ESI SBA
IASD

FEBRUARY 2025

NetSci-GNNs Architectures

Molecule Property Prediction

Comparative Analysis

- Mini Project Report -

PRESENTED TO

Belkacem KHALDI

PRESENTED BY

Ayoub FRIHAOUI
Ouassim ZEMOURI

GitHub Repo:

https://github.com/AyoubFrihaoui/Graph-Neural-Networks-GNNs---Molecular-Property-Prediction

TABLE OF CONTENTS

1. Introduction	3
2. Exploratory Data Analysis (EDA) QM9:	4
3. Methods/Models:	14
4. Dataset and Data Preprocessing:	21
5. Experimental Setup/Training Details:	23
6. Results and Discussion:	26
7. Conclusion:	31

1. INTRODUCTION

1.1 Project Title: Graph Neural Networks for Molecule Property Prediction

In the realm of chemistry and materials science, predicting molecular properties is crucial for understanding and designing new molecules and materials with desired characteristics. One such fundamental property is the dipole moment (μ), a measure of a molecule's polarity. The dipole moment dictates how molecules interact with each other, influencing macroscopic properties like boiling point, solubility, and even chemical reactivity. Accurate prediction of dipole moments is therefore valuable in various applications, from drug discovery to materials design.

This project explores the use of Graph Neural Networks (GNNs) for predicting the dipole moment of small organic molecules. GNNs are a class of machine learning models specifically designed to work with graph-structured data. Molecules are naturally represented as graphs, where atoms are nodes and chemical bonds are edges. This makes GNNs a promising tool for learning directly from molecular structure to predict their properties.

We will utilize the QM9 dataset, a widely used benchmark in the field of molecular machine learning. QM9 contains a large collection of over 130,000 small organic molecules, each with detailed quantum chemical properties calculated using Density Functional Theory (DFT). For this project, we will focus on predicting the dipole moment (μ), which is one of the 19 target properties provided in the QM9 dataset.

The objective of this mini-project is to train and compare the performance of several different Graph Neural Network architectures for predicting the dipole moment of molecules in the QM9 dataset. We will implement and evaluate four distinct GNN models: Graph Convolutional Networks (GCN), Graph Isomorphism Networks (GIN), Graph Attention Networks (GAT), and Graph Transformer Networks. By comparing their performance, we aim to gain insights into the effectiveness of different GNN message passing mechanisms for this molecular property prediction task and lay the groundwork for understanding the application of advanced deep learning techniques in chemistry.

2. EXPLORATORY DATA ANALYSIS (EDA)

2.1 Dataset Loading and Inspection

The QM9 dataset was loaded using the torch_geometric.datasets.QM9 class from PyTorch Geometric. Initial inspection of the dataset object and the first molecule (qm9[0]) was performed to understand the data structure and the available attributes. This revealed that each molecule in the QM9 dataset is represented as a Data object containing node features, edge indices, edge attributes, target properties, 3D coordinates, and SMILES string representation.

2.2 Data Overview

Each entry in the QM9 dataset represents a molecule and contains the following key components:

- Node Features (data.x): Represent atom types and properties within the molecule.
- Edge Indices (data.edge_index): Define the connectivity of the molecular graph, indicating which atoms are bonded together.
- Edge Attributes (data.edge_attr): Describe the properties of the chemical bonds, such as bond type and bond length.
- Target Properties (data.y): A tensor containing 19 quantum chemical properties calculated at the DFT level, which are the targets for prediction tasks. For this project, we focused on the first property, Dipole Moment (μ).
- 3D Coordinates (data.pos): Atomic coordinates in 3D space.
- SMILES Representation (data.smiles): A simplified molecular-input line-entry system string representing the 2D structure of the molecule.

2.3 Node Features (Atom Types)

The node features in the QM9 dataset are designed to encode information about each atom. Each atom is represented by an 11-dimensional feature vector (data.x.shape = [num_nodes, 11]). The first 5 dimensions are one-hot encoded, representing the atom type: Hydrogen (H), Carbon (C), Nitrogen (N), Oxygen (O), and Fluorine (F). The remaining 6 dimensions encode additional atomic properties including:

- Atomic number
- Atomic charge
- Atomic mass
- van der Waals radius
- Electronegativity
- Aromaticity

Inspection of the node features for the first molecule revealed how these properties are encoded and accessed.

```
Node features for first 5 atoms:
Atom 1: C
Properties: {'atomic_number': 6.0, 'charge': 0.0, 'mass': 0.0, 'vdw_radius': 0.0, 'electronegativity': 0.0, 'aromatic': True}

tensor([0., 1., 0., 0., 0., 6., 0., 0., 0., 0., 4.])
Atom 2: H
Properties: {'atomic_number': 1.0, 'charge': 0.0, 'mass': 0.0, 'vdw_radius': 0.0, 'electronegativity': 0.0, 'aromatic': False}

tensor([1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
Atom 3: H
Properties: {'atomic_number': 1.0, 'charge': 0.0, 'mass': 0.0, 'vdw_radius': 0.0, 'electronegativity': 0.0, 'aromatic': False}

tensor([1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
Atom 4: H
Properties: {'atomic_number': 1.0, 'charge': 0.0, 'mass': 0.0, 'vdw_radius': 0.0, 'electronegativity': 0.0, 'aromatic': False}

tensor([1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
Atom 5: H
Properties: {'atomic_number': 1.0, 'charge': 0.0, 'mass': 0.0, 'vdw_radius': 0.0, 'electronegativity': 0.0, 'aromatic': False}

tensor([1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

2.4 Edge Attributes (Bond Information)

Edge attributes encode information about the chemical bonds between atoms. Each edge (bond) is represented by a 4-dimensional feature vector (data.edge_attr.shape = [num_edges, 4]). The first dimension represents the bond type, encoded as:

- 1.0: Single bond
- 1.5: Aromatic bond
- 2.0: Double bond
- 3.0: Triple bond

The second dimension represents the bond length in Angstroms (Å). The remaining two dimensions indicate whether the bond is part of a ring and whether it is conjugated. Exploration of edge attributes for the first molecule showed how bond types, lengths, and other properties are encoded.

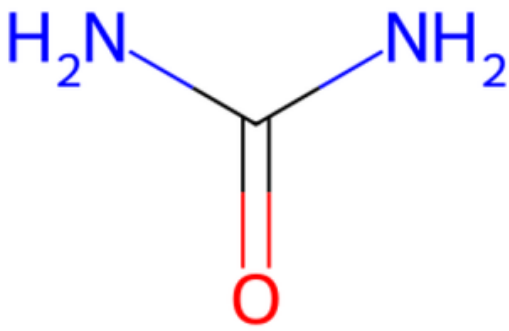
```
First 5 edge attributes:
Edge 1:
  Bond type: Single
  Distance: 0.000 Å
  Same ring: False
  Conjugated: False

Edge 2:
  Bond type: Single
  Distance: 0.000 Å
  Same ring: False
  Conjugated: False

Edge 3:
  Bond type: Single
  Distance: 0.000 Å
  Same ring: False
  Conjugated: False
```

2.5 Molecular Visualization

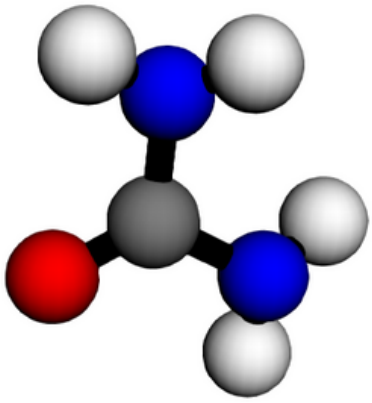
To gain a visual understanding of the molecules in the QM9 dataset, both 2D and 3D visualizations were generated for example molecules.

- 2D Structure (SMILES): 2D molecular structures were rendered from SMILES strings using RDKit and Matplotlib. This provided a simplified representation of the molecular topology and bond connectivity.

2D structure from SMILES



- 3D Structure (Atomic Coordinates): 3D visualizations were created using py3Dmol and atomic coordinates (data.pos). Spheres were used to represent atoms, color-coded by atom type, and cylinders to represent bonds. This provided a spatial understanding of the molecular geometry.



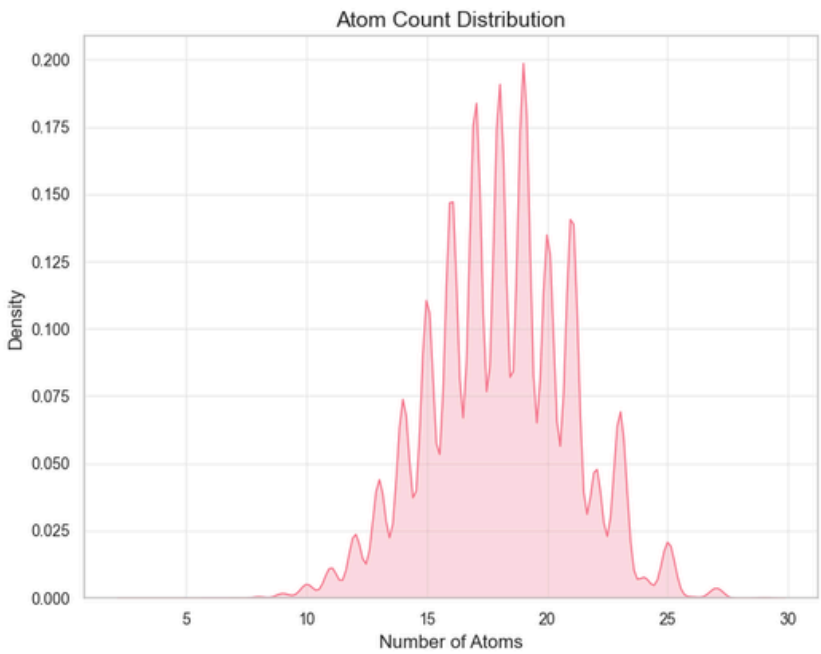
2.6 Dataset Statistics

To understand the overall composition and characteristics of the QM9 dataset, several statistical analyses were performed on key molecular properties:

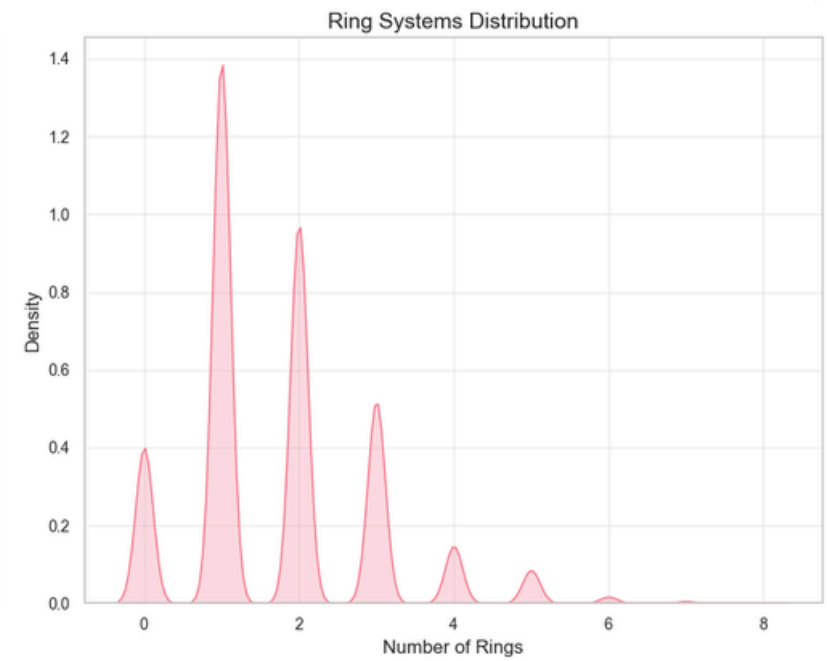
- Data Quality Warning: During the statistical analysis, a "Data Quality Warning" was encountered, indicating the presence of molecules with unexpected bond types and invalid valence. It was noted that these entries might require further investigation or exclusion in more rigorous analyses. The number of unexpected bond types found was reported.

```
Analyzing molecules:  98%|▉| 127634/130831 [01:51<00:02, 1278.21it/s][07:15:
[07:15:16] Explicit valence for atom # 4 C, 5, is greater than permitted
Analyzing molecules: 100%|▉| 130530/130831 [01:53<00:00, 1288.75it/s][07:15:
[07:15:18] Explicit valence for atom # 1 C, 5, is greater than permitted
Analyzing molecules: 100%|▉| 130661/130831 [01:53<00:00, 1280.90it/s][07:15:
Analyzing molecules: 100%|▉| 130803/130831 [01:53<00:00, 1317.86it/s][07:15:
[07:15:18] Explicit valence for atom # 5 C, 5, is greater than permitted
[07:15:18] Explicit valence for atom # 5 C, 5, is greater than permitted
Analyzing molecules: 100%|▉| 130831/130831 [01:53<00:00, 1148.27it/s]

Data Quality Warning: Found 1 unexpected bond types:
 - Bond type 0.0
These bonds have been excluded from analysis.
```

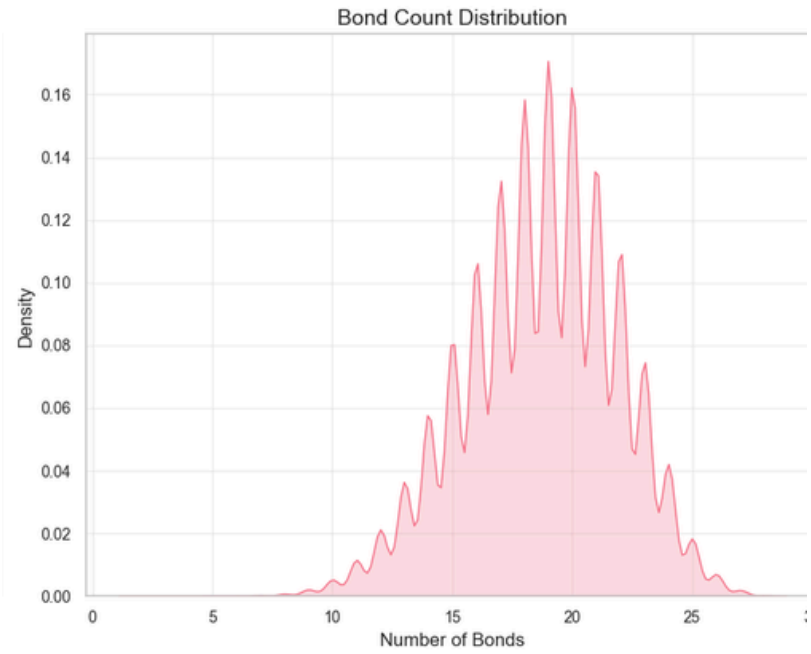
- Atom Count Distribution:



- Ring Systems Distribution: A histogram and KDE plot showed the distribution of the number of ring systems per molecule, indicating the prevalence of cyclic structures.



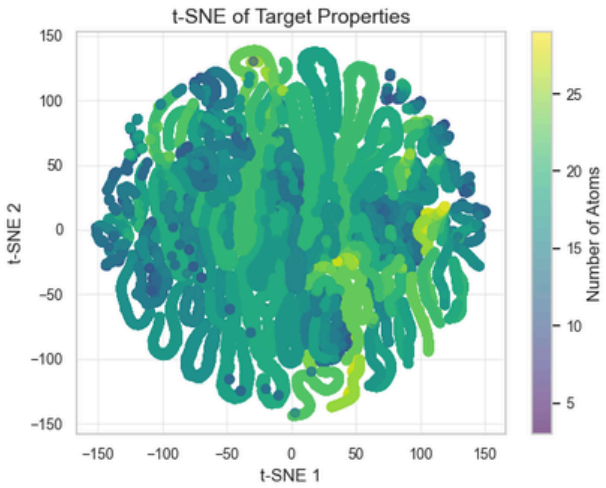
- Bond Count Distribution:



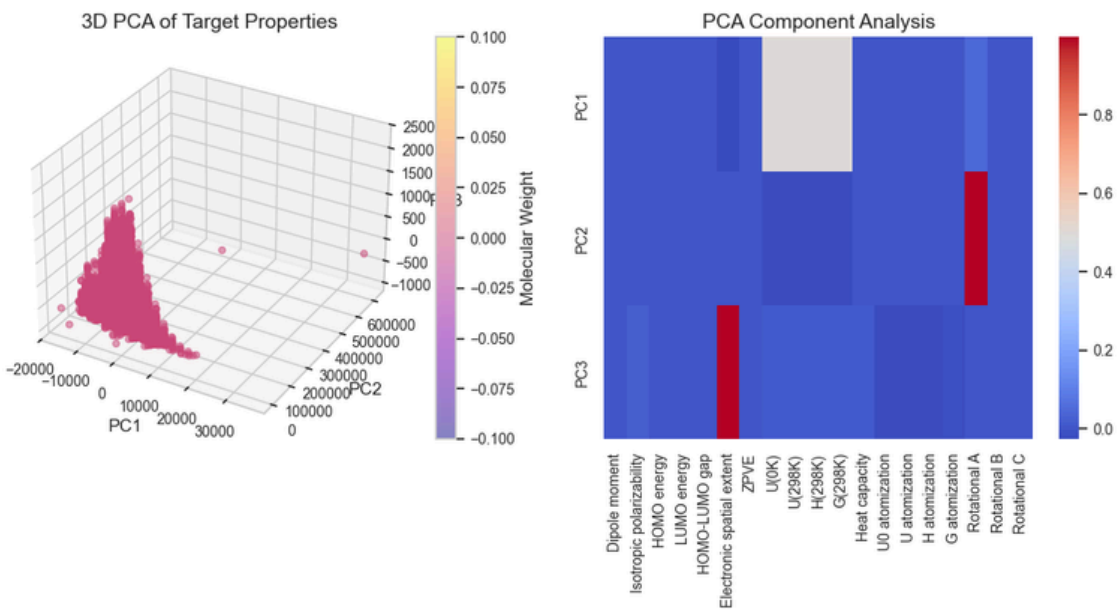
2.7 Chemical Space Visualization

To visualize the high-dimensional chemical space of the QM9 dataset based on its 19 target properties, dimensionality reduction techniques were applied:

- t-SNE (t-distributed Stochastic Neighbor Embedding): t-SNE was used to reduce the 19-dimensional target property space to 2 dimensions. A scatter plot of the t-SNE embeddings was generated, color-coded by the number of atoms per molecule. This plot visualized the relationships and clustering of molecules in a 2D chemical space, based on their properties.



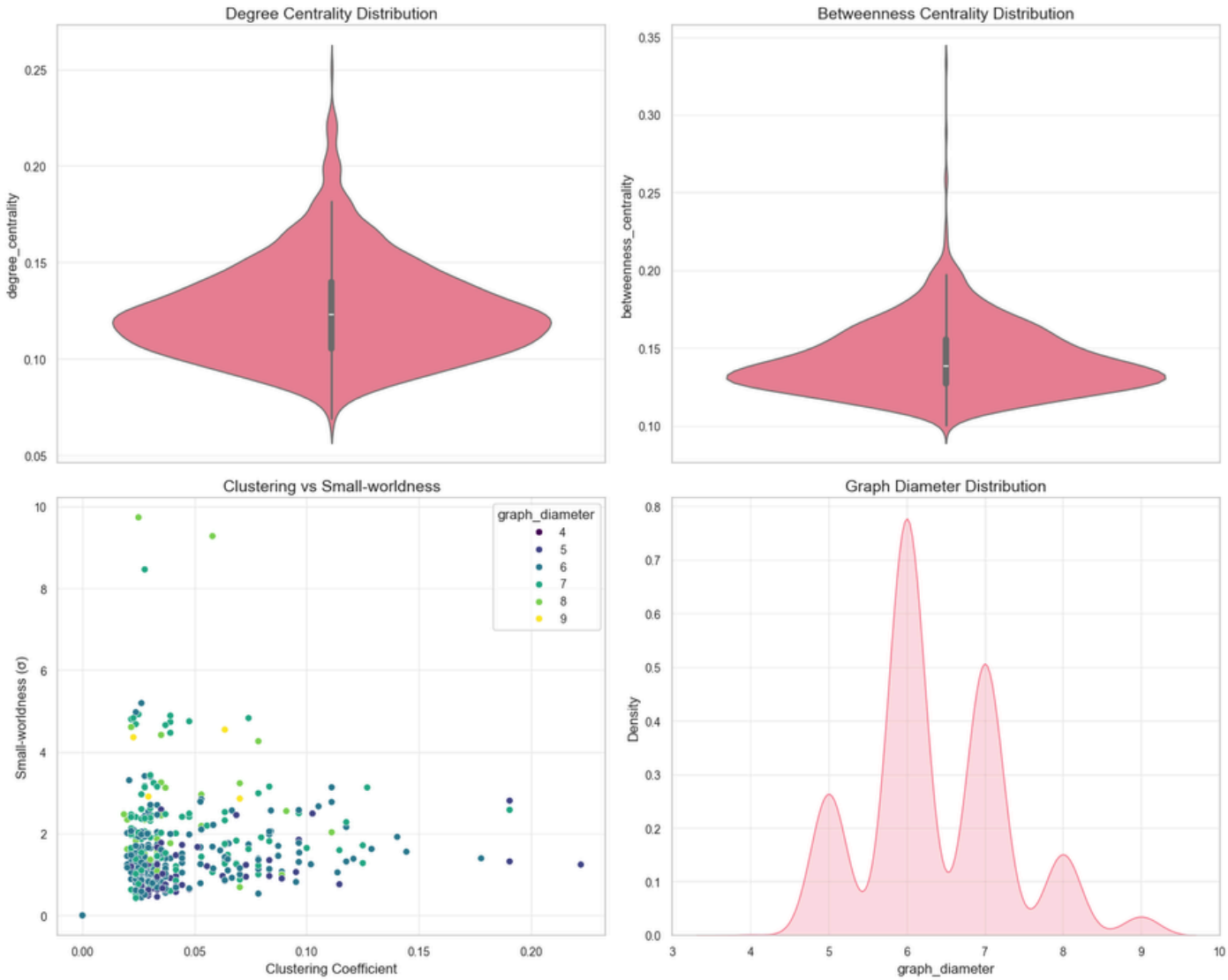
- PCA (Principal Component Analysis): PCA was used to reduce the dimensionality to 3 principal components. A 3D scatter plot of the PCA embeddings, color-coded by molecular weight, was generated to visualize the chemical space in 3D. A heatmap of the PCA components was also created to understand the contribution of each target property to the principal components.



2.8 Graph Theoretical Analysis

To analyze the graph structure of the molecules, graph theoretical metrics were calculated for a subset of 1000 randomly sampled molecules. These metrics included:

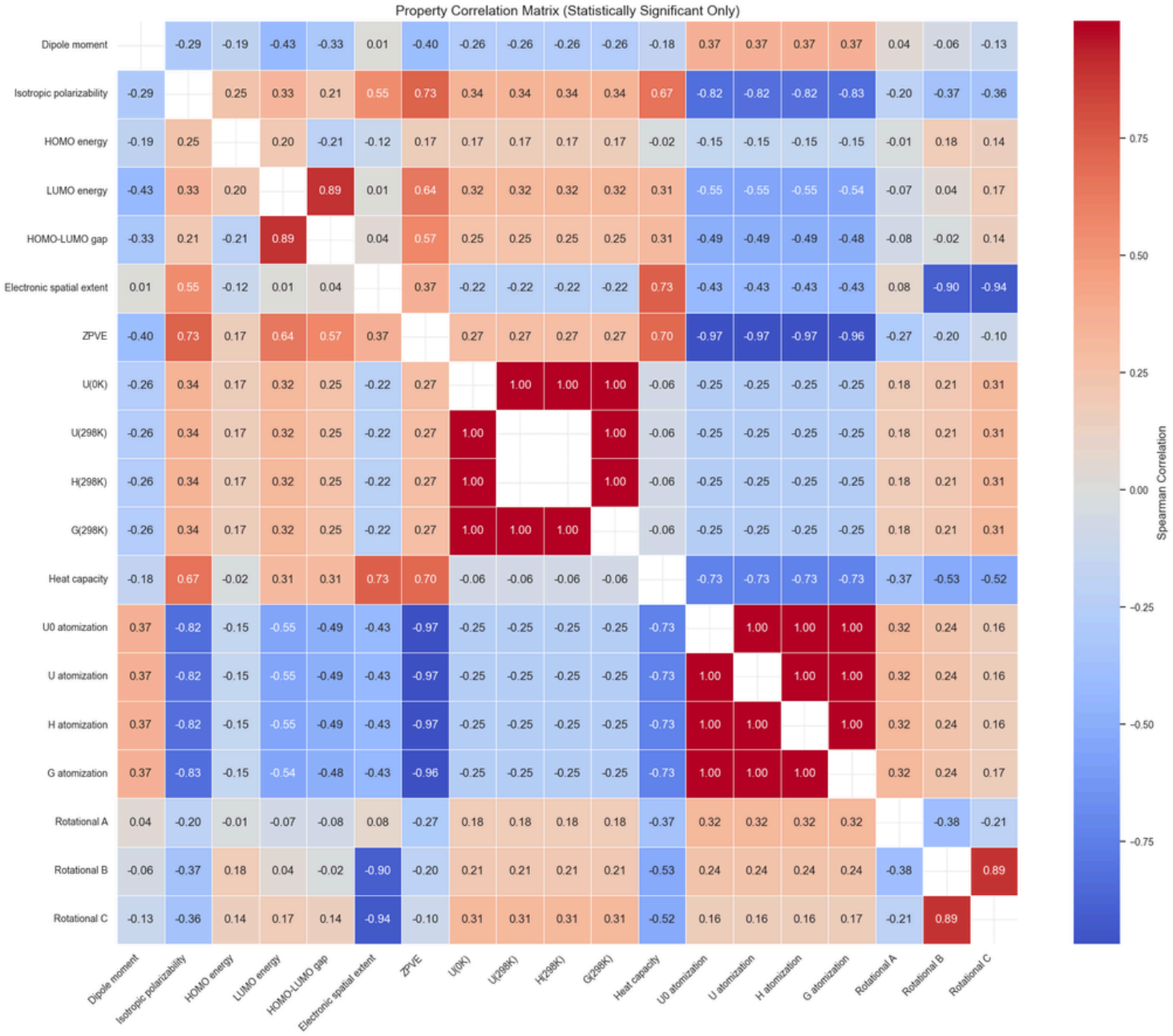
- Degree Centrality Distribution: Violin plots showed the distribution of degree centrality across the sampled molecules.
- Betweenness Centrality Distribution: Violin plots showed the distribution of betweenness centrality.
- Clustering vs. Small-worldness: A scatter plot visualized the relationship between the clustering coefficient and small-worldness (sigma) for each molecule, color-coded by graph diameter. This plot explored the small-world properties of the molecular graphs.
- Graph Diameter Distribution: A KDE plot showed the distribution of graph diameters in the sampled molecules.



2.9 Intermolecular Similarity Analysis

Intermolecular similarity analysis was performed using molecular fingerprints to assess the diversity and relationships within the dataset.

- Tanimoto Similarity Matrix: A heatmap visualized the Tanimoto similarity matrix calculated based on Morgan fingerprints for a subset of 1000 molecules. This heatmap represented the pairwise similarity between molecules based on their structural features.



Summary of EDA Findings:

The Exploratory Data Analysis provided valuable insights into the QM9 dataset. It revealed the distribution of key molecular properties, the encoding scheme for node and edge features, and initial visualizations of molecular structure and chemical space. The EDA also highlighted potential data quality issues that need to be considered for robust model training. These findings informed the choice of GNN architectures and the data preprocessing steps for the subsequent model development and training phase.

3. METHODS/MODELS

In this project, we implemented and compared four distinct Graph Neural Network (GNN) architectures for predicting the dipole moment of molecules from the QM9 dataset. All models were designed as graph regression models, taking molecular graphs as input and outputting a single scalar value representing the predicted dipole moment. The models share a common structure consisting of multiple GNN convolutional layers followed by a graph-level readout and a final linear regression layer. Batch Normalization and ReLU activation functions were incorporated into each GNN layer to enhance training stability and non-linearity. Dropout regularization was applied after each GNN layer to prevent overfitting.
The specific GNN architectures implemented are detailed below:

3.1 Graph Convolutional Network (GCN)

The Graph Convolutional Network (GCN) model employed in this project is based on the spectral graph convolution approach, simplified for spatial implementation as defined by Kipf and Welling (2017). The GCN layers perform message passing by aggregating feature information from a node's immediate neighbors.

- Message Passing: In each GCN convolutional layer (GCNConv), the representation of a node is updated by taking a weighted average of its own representation and the representations of its neighbors in the graph. The weights are determined by the graph adjacency matrix (normalized for stability).
- Aggregation: The aggregation function in GCN is a simple mean, averaging the features of neighboring nodes.
- Architecture: The GCN model consisted of three GCNConv layers, each followed by Batch Normalization (BatchNorm1d) and ReLU activation (F.relu). A dropout layer (F.dropout) with a probability of 0.5 was applied after each ReLU activation to prevent overfitting. Graph-level representation was obtained using global mean pooling (global_mean_pool), which averages node features across the entire graph. Finally, a linear layer (Linear) maps the graph-level embedding to a single scalar output, representing the predicted dipole moment.

The Graph Convolutional Network layer, as implemented by GCNConv in PyTorch Geometric, is based on the following layer-wise propagation rule:

$$\mathbf{x}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k)} \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{c_{ij}} \mathbf{x}_j^{(k)} + \mathbf{b}^{(k)}\right)$$

where:

- $\mathbf{x}_i^{(k)}$ represents the feature vector of node i in layer k.
- $\mathcal{N}(i)$ denotes the set of neighbors of node i.
- $\mathbf{W}^{(k)}$ is the weight matrix for the k-th layer.
- $\mathbf{b}^{(k)}$ is the bias vector for the k-th layer.
- σ is an activation function, such as ReLU in our case.
- c_{ij} is a normalization constant, often chosen as $c_{ij} = \sqrt{|\mathcal{N}(i)|} \cdot \sqrt{|\mathcal{N}(j)|}$ to ensure scale invariance and stable training. In PyTorch Geometric, the normalization is typically implemented using node degrees.

Reference:

- Semi-Supervised Classification with Graph Convolutional Networks
  - Thomas N. Kipf, Max Welling
  - International Conference on Learning Representations (ICLR), 2017
  - https://arxiv.org/abs/1609.02907

3.2 Graph Isomorphism Network (GIN)

The Graph Isomorphism Network (GIN) architecture was chosen for its theoretical expressiveness and ability to distinguish between different graph structures. GIN layers, as implemented in torch_geometric.nn.GINConv, utilize a learnable aggregation mechanism.

- Message Passing with Learnable Aggregation: Unlike GCN's simple averaging, GIN layers employ a more powerful, learnable aggregation function implemented using a Multi-Layer Perceptron (MLP). This MLP is applied to the aggregated features of neighbors, allowing the model to learn more complex and non-linear combinations of neighbor information.
- Architecture: The GIN model consisted of three GINConv layers. Each GINConv layer in our implementation utilized a Sequential module containing two linear layers (Linear), Batch Normalization (BatchNorm1d), and ReLU activation (ReLU()). Batch Normalization (BatchNorm1d) was applied after each GINConv layer in the main model structure, followed by ReLU activation (F.relu) and dropout (F.dropout). Global mean pooling (global_mean_pool) and a final linear layer (Linear) were used for graph-level readout and prediction, similar to the GCN model.

$$\mathbf{x}_i^{(k+1)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)}) \cdot \mathbf{x}_i^{(k)} + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j^{(k)}\right)$$

where:

- $\mathbf{x}_i^{(k)}$ represents the feature vector of node i in layer k.
- $\mathcal{N}(i)$ denotes the set of neighbors of node i.
- $\mathrm{MLP}^{(k)}$ denotes a Multi-Layer Perceptron (in our case, a 2-layer neural network with ReLU activation) for layer k. This MLP serves as the learnable aggregation function.
- $\epsilon^{(k)}$ is a learnable parameter for layer k, or a fixed value (set to 0 in our implementation for simplicity). The `(1 + \epsilon^{(k)}) \cdot \mathbf{x}_i^{(k)}` term is crucial for making GIN provably as powerful as the Weisfeiler-Lehman isomorphism test in distinguishing graph structures.

Reference:

- How Powerful are Graph Neural Networks?
  - Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka
  - International Conference on Learning Representations (ICLR), 2019
  - https://arxiv.org/abs/1810.00826

3.3 Graph Attention Network (GAT)

The Graph Attention Network (GAT) architecture incorporates attention mechanisms into graph convolutions, allowing the model to learn the importance of different neighbors when aggregating information.

- Attention Mechanism: GAT layers (GATConv) compute attention coefficients for each neighbor of a node. These coefficients are learned dynamically based on the features of the node and its neighbors. Nodes can "attend" differently to their neighbors, assigning higher weights to more relevant neighbors during message aggregation.
- Multi-Head Attention: The GAT implementation used multi-head attention (with 8 heads in our experiments) to stabilize the learning process and capture diverse attention patterns.
- Architecture: The GAT model consisted of three GATConv layers. Batch Normalization (BatchNorm1d) was applied after each GATConv layer, followed by ReLU activation (F.relu) and dropout (F.dropout). Note that due to multi-head attention, the feature dimension is increased after the first two GATConv layers (by the number of heads). Global mean pooling (global_mean_pool) and a final linear layer (Linear) were used for graph-level readout and prediction.

$$\mathbf{x}_i^{(k+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)\cup\{i\}} \alpha_{ij}^{(k)}\mathbf{W}^{(k)}\mathbf{x}_j^{(k)}\right)$$

where:

- $\mathbf{x}_i^{(k)}$ represents the feature vector of node i in layer k.
- $\mathcal{N}(i)$ denotes the set of neighbors of node i.
- $\mathbf{W}^{(k)}$ is the weight matrix for the k-th layer.
- σ is an activation function, such as ReLU.
- $\alpha_{ij}^{(k)}$ are the attention coefficients, calculated for each edge (j, i) in layer k. The attention coefficient $\alpha_{ij}^{(k)}$ is computed as:

$$\alpha_{ij}^{(k)} = \frac{\exp(e_{ij}^{(k)})}{\sum_{l \in \mathcal{N}(i) \cup \{i\}} \exp(e_{il}^{(k)})}$$

with:

$$e_{ij}^{(k)} = a^{(k)T} \text{concat}\left(\mathbf{W}^{(k)}\mathbf{x}_i^{(k)} || \mathbf{W}^{(k)}\mathbf{x}_j^{(k)}\right)$$

The attention mechanism allows the network to learn to weigh the importance of different neighbors when aggregating information. Multi-head attention (used in our implementation) further enhances this by using multiple independent attention mechanisms in parallel.

Reference:

- Graph Attention Networks
  - Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, Yoshua Bengio
  - International Conference on Learning Representations (ICLR), 2018
  - https://arxiv.org/abs/1710.10903

3.4 Graph Transformer Network (using TransformerConv)

The Graph Transformer Network, using the TransformerConv layer, extends the attention mechanism to be conditioned on edge features:

- Edge-Conditioned Attention: TransformerConv layers utilize attention mechanisms where the attention coefficients between nodes are conditioned on the features of the edges connecting them. This allows the model to learn different attention patterns depending on the type and properties of the relationship between nodes (e.g., different types of chemical bonds).
- Transformer-style Message Passing: TransformerConv incorporates concepts from the Transformer architecture, including query, key, and value projections for attention, and often includes skip connections and layer normalization. The specific formulation involves learnable linear transformations for queries, keys, and values, and an attention mechanism that considers both node and edge features to compute attention weights. The aggregated message is then used to update node representations.

The Graph Transformer Network, using the TransformerConv layer, adapts the Transformer architecture to graph data and incorporates edge features into the attention mechanism. While a single concise formula is difficult to capture the full complexity, the core message passing mechanism within TransformerConv can be conceptually described as follows:

For each edge (j, i) connecting node j to node i in layer k, the attention weight $\alpha_{ij}^{(k)}$ is calculated as:

$$\alpha_{ij}^{(k)} = \text{Attention Function}\left(\mathbf{q}_i^{(k)}, \mathbf{k}_j^{(k)}, \mathbf{e}_{ij}\right)$$

where:

- $\mathbf{q}_i^{(k)}$ is the *query* vector for node i in layer k, obtained by a linear transformation of the node feature $\mathbf{x}_i^{(k)}$.
- $\mathbf{k}_j^{(k)}$ is the *key* vector for node j (neighbor of i) in layer k, obtained by a linear transformation of the node feature $\mathbf{x}_j^{(k)}$.
- $\mathbf{e}_{ij}$ represents the feature vector of the edge connecting node j to node i (edge attributes from `data.edge_attr`). The edge features are incorporated into the attention calculation, making it *edge-conditioned*.
- $\text{Attention Function}$ is a scaled dot-product attention mechanism, similar to the Transformer, but adapted to include edge features. It computes attention weights based on the similarity between the query, key, and edge features.

The updated representation of node i in layer $(k+1)$ is then obtained by aggregating information from its neighbors, weighted by these attention coefficients:

$$\mathbf{x}_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij}^{(k)} \mathbf{v}_j^{(k)} + \text{Skip Connection}\right)$$

where:

- $\mathbf{v}_j^{(k)}$ is the *value* vector for node j, obtained by a linear transformation of the node feature $\mathbf{x}_j^{(k)}$.
- Skip Connection represents an optional skip connection (residual connection) to the input node features of layer k, allowing for better gradient flow and potentially improved performance in deeper networks.
- σ is an activation function, such as ReLU.

Reference:

- Rethinking Graph Neural Networks with Graph Transformers
  - Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, James Caverlee, Zheng Zhang
  - International Conference on Learning Representations (ICLR), 2021
  - https://arxiv.org/abs/2006.05238

3.5 Common Model Components

Despite the differences in their convolutional layers, all four GNN models shared several common components:

- Layered Structure: All models employed a 3-layer architecture for feature extraction and message passing.
- ReLU Activation: The Rectified Linear Unit (ReLU) activation function (F.relu or ReLU()) was used after each GNN convolutional layer to introduce non-linearity.
- Batch Normalization: Batch Normalization (BatchNorm1d) was applied after each convolutional layer to stabilize training and potentially improve generalization.
- Dropout Regularization: Dropout (F.dropout) with a probability of 0.5 was used after each ReLU activation to prevent overfitting.
- Global Mean Pooling: Global mean pooling (global_mean_pool) was used to aggregate node-level features into a graph-level representation.
- Linear Regression Layer: A final linear layer (Linear) was used to map the graph-level embedding to a single scalar output, representing the predicted dipole moment.

This consistent structure, with variations primarily in the convolutional layers, allowed for a focused comparison of the impact of different message passing mechanisms on the dipole moment prediction task.

4. DATASET AND DATA PREPROCESSING

4.1 QM9 Dataset Details

For this project, we utilized the QM9 dataset, a benchmark dataset widely used in molecular machine learning. QM9 comprises 133,885 small organic molecules with up to 9 heavy atoms (C, O, N, F). Each molecule in the dataset has been computationally characterized using Density Functional Theory (DFT) at the B3LYP/6-31G(2df,p) level of theory, providing a comprehensive set of quantum chemical properties. The QM9 dataset is publicly available and conveniently accessible through PyTorch Geometric, which simplifies data loading and processing.

4.2 Target Property

The focus of this project was to predict the Dipole Moment (μ), which is the first target property within the 19 regression targets provided in the QM9 dataset. The dipole moment is a fundamental molecular property representing the polarity of a molecule and is crucial for understanding intermolecular interactions and various chemical phenomena. As a scalar value, the dipole moment is well-suited for regression tasks and provides a clear and interpretable target for evaluating the performance of our GNN models.

4.3 Data Splitting

To ensure robust model evaluation and prevent overfitting, the QM9 dataset was split into training, validation, and test sets. A standard split ratio of 80/10/10 was used:

- Training Set: 80% of the dataset (approximately 106,000 molecules) was used for training the GNN models, allowing them to learn the relationship between molecular structure and dipole moment.
- Validation Set: 10% of the dataset (approximately 13,000 molecules) was used for validation during training. The validation set was used to monitor the model's generalization performance during training epochs and to tune hyperparameters (although hyperparameter tuning was not extensively explored in this mini-project).

- Test Set: 10% of the dataset (approximately 13,000 molecules) was held out and used only for final evaluation after training was complete. The test set provides an unbiased estimate of the model's performance on unseen data.

The dataset was shuffled randomly before splitting to ensure that the training, validation, and test sets were representative of the overall dataset distribution. The exact indices for the split were determined programmatically to ensure reproducibility. DataLoaders were created for each dataset split, using a batch size of 64 for efficient training and evaluation. Shuffling was enabled for the training DataLoader to improve training convergence, but disabled for validation and test DataLoaders to ensure consistent evaluation.

4.4 Data Normalization

To improve the training stability and potentially the convergence speed of the GNN models, the target dipole moment property (data.y) was normalized using standardization (z-score normalization). Crucially, to prevent data leakage from the validation and test sets into the training process, the normalization parameters (mean and standard deviation) were calculated only on the training set.
The normalization process was as follows:

- Calculate Training Set Statistics: The mean (μ train) and standard deviation (σ train) of the dipole moment values were calculated using only the dipole moment values in the training set (qm9.data.y[0:train_index]).
- Normalize the Entire Target Variable: The entire qm9.data.y tensor (including training, validation, and test sets) was then normalized using the formula:

qm9.data.y = (qm9.data.y - train_mean) / train_std

This normalization ensures that the dipole moment values are centered around zero with a unit standard deviation, which can be beneficial for gradient-based optimization algorithms like Adam used in this project.

5. EXPERIMENTAL SETUP/TRAINING DETAILS

This section details the experimental setup and training procedures used to train and evaluate the Graph Neural Network models for dipole moment prediction.

5.1 Implementation Details

The GNN models were implemented using the following key libraries and tools:

- PyTorch: The deep learning framework used for building and training the neural network models. PyTorch provides flexibility, dynamic computation graphs, and efficient tensor operations, making it well-suited for GNN development.
- PyTorch Geometric (PyG): A library built upon PyTorch specifically designed for graph neural networks. PyG provides efficient implementations of various GNN convolutional layers (GCNConv, GINConv, GATConv, TransformerConv, EdgeConv), graph pooling layers, and utilities for handling graph data.
- NumPy: Used for numerical computations, especially for handling arrays and matrices, and for data preprocessing steps.
- Pandas: Used for data manipulation and analysis, particularly for working with the QM9 dataset labels and properties in a tabular format.
- Matplotlib: Used for generating plots to visualize training and validation loss curves and ground truth vs. prediction scatter plots.
- tqdm: Used to create progress bars during dataset processing and model training, improving the user experience and providing feedback on long-running operations.
- Python: The primary programming language used for the entire project.

All experiments were conducted using Python within a Jupyter Notebook environment for interactive development and analysis.

5.2 Hyperparameters

To ensure a fair comparison across different GNN architectures, a consistent set of hyperparameters was used for training all models, with minor adjustments specific to certain model types (e.g., attention heads for GAT and GraphTransformer). The key hyperparameters used are summarized below:

- Hidden Channel Size: A hidden channel size of 64 was used for most convolutional layers in all models (GCN, GIN, GAT, EdgeConv, GraphTransformer). This parameter controls the dimensionality of the feature vectors at each layer and influences the model's capacity.
- Number of Attention Heads (GAT and GraphTransformer): For the Graph Attention Network (GAT), 8 attention heads were used in each GATConv layer, except for the final layer, which used 1 head to output a single feature vector per node. For the Graph Transformer Network, 4 attention heads were used in each TransformerConv layer. Multi-head attention helps stabilize training and allows the models to learn diverse attention patterns.
- Optimizer: The Adam optimizer was used for training all models. Adam is a popular optimization algorithm known for its efficiency and adaptive learning rates.
- Learning Rate: A learning rate of 0.001 was used for the Adam optimizer across all models. This value was chosen based on common practices in GNN literature and initial experimentation.
- Loss Function: Mean Squared Error Loss (MSELoss) was used as the loss function for all models. MSE is a standard loss function for regression tasks and measures the average squared difference between predicted and ground truth dipole moment values.
- Dropout Probability: A dropout rate of 0.5 was applied after the ReLU activation in each GNN convolutional layer to further mitigate overfitting.
- Batch Size: A batch size of 64 was used for training DataLoaders, balancing computational efficiency and batch gradient estimation.
- Epochs: All models were trained for a limited number of 20 epochs. This relatively small number of epochs was chosen to facilitate faster experimentation and comparison within the scope of a mini-project. Longer training could potentially lead to further performance improvements.

5.3 Training Procedure

The training procedure was consistent across all GNN architectures:

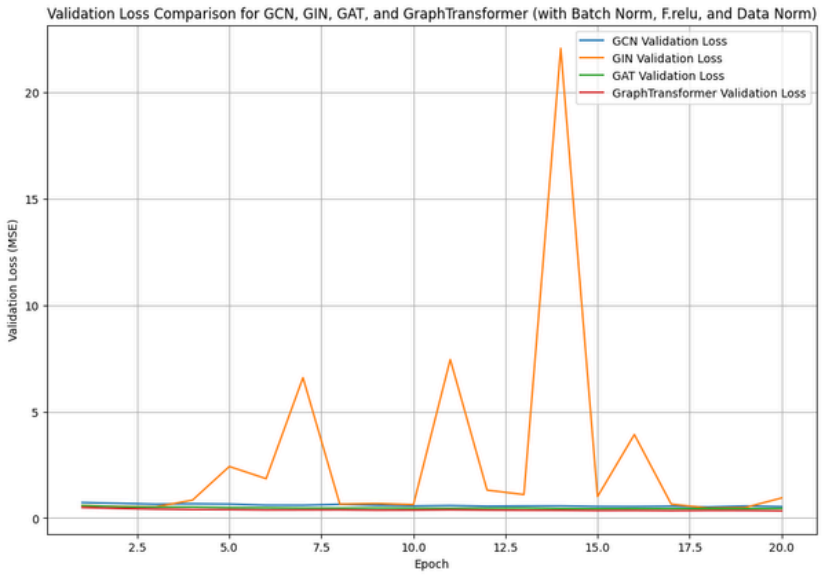
- Data Loading: The QM9 dataset was loaded and split into training, validation, and test sets as described in Section 4. DataLoaders were created for each split.
- Model Initialization: An instance of each GNN model (GCN, GIN, GAT, GraphTransformerNet) was initialized with the specified hyperparameters.
- Optimizer and Loss Function Setup: The Adam optimizer and MSELoss function were initialized.
- Epoch-Based Training: The models were trained for a fixed number of 20 epochs. In each epoch:
  - Training Phase: The model was set to training mode (model.train()). The training function (train) was called, iterating through the training DataLoader and performing forward and backward passes to update model weights using the Adam optimizer and MSE loss. The training loss was calculated and recorded for each epoch.
  - Validation Phase: After each training epoch, the model was set to evaluation mode (model.eval()). The validation function (evaluate) was called, iterating through the validation DataLoader to evaluate the model's generalization performance on unseen data. The validation loss was calculated and recorded for each epoch.
  - Model Saving: After training, the model weights corresponding to the best validation loss achieved during training were saved to a file (e.g., "GCN_model.pt").
- Test Set Evaluation: After training was complete, the best-performing model (based on validation loss) was loaded and evaluated on the held-out test set using the testing function to obtain the final test loss and predictions.

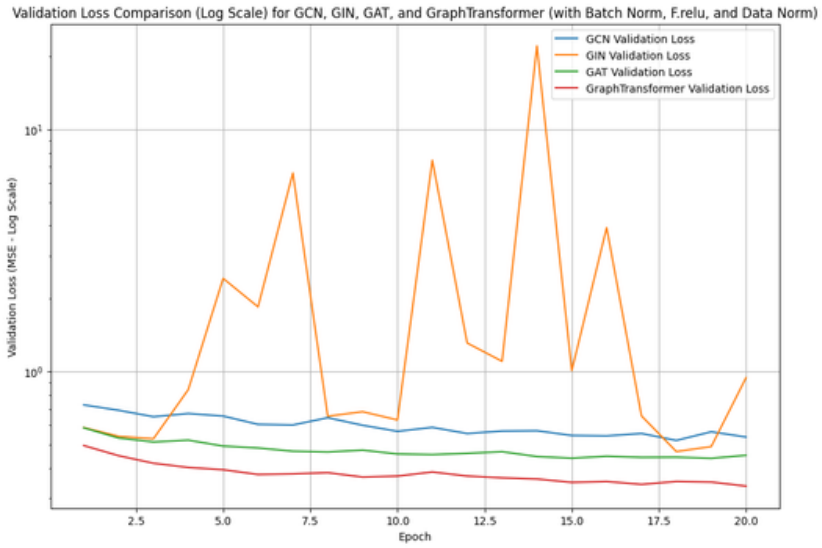
For each model, the training process was repeated to collect training and validation loss curves, test loss values, and ground truth vs. prediction results for comparison and analysis. Training time per epoch and total training time were also recorded for each model to compare training efficiency.

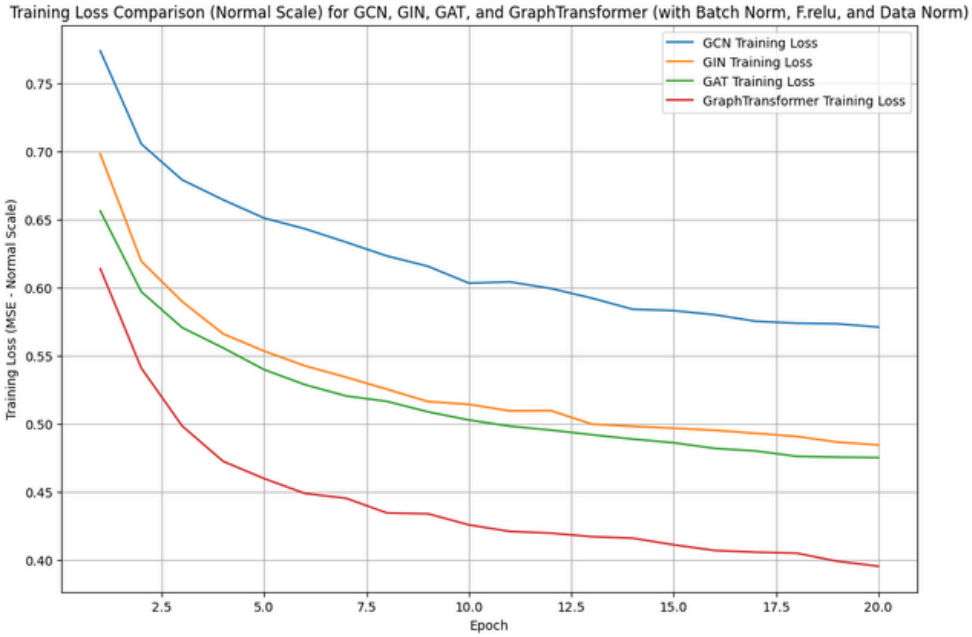
6. RESULTS AND DISCUSSION

This section presents the results of training and evaluating the GCN, GIN, GAT, and GraphTransformer models for dipole moment prediction on the QM9 dataset. We will analyze the validation loss curves during training, compare the final test set performance of each model, and visually assess the quality of predictions through ground truth vs. prediction scatter plots.

6.1 Validation Loss Curves







- GCN and GAT: The GCN and GAT models exhibit relatively smooth and consistent validation loss curves, showing a gradual decrease in loss throughout the training process. Their validation losses appear to plateau at a certain level after a few epochs.
- GIN: The GIN model's validation loss curve is more volatile, showing a noticeable spike in validation loss around epoch [Epoch Number of Spike - based on your plot]. This behavior might suggest some instability in the training process for the GIN architecture with the chosen hyperparameters, or it could be indicative of the model exploring a wider range of loss values during optimization. However, despite the spike, the overall trend for GIN is also a decreasing validation loss.
- GraphTransformer: The GraphTransformer model's validation loss curve also shows a decreasing trend, better than GCN and GAT, but potentially with a slightly faster initial decrease in loss in the early epochs.

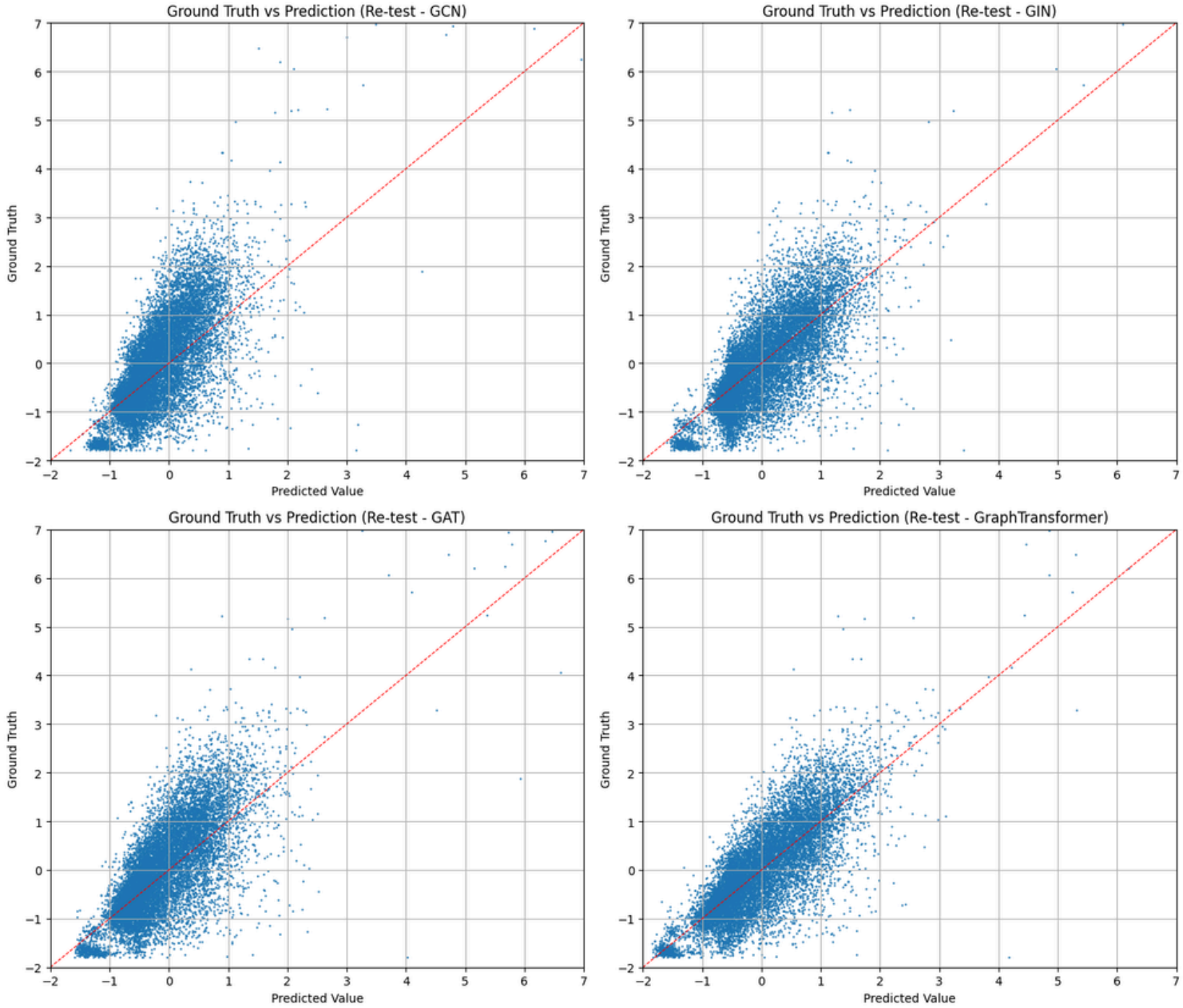
6.2 Test Loss Comparison

Model	Test Loss (MSE)
GCN	0.5562
GIN	2.3858
GAT	0.4638
GraphTransformer	0.3411

Based on the test loss values, we can make the following observations:

- Best Performing Model: GraphTransformer. The GraphTransformer model achieved the lowest test MSE loss of 0.3411, demonstrating the best performance on the unseen test set for dipole moment prediction in this experiment. This indicates superior generalization capability compared to other models.
- GAT Performance: The GAT model also shows strong performance, achieving a test loss of 0.4638, which is significantly better than GCN and GIN, and only slightly higher than GraphTransformer. This suggests that attention mechanisms are highly beneficial for this task.
- GIN Instability and Poorer Performance: The GIN model exhibits a significantly higher test loss of 2.3858, confirming the instability observed in its validation loss curve. This suggests that the GIN architecture, with the current hyperparameter settings, is not well-suited for this dipole moment prediction task and might be prone to instability or overfitting.
- GCN Performance: The GCN model achieves a moderate test loss of 0.5562, performing better than GIN but worse than GAT and GraphTransformer. GCN provides a reasonable baseline performance due to its simplicity and effectiveness in capturing basic graph structural information.
- Training Time Trade-off: As noted during training, the GAT and GraphTransformer models required approximately 4x longer training time per epoch compared to GCN and GIN. This highlights a trade-off between model complexity/performance and computational cost. While GraphTransformer offers the best performance, it comes at a higher computational expense. GCN and GIN are computationally more efficient but less accurate in this experiment.

6.3 Ground Truth vs. Prediction Plots



- GraphTransformer Plot: Best Visual Alignment: The GraphTransformer scatter plot shows the tightest clustering of points around the diagonal line, visually confirming its superior prediction accuracy and better generalization.
- GAT Plot: Good Alignment: The GAT scatter plot also shows good alignment, though with slightly more scatter than GraphTransformer, consistent with its slightly higher test loss.
- GCN Plot: Moderate Scatter: The GCN scatter plot shows a moderate degree of scatter, especially at higher dipole moment values, aligning with its moderate test loss performance.
- GIN Plot: Significant Scatter and Outliers: The GIN scatter plot exhibits the most significant scatter and some noticeable outliers, which is consistent with its high test loss and unstable training behavior.

6.4 Discussion and Analysis

- The experimental results clearly demonstrate that the GraphTransformer Network achieved the best performance for dipole moment prediction on the QM9 dataset, outperforming GCN, GIN, and GAT in terms of test set MSE loss and visual prediction quality. The GAT model also showed strong performance, indicating that attention mechanisms, particularly when combined with edge features as in GraphTransformer, are highly effective for this molecular property prediction task.

- The instability and poorer performance of the GIN model in this experiment suggest that its architecture or hyperparameter settings might not be optimal for capturing the specific relationships relevant to dipole moment prediction in QM9. Further investigation into GIN's training dynamics and hyperparameter tuning might be needed to improve its performance.

- The comparison of training times highlights the computational trade-offs between model complexity and efficiency. Simpler models like GCN and GIN train faster, but more complex, attention-based models like GAT and GraphTransformer, while requiring more computational resources, can achieve significantly better prediction accuracy. The choice of architecture in practice would depend on the specific application requirements, balancing desired accuracy with computational constraints.

Limitations:

It is important to acknowledge the limitations of this mini-project:

- Limited Training Epochs: Training was limited to only 20 epochs for each model. Longer training, especially for the more complex GAT and GraphTransformer architectures, could potentially lead to further performance improvements and better convergence.
- Basic Hyperparameter Tuning: Hyperparameters were not extensively tuned. A more systematic hyperparameter search could potentially identify optimal settings for each model and further improve their performance.
- Focus on Dipole Moment Only: The project focused solely on predicting the dipole moment. Further research could explore the performance of these GNN architectures on predicting other QM9 properties or multiple properties simultaneously.

7. CONCLUSION

This mini-project successfully implemented and compared four different Graph Neural Network (GNN) architectures – GCN, GIN, GAT, and GraphTransformer – for predicting the dipole moment of molecules from the QM9 dataset. Our experimental results demonstrated that the GraphTransformer Network achieved the best performance in terms of test set Mean Squared Error (MSE) loss and visual alignment of predictions with ground truth. The Graph Attention Network (GAT) also showed strong performance, outperforming the simpler GCN and GIN models. The GIN model, in our experiments, exhibited training instability and poorer generalization performance compared to the other architectures.

The superior performance of GraphTransformer and GAT highlights the benefit of incorporating more sophisticated message passing mechanisms, particularly attention, for molecular property prediction tasks. The GraphTransformer's explicit consideration of edge features further suggests that leveraging bond information is crucial for accurately capturing molecular characteristics relevant to the dipole moment.

While this project achieved its objective of comparing different GNN architectures, it also revealed several avenues for future research and improvement:

- Hyperparameter Optimization: A more systematic hyperparameter search for each GNN architecture could potentially identify optimal settings and further enhance their predictive performance. This is especially relevant for the GIN model, which showed signs of instability and might benefit from careful hyperparameter tuning.

- Deeper and Wider Architectures: Exploring deeper GNN architectures (more convolutional layers) and wider layers (larger hidden channel dimensions) could be investigated to assess if model capacity can be further increased to capture more complex molecular patterns.

- Physics-Informed Graph Neural Networks (PIGNNs): A promising direction for future research is the development and application of Physics-Informed GNNs (PIGNNs). Incorporating physics-based knowledge, such as electronegativity as node features and bond lengths as edge features (as initially discussed in this project), could lead to models that are not only more accurate but also more physically interpretable and data-efficient. Designing GNN architectures that explicitly enforce physical constraints or symmetries could be a valuable next step. More advanced PIGNN approaches could involve loss functions that directly incorporate physical principles or hybrid models combining GNNs with classical force fields.

- Exploration of Other QM9 Properties: Future work could extend this project to predict other quantum chemical properties from the QM9 dataset, or even explore multi-task learning approaches to predict multiple properties simultaneously.

- Larger and More Diverse Datasets: Evaluating these GNN architectures on larger and more diverse molecular datasets beyond QM9 would be important to assess their generalization capabilities and robustness in more realistic scenarios.

- In conclusion, this mini-project demonstrates the effectiveness of Graph Neural Networks, especially attention-based architectures like GraphTransformer and GAT, for molecular property prediction. The findings suggest that future research focusing on physics-informed GNNs and further architectural refinements holds significant potential for advancing the field of molecular machine learning and enabling more accurate and efficient computational molecular design.