboptimal or problematic for Graphormer. The ensuing sections describe some of the encountered issues, along with potential solutions where available.

4.3.1 Dropout Inconsistencies

The `attention_dropout` hyperparameter available in the `GraphormerConfig` remains unused, despite its availability for setting. Instead, incorrectly the the `dropout` hyperparameter of the configuration is used within the `GraphormerMultiheadAttention` class.

Moreover, the original implementation did have a separate dropout hyperparameter `activation_dropout` which should be used in the dropout module after the activation of the first linear layer in each Graphormer layer. This hyperparameter, however, is absent in the Huggingface implementation, where the `dropout` hyperparameter is mistakenly used in its place. As a result, `dropout` becomes the single determinant for all dropout probabilities in the Graphormer model during training. A review of Huggingface's unit tests reveals that `activation_dropout` was set in one of the tests, but slipped through due to `kwargs`, suggesting an oversight rather than a deliberate choice. Those problems are rectifiable by correctly aligning the respective dropout hyperparameters, a resolution demonstrated in my implementation.

4.3.2 In- and Out-Degree Redundancy

Both the Huggingface and original Microsoft implementations by [Ying et al., 2021] utilize the in-degree and out-degree features of a node within the `GraphormerNodeFeatureEncoder` class. These features are inferred by summing the adjacency matrix $\mathbf{A}$ over the second and first dimension, respectively. However, this appears to be unnecessary as the in-degree and out-degree counts, which are equivalent for molecular data, can be seamlessly added to the `node_feat` matrix. Furthermore, the corresponding tutorial of the Graphormer implementation by [Fourrier, 2023] in Huggingface already includes the degree count as part of the dataset's node features, suggesting redundancy in the original approach. Therefore, without any loss in quality, it is feasible to eliminate all instances of in- and out-degree, thus enhancing efficiency and readability, as demonstrated in my implementation.

4.3.3 Attention Bias

The `preprocess_item` function, responsible for the final preprocessing step before data is passed to the data collator, creates an attention bias tensor exclusively composed of zeros. In the data collator, a tensor of the same name is again created, also only containing zeros. This tensor then has values set to negative infinity at positions where the spatial position tensor contains values above the `spatial_pos_max` hyperparameter. Given that the spatial position tensor is passed to the model, this step appears wasteful in terms of both computation and memory. A more efficient approach would be to perform this operation on the GPU.

The possible rationale behind this implementation might be to have a distinct attention bias tensor that can externally influence the communication between nodes within the Graphormer multi-head attention. However, due to the absence of accompanying documentation, this not only renders the code harder to decipher but also needlessly expends resources.

Additionally, within the `GraphormerGraphAttnBias` class, if the attention bias contains only 0 or negative infinity (as would be the case when using the `preprocess_item` function), the addition at the end of the `forward` function becomes redundant. This is because the entries would already be negative infinity, rendering the operation unnecessary.

4.3.4 Extreme Memory Waste with the Input Edges Tensor

In the `preprocess_item` function, a large `input_edges` tensor is constructed for each graph. This tensor, of $\mathbb{N}^{h \times n \times n \times j}$, has several limitations. One of the primary issues is the extreme memory waste when $h = \max(\mathbf{S})$, with $\mathbf{S}$ being the `spatial_pos` matrix. Several factors contribute to this inefficiency:

Firstly, the `input_edges` tensor is only utilized when the `edge_type` hyperparameter is set to `multi_hop`, rendering its processing superfluous in other scenarios. This can be circumvented by verifying the `edge_type` within the `preprocess_item` function.

Secondly, even when `multi_hop` is in use, the tensor is employed only up to a hyperparameter `multi_hop_max_dist` within the model, which defines the maximum number of hops. This effectively renders all dimensions after `multi_hop_max_dist` in the `input_edges` tensor unnecessary. To address this, the `input_edges` tensor can be truncated in the `preprocess_item` function to a maximum of `multi_hop_max_dist` along the first dimension. This modification was implemented in my version of the code, directly in the Cython script that is originally used to create the `input_edges` tensor. Cython is a programming language that is a superset of Python, designed to give C-like performance with code that is written mostly in Python.

Lastly, if the input graph is disconnected (i.e., if two nodes in the graph do not have any connecting path), the original code automatically assumes h to be 510, resulting in an unnecessarily large tensor. This issue can be rectified by removing all entries larger or equal to 510 when determining h.

4.3.5 Loss Calculation

The current Graphormer implementation for classification and regression tasks in the Huggingface Transformers always requires each sample's label to be contained in a list in the dataset. This holds true whether the task is intended for a single target or multiple targets. In scenarios involving a single target, this requirement introduces unnecessary complexity and redundancy, potentially causing confusion for new users and contributing to data processing inefficiency.

Additionally, the original Huggingface system does not support multiple regression tasks. This limitation inhibits the usage of some datasets, such as QM9, which require the simultaneous prediction of multiple regression targets. The lack of support for multiple regression necessitates users to either significantly modify the existing implementation or split their regression targets into multiple separate tasks. Both approaches increase the user's burden and decrease the overall user-friendliness and efficiency of the framework. My implementation introduces the capability of handling multiple regression tasks within the framework.

In my implementation, this issues are addressed by adding a hyperparameter that determines the type of task, either "classification" or "regression". This makes the type of task that will be used less ambiguous and makes the implementation more robust.

Table 2: Minibatch Loading Speeds (Seconds)

Processing	Random	Sequential
None	5.59±0.30	0.06±0.01
On the Fly	7.05±0.29	0.37±0.05
On Disk	33.76±1.21	1.48±0.35

4.3.6 Dataset Loading Issues

A major limitation associated with the by [Fourrier, 2023] suggested approach of using the Graphormer implementation with an arrow dataset, is the slow processing speed. This slowdown is likely due to the data structure used. Each sample contains lists of lists of varying sizes, such as `node_feat` for node features. This structure precludes saving the data as an array, resulting in inefficient indexing. This becomes even more pronounced when saving the processed dataset that includes the large four-dimensional `input_edges` tensor. Furthermore, converting the dataset's features to NumPy format using the `set_format` function is also quite slow.

I evaluated the efficiency of various data loading techniques by empirically examining three distinct data processing methods on a Hard Disk Drive (HDD): 'None', 'On the Fly', and 'On Disk'. Table 2 provides an overview of the results obtained over ten trials, depicting the average time taken to load a minibatch of size 256 under 'Random' and 'Sequential' processing conditions.

The 'Sequential' processing loads a minibatch as a chunk of increasing indices (e.g., from index 200 to index 456), while 'Random' processing chooses a random data index within the range of the dataset size for every sample loaded. The 'None' category represents scenarios where data is directly loaded into memory without any processing. The 'On the Fly' method loads unprocessed data into memory (similar to 'None'), then prepares it for training using the `preprocess_item` function. The 'On Disk' method involves storing preprocessed data on disk and directly loading it into memory for model training, bypassing any need for additional processing during data collation. Despite the expectation that preprocessing data would increase efficiency, the results displayed in Table 2 contradict this assumption.

The findings reveal that random loading significantly slows down the processing speed, regardless of the chosen processing method. However, relying solely on sequential loading during model training is undesirable, as it violates the assumption of independent and identically distributed data, potentially introducing bias into the model.

Three options to tackle the issue were considered, as it turned out to be infeasible to load the data randomly from the disk. The first two options require loading of the entire used dataset into the memory, which depending on the dataset size is quite a bit. The first option there is to load the dataset in "unprocessed" format, meaning that large tensors like the `edge_input` would not be present yet. This has the advantage that is consumes a comparably small amount of memory but comes with the flaw that this will need to be computed during data collation with the `on_the_fly` option within the data collator. The second approach is to already process the data beforehand which can be quite memory intensive because of the large tensors that are generated but will be very fast during data collation, moving the processing bottleneck to the GPU. The last option, that was only considered but not implemented, is to use an entirely different data format on the disk. Ideas were saving the data in MySql format or similar but due to the other two options working sufficiently well this was not tested.

4.3.7 Limitations of Node Features in the Current Implementation

The existing implementation operates on the premise that each node feature possesses an identical number of potential options, a characteristic also mirrored in the number of model parameters. This assumption, however, lacks practicality, especially concerning molecular data. In the molecular realm, the variety of options related to the atomic number greatly outstrips any other node used feature. Specifically, in the datasets employed, the maximum atomic number encountered is 83 which is Bismuth, compared to a mere 11 for all other node features, with some of them even being binary.

This discrepancy suggests that a significant proportion of parameters, in my case 46,080, are superfluous. Furthermore, the original Huggingface implementation did not provide an efficient method to set the `single_embedding_offset` hyperparameter, which is crucial for the `preprocess_item`

function for preprocessing data. Consequently, all my experiments were conducted based on the default value of the processing function, i.e. 512.

Therefore, while the Graphormer models displayed 871,553 parameters for me, the relevant count, when adjusted to 85, which incorporates two indices for padding, would be 564,113 parameters, solely accounting for the nodes. The padding is necessary for having additional artificial tokens like the padding and the mask tokens for pretraining. It should be emphasized that a similar adjustment is also pertinent for the edge attributes.

With optimal efficiency, the total parameter count for the model could be reduced to approximately 500,000, making it suitable for the ZINC500k benchmark, which was the intention of this study. Although these adjustments did not influence performance, and I thus opted not to implement them only partially during my experiments, their importance is obvious.

4.3.8 Memory Issues

Handling large datasets, such as the PCQM4Mv2 used for pretraining, revealed a potential memory leak somewhere in the pipeline. Memory leaks can result in escalating memory usage over time, potentially causing the program to exhaust memory resources and crash. Furthermore, they can impede program performance and the overall system due to the continuous consumption of memory resources. While I was able to successfully conduct my experiments despite this memory leak issue, it is critical to highlight this issue for reproducibility and as a consideration for future research.

A comprehensive analysis was undertaken to identify the source of this leak; however, the exact cause remained elusive. The memory consumption in the data loader threads exhibits a linear increase over epochs, but it resets after each epoch, when the dataloader threads are recreated. This pattern suggests a potential bug in the backend components of NumPy or PyTorch, making the source of the leak challenging to identify.

While running experiments on the available servers, the memory size was sufficient to support the experiments, even with the memory leak. However, the memory usage peaks at around 95GB when pretraining with the PCQM4Mv2 dataset, where the epochs are the longest. This finding underscores the necessity for efficient memory management, especially when working with large datasets. It also emphasizes the importance of further investigation to identify and rectify the source of this potential memory leak, enhancing the efficiency and stability of the Graphormer implementation.

4.3.9 Size Mismatch when using Multi-GPU

Sometimes, seemingly at random when using a multi-GPU setup, the Huggingface trainer will crash with the error of a size mismatch in the model output. Interestingly, this only happened on some of the datasets but not on others. For example for ZINC it seems to work as expected but for Tox21 Original this error often occurs. It can be easily avoided by restricting to a single GPU but it slows down training a bit. The issue was investigated but it was not possible to find the source of it, especially since it is quite difficult to reproduce, as it happens at random rates apparently and is probably somewhere hidden in the logic of the Huggingface trainer. Thus, for my implementation I restricted myself to using single GPUs for the problematic datasets.

4.3.10 Contributions to the Community

Aiming to nurture the open-source ecosystem and build upon the collective knowledge base, my work also extends to improving the machine learning community's resources in the form of pull requests to the Huggingface Transformers GitHub repository. This not only involves identifying the challenges encountered during the usage of their Graphormer implementation, but also conceiving practical solutions to these problems.

These solutions tackle a range of issues, which I have detailed in the previous sections, from ironing out inconsistencies in dropout application, eliminating redundant data processing, optimizing memory usage and node feature handling. Each of these modifications are devised with an eye towards not only enhancing the model's performance and efficiency, but also making it more user-friendly and accessible for the broader scientific community.

4.4 Graphormer 3D Hypothesis

While developing the methodology for this study, an intriguing hypothesis formed: The 3D version of Graphormer as proposed by [Shi et al., 2023] could, using distance information, accurately approximate the edge types of a molecule used in the standard Graphormer. In addition, this method would be computationally more efficient since there would not be a need for computing the expensive shortest path matrix and to construct the large input edges tensor, as well as needing those in the data collator. This hypothesis was based on the idea that in molecular structures, 3D distances often hold vital information, and this can be converted into edge types. This conversion could mimic the edge calculations of the non-3D Graphormer without the computational intensity.

Therefore, the choice was made to implement and test the 3D version of Graphormer in the Huggingface environment, which was not done yet, as this approach could potentially achieve comparable results while reducing the computational burden. This methodology decision not only informs this study but also contributes valuable insight for future work in the field, highlighting the potential of leaner, more efficient models without sacrificing performance.

The 3D version of Graphormer, introduced by [Shi et al., 2023] and available on the official Graphormer GitHub repository, originally did not provide the option to include more than one node feature in the model. This limitation is a disadvantage compared to the standard Graphormer for two reasons:

1. The node embeddings themselves are less informed than with the usual Graphormer as essentially then we only have an embedding as we know if also from NLP tasks, not utilizing the method described in Section 3.3.1.
2. In the standard Graphormer3D, the edge types are uniquely created using the atom types from the start and the finish node. However, all other node information is disregarded in this creation process.

I addressed it by adopting a more evolved approach. Specifically, two distinct node embeddings were created in a similar way to the non-3D Graphormer, acting as the start and finish points of the edge. These embeddings were then broadcast and summed element-wise to produce an edge embedding between all nodes. This modification enabled the Gaussian layer to project these embeddings to a one-dimensional space, facilitating similar operations to the original 3D implementation while allowing more than one node feature. Furthermore, for the standard node embedding I went with using the start point node embedding for this. Generally this approach allows for a more rich Graphormer3D implementation that can, as can be seen in the experiments in Section 5.4.2, outperform the original Graphormer3D by a large margin. In all further mentions of 'Graphormer3D', my modified version is usually meant, instead of the original implementation by [Shi et al., 2023], unless stated otherwise.

4.5 Training Procedure

The training procedure encompasses several key elements that ensure the efficiency and reproducibility of the machine learning models. These include specific choices of hyperparameters, techniques for seeding to promote consistency, an evaluation strategy tailored to each task, and a robust approach to data splitting. Each of these elements is interconnected and plays a significant role in model performance. In the following subsections, I detail the considerations and methodologies involved in each aspect of the training procedure.

4.5.1 Hyperparameter Choices

For the specific hyperparameter setting a similar setup to the one from [Ying et al., 2021] used for the ZINC dataset as those are already verified to work. Moreover, in all the experiments the parameter count is restricted to be only around 500k to make it easier comparable to other works, as the ZINC-500k variation is a common one. The structural model hyperparameters for this setup are an embedding dimension of 80, eight attention heads and 12 Graphormer layers. Moreover the multi hop maximum distance is set to five and the encoder uses a layer norm before the encoder block. A dropout of 0.1 is used in the multi head attention, the fully connected layers and the end of the encoders but not at the activation of the encoder. The activation function used is the Gaussian error linear unit (GELU). For the optimizer Adaptive Moment Estimation with Weight Decay (AdamW) with the hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-8}$, weight decay $= 0.01$ is used. Moreover,

a linear scheduler with 40,000 warm up steps, 400,000 maximum steps and a maximum learning rate of $2e^{-4}$ is used. The learning rate with an exception on the Tox21 Original dataset with the Graphormer model without pretraining, where a learning rate of $2e^{-5}$ was used due to appearing instability of the model parameters when using the higher learning rate. The mini-batch size is set to 256 across all experiments. Finally, for pretraining a masking probability of 0.1 is set with the tweak that at least one node is masked for each input graph. There would be no point in inputting a graph without any masked nodes as it would not contribute to the loss, however one could also consider to remove a graph without any masked nodes from the mini-batch instead. The loss is averaged across all masked nodes, which naturally gives larger graphs more impact on the loss which might still not be ideal. For the 3D implementation of the graph, the dropout rates were marginally increased to 0.15 in response to evidence of over-training in the dataset. This adjustment to the dropout rate appeared to enhance performance by providing regularization. Additionally, the output size of the Gaussian Layer was set to 128, in accordance with the default configuration suggested by [Shi et al., 2023].

While it was considered during the experiments to do hyperparameter fine-tuning, it was decided that this is out of the research scope of this work and would not reveal much information related to pretraining. Moreover, running doing fine-tuning is very computationally demanding and given that a single training run already takes a day it seemed unreasonable to implement this. Furthermore, the used hyperparameters are based on the ones officially used by the inventors of the Graphormer architecture to the best of my knowledge and I rely on that this choice was backed up by enough thought. Finally, while fine-tuning could potentially reveal better absolute results in terms of loss, more of interest for this research is the difference between a model version that had pretraining and a version that was trained from scratch.

4.5.2 Training Loop

The training loop, which is responsible for passing the data from the dataset via dataloaders to the model, backpropagate the received loss and to do the logging evaluation results and checkpoints via weights and biases was done by using the Huggingface trainer implementation which incorporates all those functionalities. Finally in the end of the training the best model according to the chosen evaluation metric on the evaluation set is used to infer the results on the separated test set.

4.6 Seeding and Reproducibility

Reproducibility is a cornerstone in scientific research, extending to the field of machine learning as well. To ensure that the results of this research can be independently verified and reproduced, rigorous measures were taken to set consistent seeds across all utilized environments, namely Python, NumPy, PyTorch, and CUDA. Nevertheless, absolute determinism in machine learning experiments presents its challenges, largely due to the complexities introduced by multi-threading in data loaders and the use of a multi-GPU setup. Although theoretically, full determinism could be achieved by restricting the setup to a single dataloader and enabling `torch.backends.cudnn.deterministic`, such measures would significantly extend the training duration, rendering it beyond the feasible scope of this research project.

To further bolster the statistical reliability of the results, all fine-tuning experiments were conducted across ten different seeds. However, due to the extensive time requirements and computational resources involved, the pretraining runs were limited to a single run. Specifically, each of these runs necessitated over four days of processing time, primarily due to the constraints of storing the fully processed dataset in memory, given the extreme size of the input-edge tensors. Despite these challenges, every effort was made to maintain the highest standards of research integrity and ensure the robustness of the experimental results.

4.7 Dataset Splitting

In this research, the approach to dataset splitting was tailored to promote robust performance assessment and facilitate meaningful comparisons with prior studies. For the ZINC and the original Tox21 datasets, the predefined splits provided with these datasets were employed. Adherence to these established partitions supports consistency with previous works and enhances the comparability of my results with those of notable studies in the field, for instance by [Ying et al., 2021, Rampasek et al., 2022, Zhang et al., 2022, Kim et al., 2022].

For the remaining fine-tuning datasets, specifically Tox21 and QM9, I embraced a k-fold cross-validation strategy. This robust methodology aims to diminish the dependency of my model's performance evaluation on a particular data partition and to provide a more reliable appraisal of the model's generalization capability. Within this strategy, the data are segmented into k+1 subsets, with one acting as the test set. The other k subsets participate in k training runs, each using k-1 subsets for training and one for validation. The optimal model parameters are deduced based on validation set performance. The final test set is then assessed with the models from the k training runs, and the results are averaged to provide an aggregate performance estimate.

Beyond traditional partitioning techniques, more inventive methodologies for data splitting have surfaced in the field of molecular property prediction. For example, [Mayr et al., 2018] proposed a method known as cluster-cross-validation. This approach forms data subsets (or "folds") according to data clustering, aiming to improve the model's ability to generalize to novel laboratory measurements. Although these assays may be expensive in a wet lab setting, they become substantially more cost-effective when applied through machine learning.

It's important to note that while the implementation of advanced partitioning methods like cluster-cross-validation could theoretically enhance the rigor and generalizability of the model's performance, their adoption remains limited in the current literature. This factor, coupled with the substantial time and coding resources required for such implementations, influenced the decision to not utilize these methods in this research project. However, the lack of these sophisticated techniques in this study shouldn't overshadow their potential benefits. As the field progresses and these methods become more commonly employed, they could potentially evolve into a new standard, paving the way for more nuanced comparisons.

In this research, the strategy for data splitting was designed with robustness, practicality, and comparability with other studies in mind. As a single researcher undertaking this project, I firmly believe in the continuous evolution and refinement of these methodologies. Thus, I advocate for the exploration and adoption of improved data splitting techniques in future molecular property prediction research.

4.8 Evaluation

The evaluation metrics utilized in this research were carefully selected to align with the tasks' characteristics and with practices commonly accepted in the relevant literature. For regression tasks on the ZINC and QM9 datasets, the Mean Squared Error (MSE) was chosen as the principal evaluation metric. This choice is grounded on a robust theoretical foundation. Furthermore, the mean average error (MAE) was additionally implemented in the later course of the project to establish a stronger foundation of comparison to other methods in the research domain.

The Tox21 dataset, featuring a multi-label classification task with some missing labels, necessitated a distinct approach to evaluation. As such, the mean Receiver Operating Characteristic - Area Under the Curve (ROC-AUC) was employed as the primary metric, a decision echoing its prevalent use for tasks of this nature. Specifically, the "macro" version of ROC-AUC, which averages the measure for each label, was utilized to handle the multi-label nature of the task.

Given the ROC-AUC's non-differentiable nature, direct optimization during training was not feasible. In response, a slightly modified version of the Binary Cross-Entropy loss was implemented. This version accounts for missing labels by distributing the training signal evenly across all samples, regardless of the number of available labels per sample. However, as a result of an unintentional setup, this reweighted loss was used only for the Tox21 dataset, while the original version was employed for the Tox21 Original dataset. This accidental variation could potentially provide interesting insights into the impact of the loss function modification on the training process and final model performance.

In addition to these main evaluation metrics, supplementary monitoring of simple accuracy was implemented for the Tox21 task, providing a richer perspective on model performance and enhancing the interpretability of the results.

Moreover, continuous monitoring of these metrics during training is crucial for early detection of potential issues such as overfitting, thus enabling timely interventions and adjustments. Accordingly, throughout the training procedure, the Weights and Biases (wandb) platform was employed to systematically track all metrics. This tool facilitated a more nuanced comprehension of the model's

progress during training, and provided a robust framework for comparing the outcomes of training from scratch against those obtained through the use of a pretrained model.

4.9 Pretraining

As the main focus of this thesis is on pretraining GTs, two distinct pretraining approaches were employed, that both fall under the category of self-supervised representation learning. Firstly, masked (language) prediction, similar to how it is done in BERT and secondly 3D noise prediction similar to the work by [Zaidi et al., 2022] were utilized.

4.9.1 Masked Graph Node Prediction

This method, a simpler approach drawing inspiration from the BERT model, entails the replacement of a certain percentage, at a minimum one, of the input nodes for each graph with specialized mask tokens. It is important to note that in the current context, each node is associated not merely with a single feature, but with a plurality of graph node features; in this particular study, nine such features were identified. Accordingly, the mask token was implemented for each of these features.

During the pretraining phase, the primary objective of the model is to accurately predict the embedding index of each node feature independently. To facilitate this, each node feature is assigned a distinct prediction head, implemented in this study as a linear layer. Upon completion of the prediction process, the model's outputs are subjected to a softmax function, which serves to generate a probability distribution over the potential tokens.

Subsequently, the Cross Entropy Loss function is employed to compute the loss. This process essentially involves calculating the Negative Log-Likelihood (NLL) loss corresponding to the correct tokens. In this specific context, the Cross Entropy Loss and NLL are functionally equivalent, both serving to evaluate the performance of the model during the pretraining stage. By harnessing the aforementioned methodology, the model is equipped to efficiently learn the representations of node features, thereby laying a solid foundation for the subsequent tasks.

4.9.2 3D Noise Prediction

This methodology, inherently unique to Graphormer3D due to its necessity for three-dimensional data, demanded more careful deliberation compared to the Masked Graph Node Prediction. The principal challenge arises from the impracticality of accurately predicting the 3D loss vector, a methodology employed by [Zaidi et al., 2022]. The constraint primarily stems from the required model invariance to 3D rotations which presents a significant hurdle. Despite this, [Zaidi et al., 2022] were successful in their implementation, owing to the specific attributes of the model they employed, the GNS. This model, initially proposed for particle simulations, belongs to the GNN family.

In principle, it may have been feasible to devise a Transformer-style architecture that parallels GNS. However, such an endeavor would extend beyond the primary focus of this investigation, which centers around understanding the implications of pretraining and eschews the creation of entirely new architectural constructs.

Therefore, in lieu of predicting the 3D loss vector, an alternative metric was identified. This new target, the norm of the 3D noise, still retains the potential to assist the Graphormer3D in learning robust representations of graphs in a self-supervised manner. The 3D noise vector can be conceptually deconstructed into two components: its magnitude and direction. In this new methodology, only the magnitude, or the norm, is predicted.

A mathematical interpretation of a 3D vector $\mathbf{v}$ in Euclidean space can be represented as $\mathbf{v} = |\mathbf{v}|\hat{\mathbf{v}}$, where $|\mathbf{v}|$ signifies the magnitude (norm) of the vector, and $\hat{\mathbf{v}}$ denotes its unit direction vector. By focusing solely on predicting $|\mathbf{v}|$, the model effectively disregards $\hat{\mathbf{v}}$, thus ensuring complete invariance to 3D rotations.

Furthermore, it's worth noting that the magnitude follows a Chi distribution if the noise is drawn from a zero-mean Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I} \cdot \sigma_{\mathcal{N}})$, as it is in the case of the work of [Zaidi et al., 2022]. The Chi distribution has a mean of $\mu = \sqrt{2} \cdot \frac{\Gamma(2)}{\Gamma(1.5)} = 1.596$ and a standard deviation of $\sigma = \sqrt{6 - \mu^2} \cdot \sigma_{\mathcal{N}} = 1.858 \cdot \sigma_{\mathcal{N}}$.

To the best of my knowledge, this pretraining methodology represents a novel approach in the field. The strategy of predicting solely the magnitude of the noise may prove to be beneficial, as I suspect that the magnitude often encapsulates the essential information about the scale and impact of noise. Moreover, this strategy facilitates the learning of representations that are invariant to transformations like rotations, which may be crucial for certain applications. Additionally, given the common difficulty that GT and GNN architectures face when predicting rotationally invariant vectors for each node, this novel pretraining method could potentially pave the way for broader applications in this research field.

Of course, the ultimate utility and effectiveness of this approach are determined by empirical testing and its performance on downstream tasks as can be seen in Section 5.4.

5 Experiments and Results

The experiments conducted during this research have the primary goal to elaborate the performance difference between GT architectures with and without prior pretraining. Therefore the same experiments were conducted on pretrained and from-scratch versions of the same models (Graphormer, Graphormer3D). The two employed pretraining methods both fall under the category graph reconstruction as described by [Xie et al., 2023].

The conducted experiments gave vast insight into the importance of pretraining of GTs as will be discussed in this section. However, it is important to note that not all GTs are the same and in my experiments I mainly focused on the Graphormer approach and its 3D variation. Although similar results probably hold for very different GT approaches like [Rampasek et al., 2022] this still needs to be confirmed.

5.1 Experimental Setup

All computational tasks for this study were conducted on the student GPU-servers available from the Institute of Machine Learning at Johannes Kepler University in Linz. The servers run on Rocky Linux version 8.8 (Green Obsidian), a stable and reliable enterprise-level Linux distribution that is compatible with Red Hat Enterprise Linux and CentOS.

The machine is powered by an Intel Xeon CPU E5-2630 v4, boasting 20 cores that can reach frequencies of up to 3.1 GHz. The system also houses 125 GB of memory. This computational setup enabled effective distribution of the intensive tasks involved in the training and evaluation of the models. For graphical processing, multiple NVIDIA GPUs were available, each offering around 12 GB of memory.

Coupled with Tmux, PyTorch and Huggingface Transformers that were used to manage the computational tasks across the cores and GPUs, these resources provided an optimal environment for ensuring the accuracy and reproducibility of the model training and evaluation processes.

5.2 Pretraining Design and Considerations

The focus of this research was to examine the impact of pretraining on the Graphormer and Graphormer3D models, and as such, it was necessary to pay special attention to the design of the pretraining process. The hyperparameters selected for pretraining are equivalent to the ones used for fine-tuning. Moreover, a evaluation set was split of the pretraining datasets that measure the generalization of the models. The pretraining runs were set to go for 500,000 training steps which is for PCQM4Mv2 equivalent to around 38 epochs. For the masked graph node prediction described in Section 4.9.1 a masking probability of 0.1 was decided initially and a bit arbitrarily and kept as the results seemed sufficiently good. For the 3D noise prediction, detailed in Section 4.9.2, it was referred to the work of [Zaidi et al., 2022], who used a noise standard deviation of 0.04.

5.3 Experiment Design

The design of the experiments conducted for this study was specifically aimed at accurately assessing the impact of pretraining on the performance of Graphormer and Graphormer3D models.

Considering the stochastic nature of model training, the experiments were designed to be run multiple times to ensure the reliability of the results. Each experiment was repeated ten times with unique random seeds. In particular, the seeds were set in with the initial model parameters and with the dataloaders which decide the shuffle of the data. For the pretrained models of course the initial random seeds only were relevant for the classification head of the encoder. This approach diminished the impact of any individual anomalous run on the final reported performance, thus providing a more accurate and robust estimation of the models' true performance.

To assess the benefits of pretraining, two distinct setups were used for model training. In one, the models were pretrained using self-supervised methods as discussed in Section 4.9, before being fine-tuned on the task at hand. In the other setup, models were trained from scratch directly on the task. The same datasets and hyperparameters, as described in the methodology section, were used across both setups to ensure a fair comparison.

The progress of the training was monitored using the validation set, and early stopping was employed to prevent overfitting and to not waste compute. It was employed with a patience of 50,000 training steps and a threshhold of 0.0025 for ZINC and QM9 and 0.01 for Tox21 and Tox21 Original, which means that if the evaluation loss would fall within this threshhold of the previous best evaluation result again, the patience will be reset. The reasoning behind this strategy that incudes a threshhold stems from the observed evaluation loss curve. In particular, especially for the ZINC dataset, it seemed that even if there was a longer period of training with no improvement on the evaluation, if the general trend of the evaluation is to stay low, then there sometimes are still negative peaks in the loss that turn out to be the best result. Moreover, the evaluation was done every epoch initially, but later on it was decided that its enough to do it every 250 training steps which increased training speed significantly.

The performance of the models was then evaluated using a separate test set. This ensured an unbiased estimation of the models' generalization capabilities, providing a realistic assessment of how the models would perform on unseen data.

Overall, the experimental design was centered around accurately measuring the impact of pretraining on the models' performance, while ensuring the reliability and reproducibility of the results.

5.4 Results and Discussion

Pretraining of models significantly improves performance in both classification and regression tasks in all cases with few exceptions where only similar performance is obtained, as substantiated by the results presented in Table 3 and Table 4. This is primarily due to pretrained models beginning with parameters that are less random and more representative of the data, having been optimized through exposure to similar data.

Moreover, Figure 5 shows the strong impact of pretraining the model before training on downstream tasks. Not only, does the evaluation MSE of the metric go down significantly faster, but also the final performance of the pretrained models exceed the from scratch trained models by a large margin.

Notably, I observed a performance discrepancy between the two pretraining variants for 3D models. In regression tasks, noise pretraining outperformed mask pretraining, while the latter showed slightly superior results in classification tasks. While the standard deviations do indicate a possibility of these differences occurring by chance, the consistent pattern across tasks suggests a non-random cause. Two plausible explanations come to mind:

1. The respective loss functions used for pretraining might influence task performance. Noise pretraining utilizes a regression loss function, aligning well with the nature of regression tasks. Conversely, mask pretraining employs a cross-entropy loss function, which resembles the binary cross-entropy loss employed in classification tasks. This congruity in loss functions might partly explain the observed performance pattern.

2. The type of molecular properties targeted by the dataset could also be a factor. In datasets like ZINC and QM9, which aim to model chemical properties of molecules, noise pretraining demonstrated superior performance. This could be because noise pretraining models a force field, as suggested by [Zaidi et al., 2022], which might be making tasks that are related to the internal workings of the molecule easier. On the other hand, mask pretraining yielded better

Table 3: Classification Results (ROC-AUC)

Dataset	3D	From Scratch	Mask Pretrain	Noise Pretrain
Tox21	No	0.827±0.008	0.839±0.006	-
Tox21	Yes	0.818±0.012	0.836±0.006	0.827±0.015
Tox21 Original	No	0.785±0.006	0.789±0.013	-
Tox21 Original	Yes	0.790±0.014	0.806±0.013	0.788±0.009

Table 4: Regression Results (Mean Squared Error)

Dataset	3D	From Scratch	Mask Pretrain	Noise Pretrain
ZINC	No	0.156±0.117	0.080±0.018	-
ZINC	Yes	0.130±0.031	0.082±0.014	0.056±0.016
QM9	No	0.016±0.002	0.013±0.001	-
QM9	Yes	0.007±0.003	0.005±0.001	0.004±0.001

results on the Tox21 and Tox21 Original datasets, which focus on predicting non-physical external properties, specifically toxicity.

While these conjectures remain speculative, a relationship between these factors and the observed performance pattern is plausible and worth further investigation.

5.4.1 Study Limitations

The study's limitations must be acknowledged. Time and resource constraints inevitably limit the scope of experiments. Originally a bigger diversity of pretraining methods was planned but extensive time requirements of at least six hours for a single run rendered this impossible with the available resources, especially because of the intention of making ten differently seeded runs for each experiment, in order to establish results that are not impacted by randomness.

Less extensive experiments were conducted on the PCBA dataset, which can be attributed to the resource constraints and the particulary long pretraining times with PCBA, which is because of the large graphs in the dataset, some over 300 atoms. However, experiments done when using PCBA for pretraining suggest slightly weaker results then when pretraining on PCQM4Mv2 which could be because of the significantly smaller amount of graphs contained in PCBA also visible in Table 1.

5.4.2 Implications for the Graphormer3D Model

The modified Graphormer3D model, developed during this research, as detailed in Section 4.4, presents a promising approach to improving performance and should be further investigated by the research community. In an experiment over ten runs on the ZINC dataset without pretraining, the version without added node features obtained a MSE of 0.294±0.191 while my modified version obtained a score of 0.130±0.031.

Furthermore, compared to the non-3D Graphormer, the Graphormer3D achieved comparable results, sometimes even outperforming the other method as shown with ZINC and QM9 in Table 4. Moreover, the 3D version appears to be around 20% faster during training compared to the non-3D version, as determined on the ZINC dataset. As for the question how much worse 3D conformers generated cheaply using RDKit compared to DFT perform for Graphormer3D, it seems, with the available results, that even with RDKit generated conformers, as we can see in the case of the ZINC dataset, 3D results can be very satisfying, even outporforming the non-3D Graphormer sometimes. However, more dedicated experiments for this topic might be helpful to determine the exact standings in this matter.

These findings, along with other future directions, are discussed more extensively in the following Section 6.2 of the thesis.

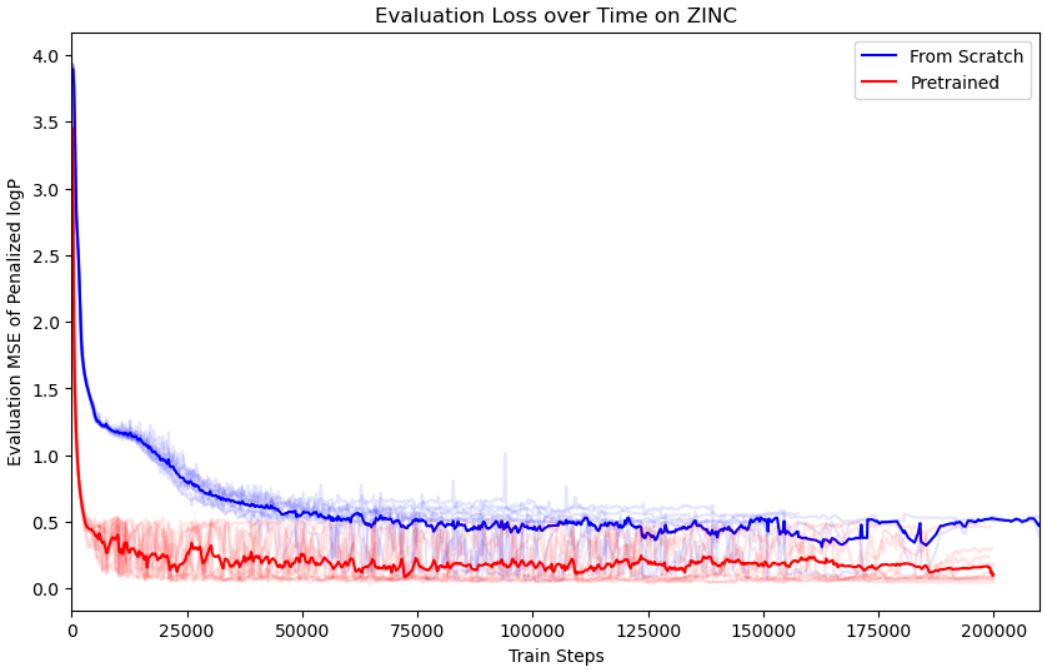

Figure 5: Comparison between the evaluation loss progression between models pretrained on PCQM4Mv2 and those trained from scratch. Different lenghts are due to early stopping. The "From Scatch" versions do not go down further after 200,000 steps.

5.5 Contributions to the Huggingface Transformers Repository

As part of my commitment to open-source collaboration and the improvement of the Graphormer, I have sought to contribute several important modifications to the Huggingface Transformers repository as described in Section 4.3.10.

Being a novice contributor to such a widely used repository, the process has presented me with unique challenges. My first pull request to such an influential repository, which addresses some crucial changes to the original version of the Graphormer, was submitted approximately a month ago. However, as of the writing of this thesis, the pull request is still awaiting review from the repository maintainers. This has been an instructive experience in understanding the dynamics of contributing to a high-profile, open-source project, where many pull requests are likely under consideration at any given time.

6 Conclusion and Future Work

6.1 Conclusion

In this thesis, I have effectively integrated partially novel pretraining methods that can be utilized in the field of GTs and potentially GNNs. Among these, the noise pretraining method stands out as an innovative adaptation in this domain that can potentially enhance the field of research substantially. The mask pretraining method, while based on existing approaches for GNNs or NLP models, has been seamlessly applied to the context of GTs. The efficacy of these methods has been thoroughly substantiated through rigorous experiments conducted during this research.

In addition, this work introduces an improved version of Graphormer3D for graph property prediction tasks. This optimized model offers increased computational efficiency while outperforming the regular Graphormer in certain aspects.

This research notably provides a robust foundation for establishing a new benchmark for GTs, integrating pretrained models to achieve superior results. By attempting to implement these pretraining

methods in the Huggingface framework and contributing to the Huggingface community, this work aims to facilitate wider access and adoption of GTs. This opens up possibilities for researchers and practitioners to pretrain their own GTs, contributing to the democratization of this technology.

However, the research presented here is not without its limitations. Despite the comprehensive exploration of pretraining methods and datasets, the scope of investigation was restricted due to time constraints. Furthermore, there may be room for further optimization in the implementation of some parts of the code. These are typical challenges in this field, and future research could address these issues, extending the exploration to more methods and datasets.

By making the code freely available, this research contributes to the open-source ethos of the scientific community, offering valuable resources for those interested in GTs.

In conclusion, this thesis advances the field of graph machine learning, providing robust methodologies and insightful findings for future studies. Despite certain limitations, the outcomes of this research signify a substantial step forward in our understanding and utility of GTs and machine learning algorithms. Ultimately, this work brings GTs closer to the broader machine learning community, helping to shape the future of this burgeoning field.

6.2 Future Work

6.2.1 Contribution to the Hugginface Transformers Repository

Continuing the efforts to contribute to the Huggingface repository will be an important aspect of future work. Once feedback on my initial pull request is received, this will inform the submission of subsequent contributions, potentially leading to the integration of the improvements and new methods introduced in this thesis into the Huggingface Transformers repository. This would greatly extend the impact of this research, bringing these improvements and novel methods to a wider audience in the research and development community.

6.2.2 Potential Enhancements to the 3D Noise Prediction Method

In my approach, I attempted to predict the noise of the Chi distribution as it is. However, it might be interesting to see if the performance improves if firstly, the 3D noise is drawn such that the norm of it is close to a normal distribution. Secondly, if the noise is normalized. Thirdly, if I just predict the squared sum of the vector components which would correspond to a Chi-Squared distribution. These adjustments to the methodology could potentially pave the way for broader applications in this research field. The ultimate utility and effectiveness of these enhancements can be determined by empirical testing and their performance on downstream tasks.

6.2.3 Pretraining CLS token

Multiple possible future experiments are possible. Firstly, in the pretraining only node-based methods were used, which means that the CLS token that is used for graph-based classification is not trained during pretraining at all. It would be possible to try out graph-based pretraining or possibly also a combination of different pretraining methods at once to make the loss during pretraining more informed. Moreover, it would also be possible to utilize supervised pretraining or during pretraining to combine multiple pretraining datasets.

6.2.4 Exploration of Evaluation Metrics

The choice of evaluation metrics can have a significant influence on the assessment and comparison of different models and methods. Throughout this thesis, the ROC-AUC metric (in its "macro" variant) was primarily used for evaluation of multi-label classification tasks. While this is a standard practice in the field, it is worth noting that different variants of ROC-AUC ("micro", "macro", "samples") have different characteristics and may be more or less suitable depending on the specifics of the problem at hand.

Future research could conduct a comprehensive investigation into the effects of using different ROC-AUC variants for the evaluation of molecular property prediction tasks, and perhaps even propose a novel metric or methodology specifically tailored to these tasks. Additionally, a more detailed investigation of how to handle missing labels in multi-label datasets like Tox21 would be beneficial.

It would be particularly interesting to explore how different loss function implementations align with different ROC-AUC variants, and how these could be leveraged to handle class imbalance in such tasks.

6.2.5 Aggregating Information Across Categories

One of the significant challenges in machine learning, particularly in the domain of molecular property prediction, is handling categories (such as embeddings or hidden states) that are rarely encountered. These "rare" categories may not receive sufficient "training signal" during the learning process, potentially limiting the model's ability to generalize to unseen data.

In the Graphormer model, which is the focus of this thesis, a potential approach to addressing this issue was presented in the form of multi-hop strategy where the attention bias is created with `input_edges`. This involves the aggregation of information across categories via batch matrix multiplication. While the original Graphormer implementation does not explicitly justify or delve into the specifics of this method, it nonetheless represents a novel and intriguing way of managing these "rare" categories.

Future work could explore this method in depth, examining its theoretical properties, integrating it into other architectures (such as other variants of GTs or GNNs), and empirically evaluating its performance on a variety of datasets. Particularly, it would be valuable to compare this method against conventional approaches that utilize distinct embeddings or hidden states for each category. The focus of such a comparison should be not only on model expressiveness, but also on the ability of the model to generalize.

Moreover, it would be worthwhile to consider the potential of this approach beyond the context of embeddings, extending it to other types of layer outputs or hidden states. Such a direction of research could provide valuable insights into the development of more robust and generalizable machine learning models for molecular property prediction, along with other tasks involving categorical data with many rare categories.

6.2.6 Possible Further Enhancement to the Graphormer3D Variant

In the course of this research, I developed the idea of a modified version of Graphormer3D. This alternative model leverages the minimal and maximal bound distances as a measure of distances between nodes, deviating from the typical use of the Euclidean norm. The inspiration for this modification came from the work of [Gasteiger et al., 2021].

The implementation envolves using RDKit to discern the min and max distances, which can subsequently be passed through two distinct Gaussian layers, with the results being summed up, potentially after a linear layer. This innovative approach aims to refine the model's understanding of inter-node relationships, ultimately enhancing its predictive capabilities regarding molecular properties.

The reasoning behind this modification stems from the understanding that not all graphs come equipped with 3D information, particularly when dealing with density functional theory calculations. Furthermore, even when 3D information is available from DFT calculations, it can be subject to variability due to randomness in initial positioning. By using the minimal and maximal bound distances as a more consistent measure, this alternative model seeks to sidestep these potential issues, providing a robust approach to dealing with a wider variety of molecular datasets.

Acknowledgements

I would like to acknowledge Andreas Mayr, who served as my supervisor, for offering feedback on my findings and approach throughout the research process.

Moreover, I extend my sincere appreciation to the Institute of Machine Learning at the Johannes Kepler University in Linz. The computational resources they provided were invaluable in facilitating the execution of the computationally intensive experiments intrinsic to this research. Their support undeniably played a significant role in advancing and finalizing my project.

References

C. Cai, T. S. Hy, R. Yu, and Y. Wang. On the Connection Between MPNN and Graph Transformer. June 2023. URL https://openreview.net/forum?id=1EuHYKFPgA.

J. Chaitanya. Transformers are Graph Neural Networks, Sept. 2020. URL https://thegradient.pub/transformers-are-graph-neural-networks/.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Oct. 2020. URL https://openreview.net/forum?id=YicbFdNTTy.

V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

C. Fourrier. Graph Classification with Transformers, Apr. 2023. URL https://huggingface.co/blog/graphml-classification.

J. Gasteiger, C. Yeshwanth, and S. Günnemann. Directional Message Passing on Molecular Graphs via Synthetic Coordinates. Nov. 2021. URL https://openreview.net/forum?id=ZRu0_3azrCd.

P. Hohenberg and W. Kohn. Inhomogeneous Electron Gas. *Physical Review*, 136(3B):B864–B871, Nov. 1964. doi: 10.1103/PhysRev.136.B864. URL https://link.aps.org/doi/10.1103/PhysRev.136.B864.

W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html.

J. J. Irwin, K. G. Tang, J. Young, C. Dandarchuluun, B. R. Wong, M. Khurelbaatar, Y. S. Moroz, J. Mayfield, and R. A. Sayle. ZINC20—A Free Ultralarge-Scale Chemical Database for Ligand Discovery. *Journal of Chemical Information and Modeling*, 60(12):6065–6073, Dec. 2020. ISSN 1549-9596. doi: 10.1021/acs.jcim.0c00675. URL https://doi.org/10.1021/acs.jcim.0c00675.

J. Kim, D. T. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong. Pure Transformers are Powerful Graph Learners. May 2022. URL https://openreview.net/forum?id=um2BxfgkT2_.

A. Krauck. One hot graph. https://github.com/alexanderkrauck/OneHotGraph, 2023.

D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou. Rethinking Graph Transformers with Spectral Attention. In *Advances in Neural Information Processing Systems*, volume 34, pages 21618–21629. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/b4fd1d2cb085390fbbadae65e07876a7-Abstract.html.

G. Landrum. Rdkit: Open-source cheminformatics, 2023. URL http://www.rdkit.org.

A. Mayr, G. Klambauer, T. Unterthiner, M. Steijaert, J. Wegner, H. Ceulemans, D.-A. Clevert, and S. Hochreiter. Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chemical Science*, 9, June 2018. doi: 10.1039/C8SC00148K.

M. Nakata and T. Shimazaki. PubChemQC Project: A Large-Scale First-Principles Electronic Structure Database for Data-Driven Chemistry, May 2017. URL https://pubs.acs.org/doi/pdf/10.1021/acs.jcim.7b00083.

M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1:140022, 2014. ISSN 2052-4463. doi: 10.1038/sdata.2014.22.

L. Rampasek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. May 2022. URL https://openreview.net/forum?id=lMMaNf6oxKM.

B. Ramsundar, P. Eastman, P. Walters, and V. Pande. *Deep Learning for the Life Sciences: Applying Deep Learning to Genomics, Microscopy, Drug Discovery, and More*. O'Reilly Media, Sebastopol, CA, 1st edition edition, Apr. 2019. ISBN 9781492039839.

A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. I. Goddard, and W. M. Skiff. UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the American Chemical Society*, 114(25):10024–10035, Dec. 1992. ISSN 0002-7863. doi: 10.1021/ja00051a040. URL https://doi.org/10.1021/ja00051a040.

Y. Rong, Y. Bian, T. Xu, W. Xie, Y. WEI, W. Huang, and J. Huang. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In *Advances in Neural Information Processing Systems*, volume 33, pages 12559–12571. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/hash/94aef38441efa3380a3bed3faf1f9d5d-Abstract.html.

L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond. Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. *Journal of Chemical Information and Modeling*, 52(11):2864–2875, Nov. 2012. ISSN 1549-9596. doi: 10.1021/ci300415d. URL https://doi.org/10.1021/ci300415d.

A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8459–8468. PMLR, Nov. 2020. URL https://proceedings.mlr.press/v119/sanchez-gonzalez20a.html.

Y. Shi, S. Zheng, G. Ke, Y. Shen, J. You, J. He, S. Luo, C. Liu, D. He, and T.-Y. Liu. Benchmarking Graphormer on Large-Scale Molecular Modeling Datasets, Jan. 2023. URL http://arxiv.org/abs/2203.04810. arXiv:2203.04810 [cs].

Unknown. Tox21 challenge, 2014. URL https://tripod.nih.gov/tox21/challenge/.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. Gomez, S. Gouws, L. Jones, \. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit. Tensor2Tensor for Neural Machine Translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 193–199, Boston, MA, Mar. 2018. Association for Machine Translation in the Americas. URL https://aclanthology.org/W18-1819.

A. Vaswani, P. Ramachandran, A. Srinivas, N. Parmar, B. Hechtman, and J. Shlens. Scaling local self-attention for parameter efficient visual backbones. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12889–12899, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society. doi: 10.1109/CVPR46437.2021.01270. URL https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01270.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390294. URL https://doi.org/10.1145/1390156.1390294.

Y. Wang, S. Li, T. Wang, Z. Wang, X. He, B. Shao, and T.-Y. Liu. An ensemble of visnet, transformer-m, and pretraining models for molecular property prediction in ogb large-scale challenge @ neurips 2022. *CoRR*, abs/2211.12791, 2022. URL https://doi.org/10.48550/arXiv.2211.12791.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao,

S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. MoleculeNet: A Benchmark for Molecular Machine Learning, Oct. 2018. URL http://arxiv.org/abs/1703.00564. arXiv:1703.00564 [physics, stat].

Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji. Self-Supervised Learning of Graph Neural Networks: A Unified Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2): 2412–2429, Feb. 2023. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3170559.

C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do Transformers Really Perform Badly for Graph Representation? In *Advances in Neural Information Processing Systems*, volume 34, pages 28877–28888. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f1c1592588411002af340cbaedd6fc33-Abstract.html.

J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec. Identity-aware Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10737–10745, May 2021. ISSN 2374-3468. doi: 10.1609/aaai.v35i12.17283. URL https://ojs.aaai.org/index.php/AAAI/article/view/17283.

S. Zaidi, M. Schaarschmidt, J. Martens, H. Kim, Y. W. Teh, A. Sanchez-Gonzalez, P. Battaglia, R. Pascanu, and J. Godwin. Pre-training via Denoising for Molecular Property Prediction. Sept. 2022. URL https://openreview.net/forum?id=tYIMtogyee.

B. Zhang, S. Luo, L. Wang, and D. He. Rethinking the Expressive Power of GNNs via Graph Biconnectivity. Sept. 2022. URL https://openreview.net/forum?id=r9hNv76KoT3.