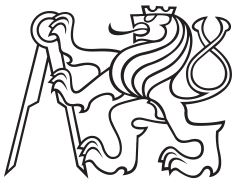
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Deep learning for computational chemistry with differentiable background knowledge

Emir Hodžić

Supervisor: Ing. Gustav Šír, Ph.D.
Field of study: Artificial Intelligence
Study program: Open Informatics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hodžić Emir** Personal ID number: **510705**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

Deep learning for computational chemistry with differentiable background knowledge

Master’s thesis title in Czech:

Hluboké učení pro výpočetní chemii s diferencovatelnou doménovou znalostí

Guidelines:

The subject of this master's thesis is to develop deep learning architectures for generic molecular prediction tasks. Particularly, the student is expected to extend the modern Graph Neural Network principles [1], which constitute the current state-of-the-art, with various background knowledge from the domain of chemistry. An existing framework for differentiable logic programming [2], allowing to use expressive relational logic for encoding both complex domain knowledge and neural models, should be utilized for the task. The student is expected to:
1) Review the domain of computational chemistry [3], existing frameworks, and state-of-the-art over various molecular datasets.
2) Review the principles of Graph Neural Networks [1] and Lifted Relational Neural Networks [2], and get acquainted with the framework [4].
3) Propose suitable modes of incorporating chemical background knowledge into GNN models. Focus on structural generalizations of the GNN principles, not just features.
4) Evaluate proposed modeling concepts on common molecular benchmarks.

Bibliography / sources:

[1] Zhou, Jie, et al. "Graph neural networks: A review of methods and applications." AI Open 1 (2020): 57-81.
[2] Šourek, Gustav, Filip Železný, and Ondřej Kuželka. "Beyond graph neural networks with lifted relational neural networks." Machine Learning 110.7 (2021): 1695-1738.
[3] Goh, Garrett B., Nathan O. Hodas, and Abhinav Vishnu. "Deep learning for computational chemistry." Journal of computational chemistry 38.16 (2017): 1291-1307.
[4] NeuraLogic framework: https://github.com/GustikS/NeuraLogic

Name and workplace of master's thesis supervisor:

Ing. Gustav Šír, Ph.D. Intelligent Data Analysis FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **31.01.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Gustav Šír, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I am eternaly grateful to my parents for their immeasurable love and support, alongside my siblings and the rest of my family, friends and close ones for their understanding and encouragement.

I am grateful to my mentor for his patience and willingness to guide me. I am grateful to everyone and everything that helped me in my path so far.

Finally, I thank myself for pushing through and promise to make everyone, including myself, proud.

Declaration

I hereby declare this thesis was written solely by myself, and all used literature is properly cited.

In Prague, 26. May 2023

Abstract

This thesis explores the application of Graph Neural Networks (GNNs) in computational chemistry, aiming to enhance the accuracy and predictive capabilities of machine learning models for molecular properties.

The study investigates state-of-the-art GNN architectures and proposes novel implementations leveraging relational learning techniques. By integrating domain-specific chemical rules into the models, the results indicate improved accuracy and predictive power on some benchmark datasets. This knowledge-based approach taps into the vast potential for refining and expanding the set of chemistry concepts encoded in relational rules, allowing for a better representation of complex chemical systems

This thesis contributes to the growing field of computational chemistry by demonstrating the effectiveness of GNNs augmented with a chemistry knowledge base. The findings offer valuable insights for researchers and practitioners seeking to leverage machine learning in the discovery of new molecules and materials, with the potential for transformative advancements in drug development and chemical research.

Keywords: chemistry, molecule, deep learning, background knowledge

Supervisor: Ing. Gustav Šír, Ph.D.

Abstrakt

Tato práce se zabývá použitím grafových neuronových sítí (GNN) ve výpočetní chemii s cílem zvýšit přesnost a predikční schopnosti modelů strojového učení pro molekulární data.

Studie zkoumá nejmodernější architektury GNNs a navrhuje nové implementace využívající techniky relačního učení. Díky integraci modelů a chemických pravidel, specifických pro danou oblast, výsledky ukazují na zvýšenou přesnost a prediktivní schopnost na některých referenčních souborech dat. Tento přístup založený na znalostech využívá potenciál pro zpřesnění a rozšíření souboru chemických pojmů zakódovaných v relačních pravidlech, což umožňuje lepší reprezentaci složitých chemických systémů.

Tato práce přispívá k rostoucímu oboru výpočetní chemie tím, že demonstruje efektivitu GNN rozšířených o bázi chemických znalostí. Zjištění nabízejí cenné poznatky pro výzkumné pracovníky a odborníky z praxe, kteří se snaží využít strojové učení při objevování nových molekul a materiálů, s potenciálem pro transformační pokrok ve vývoji léčiv a chemickém výzkumu.

Klíčová slova: chemie, molekuly, hluboké učení, postranní znalost

Contents

Figures

Tables

Chapter 1

Introduction

In recent years, there has been a significant paradigm shift in the types of data that are being used in machine learning and artificial intelligence applications. While traditional machine learning models such as linear and logistic regression are designed to handle tabular data, real-world problems often involve data that is not easily represented in this format. Graph data, on the other hand, provides a natural way to represent complex relationships between entities, making it ideal for handling many real-world problems. In a graph, data is represented as a set of nodes and edges, where each node represents an entity and each edge represents a relationship between two entities. Graphs can capture complex relationships and dependencies between entities, which is not possible with tabular data.

To tackle this shift, new models are being developed, among which are Graph Neural Networks. Graph Neural Networks (GNNs) are a class of deep learning models that are explicitly designed to process data represented as graphs. They work by representing each node in the graph as a feature vector, known as the node embedding, and then learning the relationship between the nodes and edges in the graph. This is achieved by iteratively updating the node embeddings through a series of layers, where each layer performs some computation on the node embeddings alongside the edge information.

GNNs have been successfully applied to a wide range of tasks in different domains of science, including chemistry [13], which is the focus of this thesis. Computational chemistry is an important branch of chemistry, which uses computational methods and simulations to study the properties and behavior of molecules and materials. It plays a vital role in new drug discovery with

software simulations or models that can predict the behavior of potential drugs and identify new drug candidates. It can also be used to study the interactions between drugs and their target molecules, which can lead to the development of new drugs with improved efficacy and safety.

It has already been shown that GNNs can effectively model complex atomic and intermolecular interactions, making them particularly useful in the field of computational chemistry. This makes them a powerful tool for predicting various properties of molecules, simplifying and directing the process of discovering molecules of interest. Tools such as GNNs serve as a guiding light in the discovery process, ensuring the scientists' creativity, focus and time is directed to more important stages following the discovery, which cannot yet be enhanced with machine learning methods and artificial intelligence.

However, despite their success, standard GNN models lack an inherent understanding of common chemical structures, such as rings or specific arrangements of atoms, which are crucial for accurately capturing molecular properties. This raises the question of whether the performance of GNNs can be enhanced by incorporating domain-specific background knowledge that directs the model to effectively learn and represent these chemical structures.

Integrating domain-specific knowledge has been proven to be an effective approach for improving model performance in various fields. In the domain of chemistry, background knowledge offers valuable insights into the structural and functional aspects of molecules. By leveraging this knowledge, GNNs can effectively capture relevant chemical features and relationships, leading to more precise predictions. For example, understanding the significance of different atom types, bond types, and hybridization states allows for the design of informative graph representations that reflect the underlying molecular structure. This, in turn, facilitates the accurate modeling of complex atomic and intermolecular interactions.

In this thesis, the primary objective is to explore the capabilities of GNNs in more depth and investigate their potential for improving the accuracy in predicting the properties of molecules. To achieve this, the thesis will explore the current state of the art GNNs, and propose new architectures and implementations using relational learning. The models will also be enriched with chemical background knowledge to enhance their effectiveness. This will be done by leveraging a relational learning [24] framework to encode the explicit chemical structures into differentiable rules, which will guide the learning process. Finally, the proposed models will be evaluated on various benchmark datasets for computational chemistry.

Part I

Theoretical background

Chapter 2

Graph Neural Networks

Graph neural networks (GNNs) are a class of deep learning models that process data in the form of graphs [25]. Graphs are defined as a pair $\mathcal{G} = (V, E)$, where V is a set of vertices $v_i \in V$ and E is a set of pairs of vertices $(v_1, v_2) \in E, v_1 \neq v_2$, called edges [23]. These edges can be directed or undirected. The neighborhood of a node is a function $\text{nbh}(n)$ mapping the given node n to a set of nodes S, such that there exists an edge $(n, u) \in E$ for each $u \in S$. Only undirected edges will be considered here, which means that the edge representations (n, u) and (u, n) are equivalent. Then the GNN model is defined as a parametrized mapping from a graph and an associated set of node and edge features $\mathcal{X}$ to an output structure.

$$\text{GNN}_\theta : (\mathcal{G}, \mathcal{X}) \to \mathcal{Y} \tag{2.1}$$

Representing molecules as graphs will be a straightforward mapping from atoms in molecules to nodes in graphs, as well as from bonds between atoms to edges in the graph [28]. The rest of the information such as the atom type, bond type and spatial information can be encoded as node or edge features.

The input for GNNs is always a graph. The information within these structures are passed along to computation layers or modules, which in turn produce the output. Depending on the task at hand, the output can take on different forms. For a regression task, the value might be extracted from the embedding of the desired node [12], for example in age prediction task in social networks. Otherwise, the value might be extracted from the desired

edge, or even from some combination of different kinds of node and edge embeddings to create a unified graph embedding [38]. In classification tasks, the label is extracted analogously and used for classification on node, edge or graph level. The format of the output structure based on task type is given in the Table 2.1.

	Regression	Classification
Node	$\mathcal{Y} \in \mathbb{R}^{\lvert V\rvert \times K}$	$\mathcal{Y} \in \mathcal{L}^{\lvert V\rvert \times K}$
Edge	$\mathcal{Y} \in \mathbb{R}^{\lvert E\rvert \times K}$	$\mathcal{Y} \in \mathcal{L}^{\lvert E\rvert \times K}$
Graph	$\mathcal{Y} \in \mathbb{R}$	$\mathcal{Y} \in \mathcal{L}$

Table 2.1: Output structure based on the type of task, where $\mathcal{L}$ is the set of labels for classification tasks and K is the number of prediction features.

In computational chemistry, the most common tasks are molecular property prediction, molecular scoring and docking, molecular dynamics simulation, molecular optimization and generation, as well as others [36]. Molecular property prediction is a graph-level regression or classification task, where the properties of the whole molecule are induced based on the aggregated node and edge embeddings.

These tasks follow the empirical risk minimization approach in supervised learning, and are defined as an optimization over a dataset of N samples containing triplets of molecular graphs and their respective features and targets $\{(\mathcal{G}_1, \mathcal{X}_1, \mathcal{Y}_1), ..., (\mathcal{G}_N, \mathcal{X}_N, \mathcal{Y}_N)\}$ as in Equation 2.2.[36]

$$\min_{\theta} \sum_{i}^{N} l(\mathrm{GNN}_\theta(\mathcal{G}_i, \mathcal{X}_i,), \mathcal{Y}_i) \tag{2.2}$$

where l is the loss function. The loss function depends entirely on the task at hand, and it can be *mean-squared* loss for regression tasks, cross entropy for classification, etc.

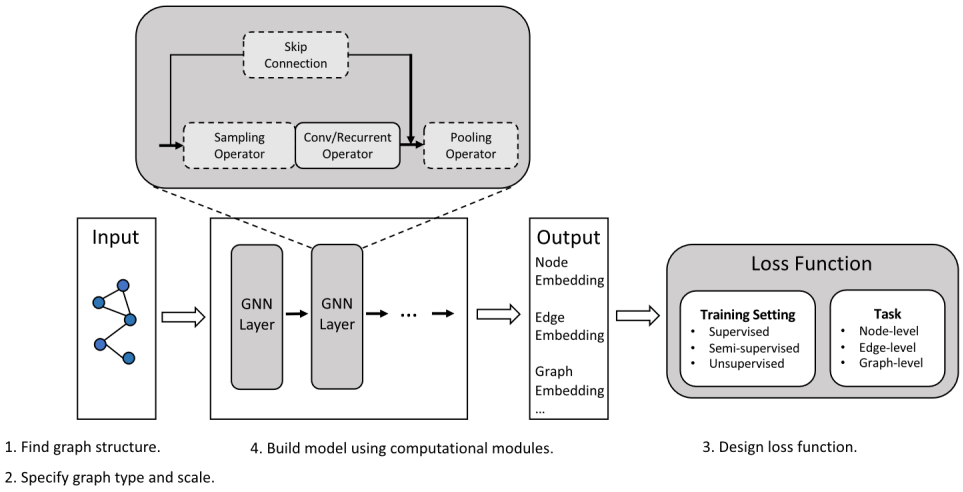


Figure 2.1: General model architecture, as well as the design pipeline in steps (image taken from [42]).

2.1 Architecture of GNNs

A GNN model is comprised out of GNN layers. These layers, in turn, can contain a combination of three types of computational modules (Figure 2.1):

- Propagation module - defines how features are propagated between layers
- Sampling module - extracts parts of graphs to propagate features from
- Pooling module - condenses information from node embeddings

Sampling modules are typically used for large graphs where there is redundant or insignificant information. Since the thesis is focused on implementations for use in working with molecules, which are typically relatively small and contain useful information in most nodes, sampling modules will be skipped. There are various pooling modules available to aggregate the information. Most used ones are the simple ones such as node-wise max, average, sum, etc.

2.1.1 Propagation Modules

Propagation modules are a fundamental component of GNNs that define rules for message passing through a graph. They are distinguished based on

the operation used for message propagation, which can be a convolutional operator, a recurrent operator, or a skip connection.

Convolutional operators in GNNs are inspired by Convolutional Neural Networks (CNNs) used for image processing, where a set of kernels or filters is used to propagate features [6]. With convolutional operators, weights are assigned to each node's embedding and updated based on the embeddings of neighboring nodes. These GNNs are then abstractions of CNNs, where an image is a graph represented in a rectangular grid of nodes, connected only to its immediate neighbors.

Convolutional operations can be understood and performed using two approaches: spatial and spectral [41]. In the spectral approach, graphs are considered as signals, and operations are done on the eigenvectors of the graph Laplacian matrix. These signals are transformed into the spectral domain using Fourier transform, and convoluted to learn the features. In the spatial approach, on the other hand, operations are done directly on the graph structure, and convolutions are defined based on topology. The challenge with spectral convolutions is defining the convolution on different-sized neighborhoods and maintaining local invariance of CNNs. However, this is easier to achieve using the spatial approach, which is why it will be the focus of this thesis.

Recurrent operators are another type of propagation module used in GNNs to capture long-range dependencies in the graph. In a recurrent operator, the hidden state of a node is updated based on its previous hidden state and the hidden states of its neighboring nodes. This allows the GNN to model the temporal evolution of a graph and capture complex interactions between nodes over time. The depth of message passing can be considered as propagation through time for some models.

Skip connections are used to improve the performance of deep GNNs by allowing the model to retain and propagate the original node features, instead of relying solely on the aggregated output. Using a skip connection, the output of a layer is enriched with its own input, and then it is forwarded to the next layer, allowing important information to be passed more easily to deeper layers and reinforced before it gets diffused into noise. This helps prevent information loss and enables the GNN to capture more complex features of the graph.

2.2 Weisfeiler-Lehman algorithm

The Weisfeiler-Lehman (WL) algorithm is a graph isomorphism test that is used to compare and distinguish graphs [9]. It utilizes color refinement by assigning a state or color to each node, which is then refined in each iteration by incorporating information from the neighboring nodes' states. The refinement process stabilizes after a few iterations and produces a graph representation. If two graphs have different representations, they are not isomorphic. While the test can accurately distinguish between a large set of graphs, there are certain cases where it fails [17] (Figure 2.2).

The Weisfeiler-Lehman algorithm is a crucial component in various Graph Neural Network (GNN) architectures and has proven to be a powerful tool for analyzing the expressive capabilities of GNNs. An extension of this algorithm involves maintaining the state not only of individual nodes but also of k-tuples of nodes. These algorithms are called k-dimensional Weisfeiler-Lehman or k-WL test. The k-WL test can then serve as a benchmark for characterizing the expressive power of algorithms including GNNs.

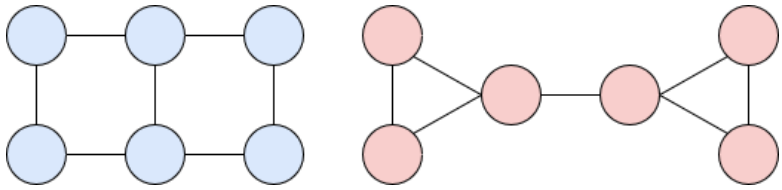


Figure 2.2: A simple example of non-isomorphic graphs that cannot be distinguished by WL algorithm. [20] [27]

GNNs are shown to be as expressive as 1-WL [39], meaning they are unable to distinguish between the mentioned graphs. This presents an issue in computational chemistry, as there are many molecules which are indistinguishable by 1-WL algorithm. An example can be seen in Figure 2.3.

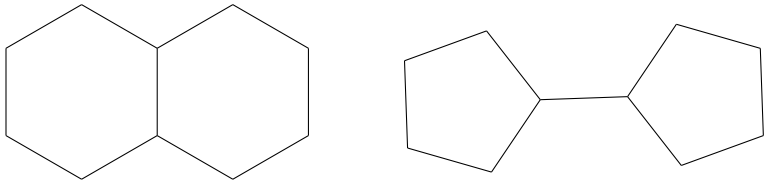


Figure 2.3: An example of two distinct molecules, decalin and bicyclopentyl, which cannot be distinguished by 1-WL [27].

Chapter 3

Relational learning

Relational learning is a machine learning paradigm that focuses on modeling and reasoning about relationships between entities [22]. This approach has some significant advantages, such as the ability to naturally represent complex data in a form which is easily understandable [10]. There has already been some successful attempts to combine machine learning on graphs with relational data [8], including the framework used in this thesis.

The learning framework that will be used is based on the Lifted Relational Neural Networks (LRNNs)[32]. The idea is to use differentiable logic programs for encoding relationships among the data, lifting them into a higher level of abstraction. LRNNs use a set of relational reasoning layers to process the graph data. Each relational reasoning layer is composed of a set of relational modules, which are used to update the node representations based on the relations between the nodes. Each relational module is a neural network layer that is applied to a set of nodes and edges, and it is designed to learn the relations between the nodes and edges in the graph.

3.1 First-order Logic

First-order logic is used as the representation formalism for LRNNs. The description of first-order logic considered in this thesis will be identical to the one in [31]. This means that the function-free first-order logic is used, with formulas defined on a set of constants, a set of variables and a set of n-ary

predicates, as well as propositional connectives $\vee$, $\wedge$ and $\neg$ [30]. All formulas and variables will be implicitly universally quantified, without the quantifiers being written. Constants are written lowercase, while variables are written capitalized.

Terms are constants or variables, atoms are n-ary predicates applied to terms, and ground atoms have only constants as arguments. A literal is either an atom or the negation of an atom, while a clause is a disjunction of literals. A definite clause is a clause with exactly one positive literal, also called a rule. A clause consisting of a single atom is also called a fact, and a set of definite clauses is called a logic program.

The positive literal in a rule is called the head of the rule, while the rest are called the body. It is written as $h \vee \neg b_1 \vee ... \vee \neg b_n$, but can also be represented by an implication instead:

$$h \leftarrow b_1 \wedge ... \wedge b_n$$

The set of first-order formulas $\mathcal{P}$ has a *Herbrand base*, which consists of all ground atoms that can be formed using the constants and predicates in $\mathcal{P}$, while respecting predicate arities. A *Herbrand interpretation*, assigns a truth value to each ground atom in $\mathcal{P}$'s Herbrand base [34]. It is said that a Herbrand interpretation I satisfies a ground atom F, written as $I \models F$, if $F \in I$. A set of ground formulas is satisfiable if there exists a *Herbrand model* in which all formulas from the set are true. The unique Herbrand model that is minimal with respect to the subset relation is the least Herbrand model of the set of definite clauses.

3.2 Logic Programming

Logic programming is a programming paradigm that uses logic programs to perform computation in a declarative manner. In this paradigm, definite clauses express facts and rules related to a specific domain, and logical inference is used to perform computation. The rules in the program are expressed as `h :-` b_1`, ...,` b_k, where each `","` represents a conjunction, and `":-"` is used in place of the implication operator. Facts are simply rules without a body.

Logic programs are a type of computer programs that are written using formal logic, first-order logic in this case. In a logic program, a set of facts and

rules that represent the problem are specified, and these facts and rules are used to automatically deduce a solution. The facts and rules are represented in a declarative manner. The main advantage of logic programming is that it allows for natural and expressive representation of knowledge, making it easier for humans to understand and reason about the problem.

The advantage of this framework is that the relations and rules are weighted, and therefore the impact of such constructs can be learned with gradient descent, as the functions represented by these rules become differentiable. This, in turn, allows the expression of chemical structures, as well as different kinds of GNN architectures, under a simple and unified representation.

3.3 Rule structure

The syntax is inspired by Datalog, with the addition of numerical parameters. Datalog is a restricted, function-free subset of Prolog [3] that is a domain-specific language used in advanced deductive database engines. Unlike Prolog, Datalog [33] is a truly declarative language, where the order of clauses does not impact execution, and it is guaranteed to terminate. Datalog allows for the creation of general and reusable programming patterns, which can be bound to different ground data structures via variable substitution, which is then key to extending it towards differentiable programming [32].

For that purpose, instead of being pure binary identities, the logical literals have an associated tensor of weights. The learning examples are sets of weighted ground facts in the following form:

$$V :: p(c_1, ..., c_2)$$

where V is a real-valued tensor, p is a predicate and c_i's are constants. The rules are then defined as follows:

$$W :: h(...) : -W_1 : b_1(...), ..., W_k : b_k(...).$$

where W's are tensors of associated weights, h and b's are predicates. These rules are composed into a logic program, called *templates* in this framework.

3.4 Architecture representation

The inputs to the models defined by the rules are examples, which are logic representations of graphs. They are composed of rules which define the given graph, for example a water molecule might be encoded as follows, where the predicate `a` represents the atom embedding and predicate `b` represents a bond:

```
a(o1), a(h1), a(h2), b(o1, h1), b(o1, h2),
b(h1, o1), b(h2, o1).
```

The sets of rules defining a weighted logic program are referred to as templates, and they correspond to a mode of computation related to a model architecture. From these templates, computational graphs are constructed for each example, as explained in Section 7.2. An example of a simple template, as well as the constructed computational graph on two examples are given in Figure 3.1.

The framework requires a goal query to be defined, which means that all predicates necessary for computing the answer have to imply the goal predicate. Similar to queries in databases, queries in logical programming are atoms which drive the computation to a specific target, finding which model of the logic program satisfies the query. This is used as a final computation of the classification labels or regression targets associated with an example for supervised learning for all examples in the dataset.

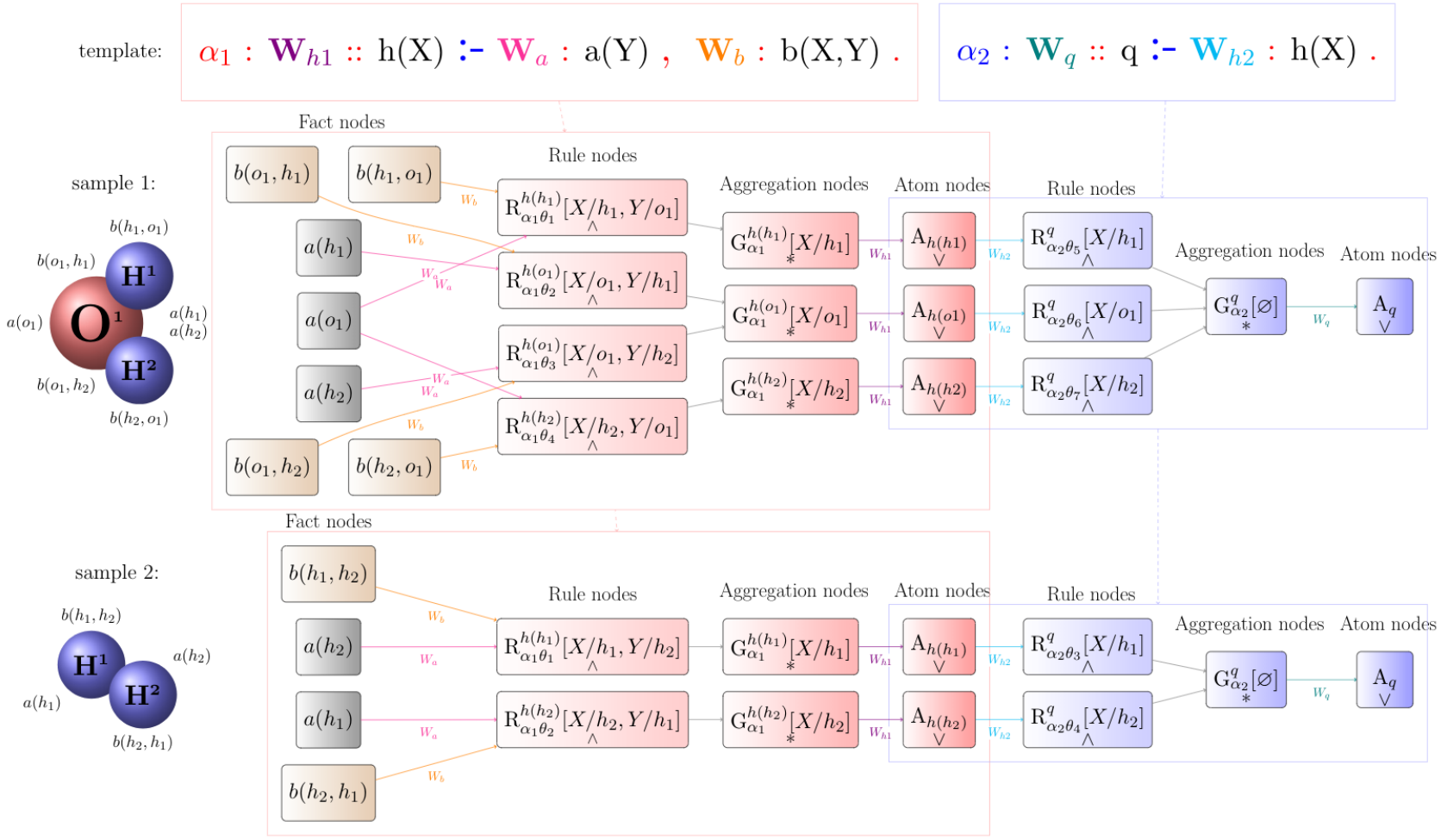


Figure 3.1: Simple template grounding on two examples. This figure is taken from [32].

Chapter 4

Organic Chemistry

Organic chemistry is the study of carbon-containing compounds, which are structures that form the backbone of most biological molecules [4]. Carbon has the unique ability to form stable covalent bonds with other atoms, including other carbon atoms, giving rise to a nearly infinite number of possible molecules with different shapes and chemical properties. This ability of carbon is due to its electron configuration. Carbon has four valence electrons, which means it can form up to four covalent bonds with the neighboring atoms.

Molecules are three-dimensional structures with diverse geometries, which have to be simplified for two-dimensional representation. In illustrations, the atoms are represented by their abbreviation, and the bonds are depicted as different kinds of lines between them. One important thing to notice is that hydrogen atoms are mostly left out of the illustration completely, and carbon chains are reduced to a pattern of connected edges, where each connection point infers a carbon atom, connected to as much hydrogen atoms as needed to ensure a stable valence shell. Both forms of illustration will be used, depending on the example. The remainder of the organic molecule is denoted as **R**, which is usually a carbon atom connected to any other combination of atoms and groups, but also can be a hydrogen atom in some cases.

4.1 Functional groups

In organic chemistry, molecules have a distinct classification, based on the molecular substructures called functional groups, which are strictly defined by the specific arrangement and types of atoms, as well as the bonds between them. These groups determine the molecule's chemical reactivity and properties [35].

The presence of a specific functional group defines which reactions a molecule can participate in, as well as other physical and chemical properties, such as solubility, boiling point, acidity, etc. For example, an alcohol functional group (-OH) can form hydrogen bonds with water molecules, making alcohols soluble in water. In contrast, molecules with non-polar functional groups, such as hydrocarbons, are typically insoluble in water and more soluble in organic solvents.

This is due to a property of molecules called *polarity*, and it happens because the functional groups consist of atoms which are mostly different from the base part of the molecule, and different atoms infer differences in the overall distribution of electric charge in the molecule. The ability of an atom to attract electrons in a chemical bond is called electronegativity. In a polar covalent bond, the electrons are shared unequally between the two atoms, with the more electronegative atom attracting the electrons closer to itself. This creates a partial negative charge on the more electronegative atom and a partial positive charge on the other atom.

4.1.1 Hydrocarbons

Hydrocarbons are a class of organic molecules comprised solely of carbon and hydrogen atoms. They are the most essential forms of organic compounds, serving as scaffolding for formation of other classes of organic compounds. Hydrocarbons are mostly stable, but are susceptible to substitution and addition reactions [5]. Aliphatic hydrocarbons typically consist of straight or branched chains of carbon atoms and can be further divided into three groups: alkanes, alkenes, and alkynes. Alkanes have single covalent bonds between their carbon atoms, while alkenes and alkynes have at least one double or triple bond, respectively. Their structure is shown in Figure 4.1. All of the mentioned types can also be cyclic.

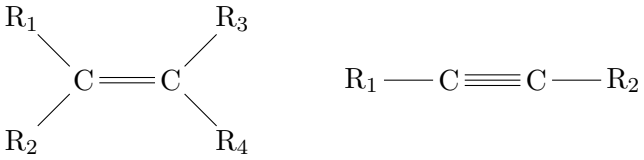


Figure 4.1: Alkene and alkyne bonds respectively.

Cyclic hydrocarbons are another important type of hydrocarbons which can be aliphatic or aromatic. Aliphatic cyclic hydrocarbons, are hydrocarbons that contain a ring of carbon atoms with aliphatic bonds between them. Aromatic hydrocarbons, on the other hand, are a unique class of cyclic hydrocarbons that contain at least one aromatic ring in their structure. Aromaticity is a fairly complicated property, which makes these compounds much more stable than similar structures [19]. It is defined as having all bonds in resonance, which means that the electrons forming bonds are delocalized between the atoms. An example would be the benzene ring, which is a molecule by itself, but is also a common substructure in more complicated molecules. The delocalization of electrons in the benzene ring gives it exceptional stability and unique chemical properties.

Aliphatic substructures present as functional groups are denoted as alkyl groups, and aromatic ones as aryl.

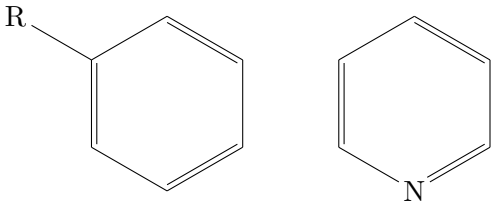


Figure 4.2: Benzene (phenyl) group, and pyridine, an example of heterocycles.

It is worth noting that some *heterocycles*, which are rings that, alongside carbon atoms, contain atoms other than carbon, can also be aromatic if they meet certain criteria for aromaticity. Examples of heterocyclic aromatic compounds include pyridine (Figure 4.2), pyrimidine, and purine, which are important building blocks for many biological molecules, such as DNA and RNA.

4.1.2 Oxygen-containing compounds

Oxygen-containing functional groups play an essential role in organic chemistry, influencing the reactivity and physical properties of organic compounds. The most significant functional groups that will be covered here include alco-

hols, aldehydes, ketones, acyl halides, carboxylic acids and their anhydrides, ethers, esters and carbonate esters.

Alcohols consist of an oxygen atom bonded to a carbon atom and a hydrogen atom, as shown in Figure 4.3. One important feature of the carbon atom in alcohols is that it has to be saturated - meaning it is bonded only by single bonds.

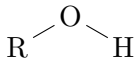


Figure 4.3: Alcohol (hydroxyl) group.

Carbonyls consist of a carbon atom double-bonded to an oxygen atom. Carbonyl groups are found, among others, in aldehydes and ketones, which are commonly used in organic synthesis. Aldehydes have a carbonyl group at the end of a carbon chain, while ketones have a carbonyl group in the middle of a carbon chain. Acyl halides are carbonyls with a halogen as one of the substituents, denoted as **X** in the Figure 4.4.

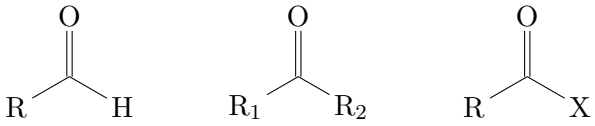


Figure 4.4: Aldehydes, ketones and acyl halides.

Carboxylic acids are also important oxygen-containing functional groups, consisting of a carbonyl group and a hydroxyl group on adjacent carbon atoms. **Carboxylic acid anhydrides**, which consist of two carbonyl groups bonded to the same oxygen atom, are formed from the reaction of two carboxylic acid molecules. This is seen in Figure 4.5.

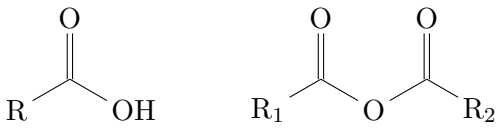


Figure 4.5: Carboxylic acids and their anhydrides.

Ethers, seen in Figure 4.6, are carbon chains connected by a single oxygen atom. They are relatively inert and unreactive, and they are commonly used as solvents for organic reactions.

Esters, which consist of a carbonyl group and an ether group on adjacent carbon atoms, are commonly used in perfumes, flavors, and solvents. And **carbonate esters** consist of a carbonyl group surrounded by two ether groups.

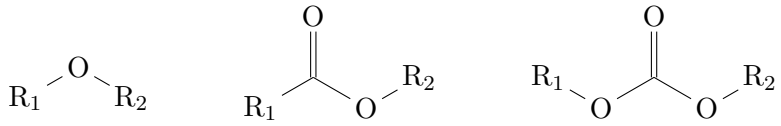


Figure 4.6: Ethers, esters and carbonate esters.

4.1.3 Nitrogen-containing compounds

Nitrogen is another important element in organic chemistry, as it can form a variety of substructures that play key roles in organic processes. The most important functional groups containing nitrogen include amines, amides, imines, imides, azides, azo-compounds, cyanates, nitro compounds, nitrates, carbamates and azidrines.

Amines are compounds with a nitrogen atom bonded to one or more alkyl or aryl groups. Amines can be classified as primary, secondary or tertiary, depending on the number of alkyl or aryl groups attached to the nitrogen atom. An important derivative is also the quaternary ammonium cation, which has 4 groups attached to it (Figure 4.7).

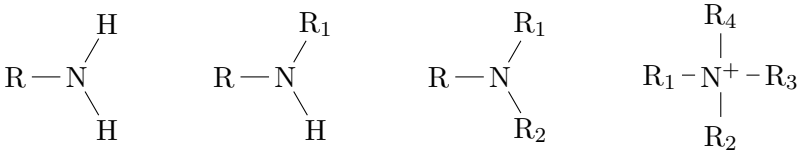


Figure 4.7: Primary, secondary, tertiary amines and quaternary ammonium ions.

Amides are derivatives of carboxylic acids. Amides contain a nitrogen atom bonded to a carbonyl group and an alkyl or aryl group. **Carbamates** are essentially an amide group, where the substituent on the carbonyl group is an oxygen.

Imines have a nitrogen double bonded to a carbon atom, while **imides** are comprised of two carbonyl groups connected by a nitrogen atom. They are somewhat analogous to carbonyls and acid anhydrides respectively. The structure of these compounds is shown in Figure 4.8.

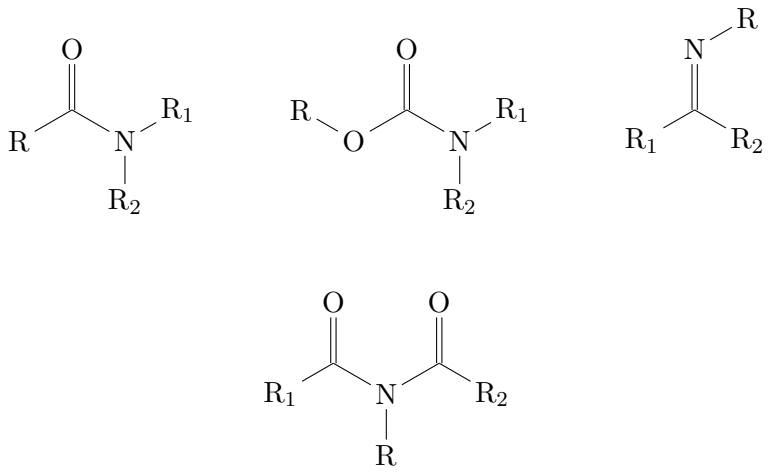


Figure 4.8: Amides, carbamates, imines and imides.

Azides contain a carbon atom attached to three nitrogens in a linear arrangement. **Azo compounds** contain two nitrogen atoms linked by a double bond. These structures, as shown in Figure 4.9, are also known as nitrogen bridges. **Cyanates** are composed of a nitrogen atom, carbon atom and an oxygen atom in a linear fashion, and depending on the arrangement of these, they can be cyanates (-OCN) or isocyanates (-NCO).

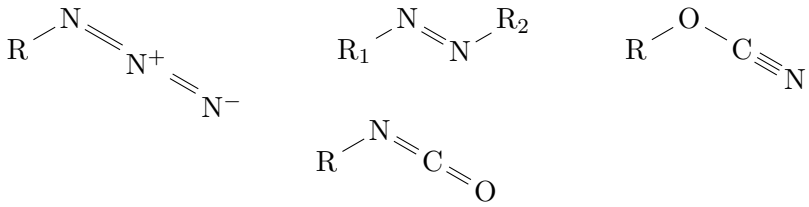


Figure 4.9: Azides, azo compounds (nitrogen bridge), cyanates and isocyanates.

Nitro compounds contain a nitro group ($-NO_2$) bonded to a carbon atom. The nitro group is highly polar, making nitro compounds relatively reactive. **Nitrates** contain the nitrate ion (NO_3^-) bonded to a carbon atom, as seen in Figure 4.10.

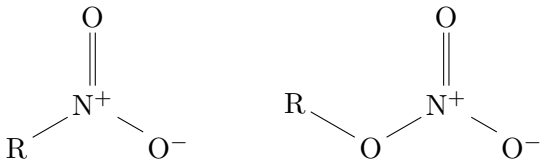


Figure 4.10: Nitro and nitrate group.

Azidrines (Figure 4.11) are three-membered heterocyclic compounds containing a nitrogen atom and two carbon atoms in a cyclic ring structure.

They are highly reactive and can undergo a variety of reactions, making them useful in organic synthesis.

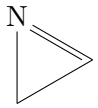


Figure 4.11: Azidrine.

4.1.4 Sulfur-containing compounds

Sulfur is another element which plays an important role in organic chemistry. Some of the most significant functional groups containing sulfur include thiocyanate, sulfide, disulfide, and thiol groups.

Thiocyanates are composed of a sulfur atom bonded to a carbon atom, which is then attached to a nitrogen atom. Isothiocyanates have the nitrogen connected to the rest of the molecule instead of the sulfur atom. This is shown in Figure 4.12.

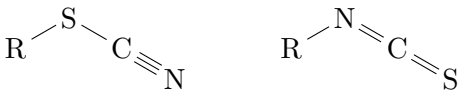


Figure 4.12: Thiocyanates and isothiocyanates.

Thiols are organic compounds that contain a sulfur atom bonded to a hydrogen atom (-SH). **Sulfides** contain a sulfur atom that is covalently bonded to two carbon atoms. Similarly, **disulfides** consist of two sulfur atoms covalently bonded together. Disulfides are often found in proteins and are essential for the folding and stability of many proteins. These structures are also called sulfur bridges (Figure 4.13).

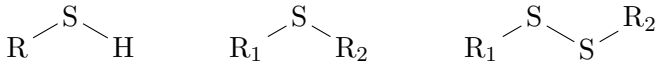


Figure 4.13: Thiols, sulfides and disulfides.

Part II

Literature Review

Chapter 5

Datasets

The datasets that the templates will be tested on are common benchmarks for cheminformatics models. They will be briefly described in this chapter.

All the datasets share the same structure, where each entry is a single graph composed of nodes and edges. In addition to that, each node and edge has its label. For edges the labels represent if it is a single, double, triple or aromatic bond in some datasets. Node labels represent the atom type (carbon, oxygen, nitrogen, etc.). Hydrogen atoms are left out completely in some of the datasets.

The *MUTAG* dataset [7] is a commonly used benchmark dataset, containing 188 molecules. The molecules are classified as mutagenic or non-mutagenic.

The *PTC* [16] dataset consists of chemical compounds which have been classified as either positive or negative for carcinogenicity in rodents, which is a standard assay used to predict the potential mutagenic effects of chemicals. There are four variants of the PTC dataset, namely PTC MR, PTC FR, PTC MM and PTC FM.

The *PTC MR* dataset consists of 344, and similarly the *PRC FR* dataset consists of 351 compounds chemical compounds that have been classified as either positive or negative for carcinogenicity in male and female rats, respectively. On the other hand, *PTC FM* and *PTC MM* datasets contain 349 and 336 compounds tested for carcinogenicity on female and male mice.

Chapter 6

State-of-the-art models

In this chapter, some of the most successful state-of-the-art models will be introduced formally, while in the Appendix B, it will be shown how they were implemented.

6.1 Standard Graph Neural Networks

The Graph Neural Network model gives an intuitive message passing framework, and they operate by propagating information through the graph's nodes and edges to capture the structural dependencies and relationships present in the data. The hidden state of a node in a GNN is calculated using a recursive update mechanism that combines information from the node's neighbors [37].

The node state is given by h_n, and it is used as the layer output for the given node. The node and edge features are given by $x_i \in \mathcal{X}_{node}$ and $x_{(i,j)} \in \mathcal{X}_{edge}$, while f is a function aggregating the features and previous layer output, and σ is an element-wise non-linear function.

$$h_n^{l+1} = \sigma\Big(\sum_{u \in \mathrm{nbh}(n)} W^l f(h_u^l, x_u, x_n, x_{(n,u)}) + W_0^l h_n^l\Big) \tag{6.1}$$

In this equation, the sum iterates over the neighboring nodes of node n. Within the summation, the function f combines the hidden state of the neighboring node h_u^l, the node features x_u and x_n, and the edge features $x_{(n,u)}$. The weights W^l and W_0^l are learnable parameters specific to layer l. The resulting sum is then combined with the previous layer's hidden state h_n^l through the weight W_0^l.

By applying this update equation iteratively for multiple layers, the GNN can capture and propagate information across the graph structure, enabling the nodes to incorporate both local and global information from their neighbors. The non-linear activation function σ introduces non-linearity into the node updates, allowing the GNN to learn complex patterns and relationships in the graph data.

6.2 Relational Graph Convolutional Networks

Relational Graph Convolutional Networks were introduced in [29] for the task of link prediction in knowledge bases. The model is defined as an autoencoder with R-GCNs as encoders which produce latent feature representations and a decoder which is a tensor factorisation model.

They extend the concept of GNNs by explicitly considering the types of relations between nodes in a graph. R-GCNs incorporate relation-specific weights to capture the varying influence of different types of relations on the node representations.

The model requires a function which determines the edge type, called relation in the model. For this purpose, we will introduce a function mapping an edge to a relation from the set of relations or edge types: $rel(n, m) : (n, m) \in E \to \mathcal{R}$.

Then the relational neighborhood of a node will be defined as a function $\text{nbh}_r(n)$ mapping the given node n and the relation r to a set of nodes S, such that there exists an edge $(n, u) \in E$, and the type of the edge given by $rel(n, u) = r, \forall u \in S$.

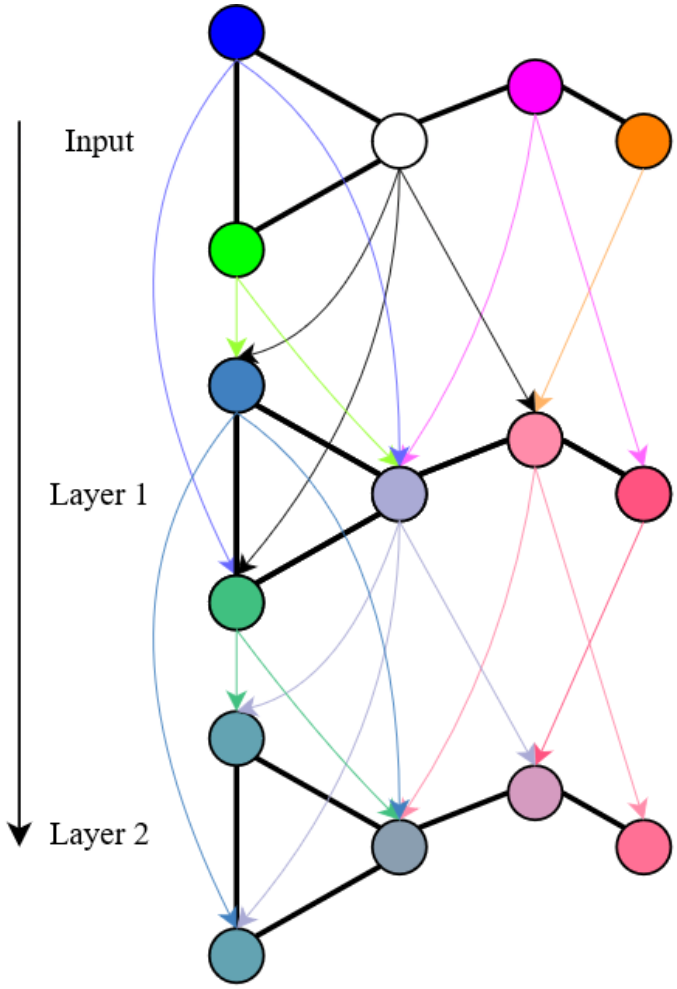


Figure 6.1: Message passing in standard GNNs. The colors are representing node features, and how they propagate throughout the graph. Each color in the next layer is composed of a combination (in this case the average) of the colors in the previous layer. Which colors are used for the computation is denoted by the colored arrows.

$$h_n^{l+1} = \sigma(\sum_{r \in \mathcal{R}} \sum_{u \in \text{nbh}_r(n)} W_r^l f(h_u^l, x_u, x_n, x_{(n,u)}) + W_0^l h_n^l) \tag{6.2}$$

In this equation, the outer summation iterates over the relations, capturing the different types of relations in the graph. Within each relation, the inner summation iterates over the neighboring nodes connected to node n through relation r. The hidden state h_u^l represents the node state of node u at layer l.

The inner summation term calculates the contribution of the neighboring node u to the updated state of node n through relation r. The weight matrix

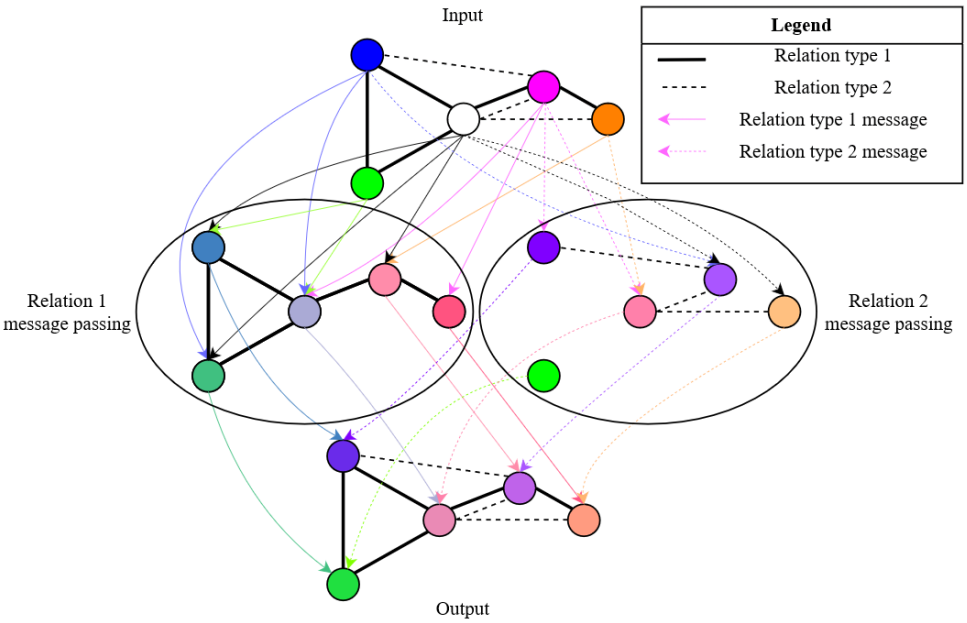


Figure 6.2: Message passing in an RGCNs layer. The different edge types, drawn by full and dashed lines, represent different types of relations. The messages are first passed between nodes based on relations, and then aggregated together.

W_r^l captures the relation-specific influence and is applied to the hidden state h_u^l. The resulting sums across all relations are combined through element-wise summation.

Finally, the combined sum is passed through the activation function σ to introduce non-linearity and produce the updated node state $h_n^{(l+1)}$ at layer $l+1$.

6.3 Higher-order Graph Neural Networks

Higher-order GNNs, introduced in [21], are based on the higher-order or k-WL algorithm, where the nodes are grouped in sets of k nodes, and their output is calculated in relation to other such sets. We will define a set of valid k-tuples S^k, such that $\forall s = (s_1, ...s_k) \in S^k, s_i \neq s_j, \forall i, j = \{1, ..., k\}, i \neq j$. For a k-tuple $n \in S^k$, the k-neighborhood is defined to be a function $\text{nbh}_G^k(n)$ which maps n to a set of k-tuples $s \in S^k$, such that $|n \cap s| = k - 1$.

$$h_n^{k,l+1} = \sigma\Big(\sum_{u \in \mathrm{nbh}_G^k(n)} W^l f(h_u^{k,l}, x_u, x_n, x_{(n,u)}) + W_0^l h_n^{k,l}\Big) \tag{6.3}$$

This equation defines a *global* k-GNN. There exists a *local* variant, which allows these to be scaled to large graphs, where a notion of local neighborhoods is defined, which is a function $\mathrm{nbh}_L^k(n)$ which maps n to a set of k-tuples $s \in \mathrm{nbh}_G^k(n)$, such that $i \in n \setminus s, j \in s \setminus n : (i,j) \in E$. This variant will be defined by the following equation.

$$h_n^{k,l+1} = \sigma\Big(\sum_{u \in \mathrm{nbh}_L^k(n)} W^l f(h_u^{k,l}, x_u, x_n, x_{(n,u)}) + W_0^l h_n^{k,l}\Big) \tag{6.4}$$

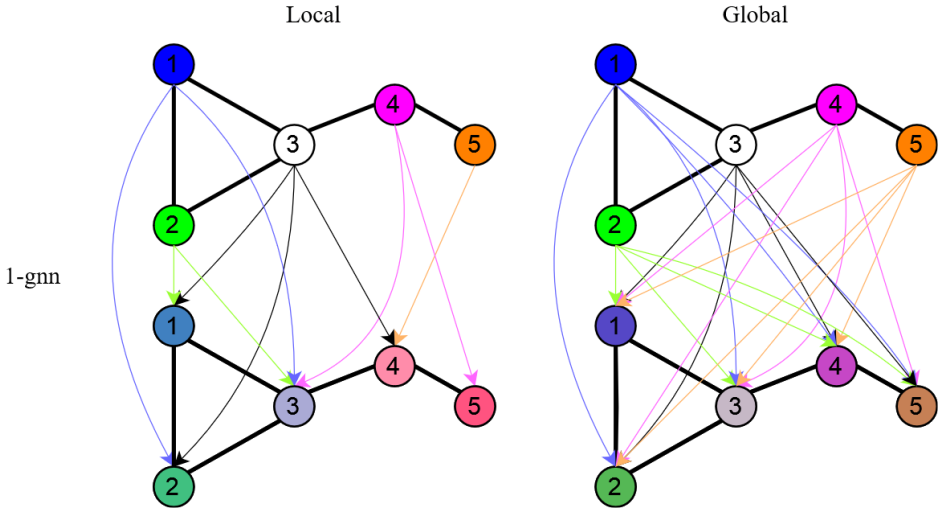


Figure 6.3: Message passing in a k-GNN layer and the difference between local and global variants.

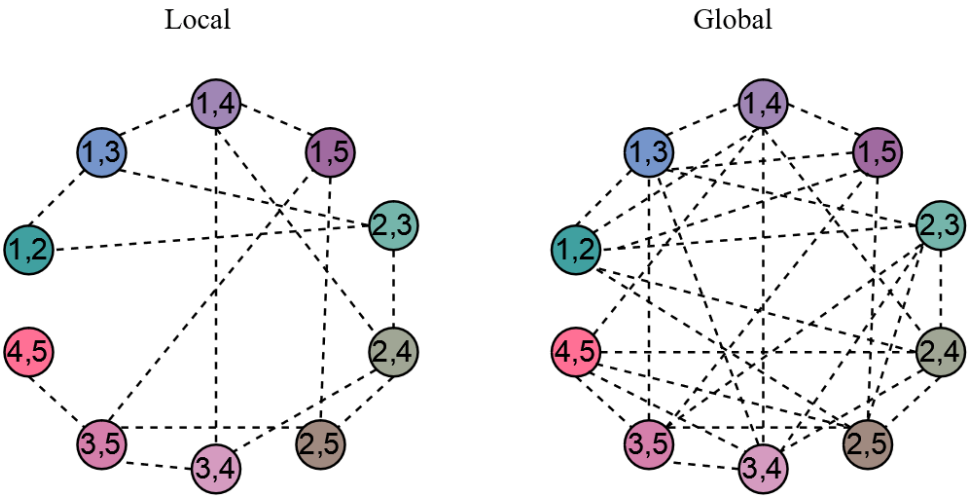


Figure 6.4: The difference between the connectivity in a local and global 2-GNN, constructed from the same original graph.

The output of a k-GNN can then be used as input for a $k+1$-GNN.

6.4 Ego-Graph Neural Networks

Ego-GNNs were motivated by the 1-WL boundary on the expressiveness of standard GNNs, an example being the inability of standard GNNs to recognize the presence of closed triangles. This is generalized to conclude the standard GNNs are unable to distinguish the ego-graphs around each node [26], for which purpose Ego-GNNs are introduced, and they aim to capture the local neighborhood information and the ego-centric view of each node within a larger graph. Ego-graphs are subgraphs defined by a node and its immediate neighbors.

The model works by creating an ego-graph for each node, and passing messages between the nodes in this graph. Afterwards, the messages are passed between all the ego graphs, based of the connectivity of the main nodes in the ego graphs. If two nodes are connected in the original graph, their ego-graphs will transfer messages.

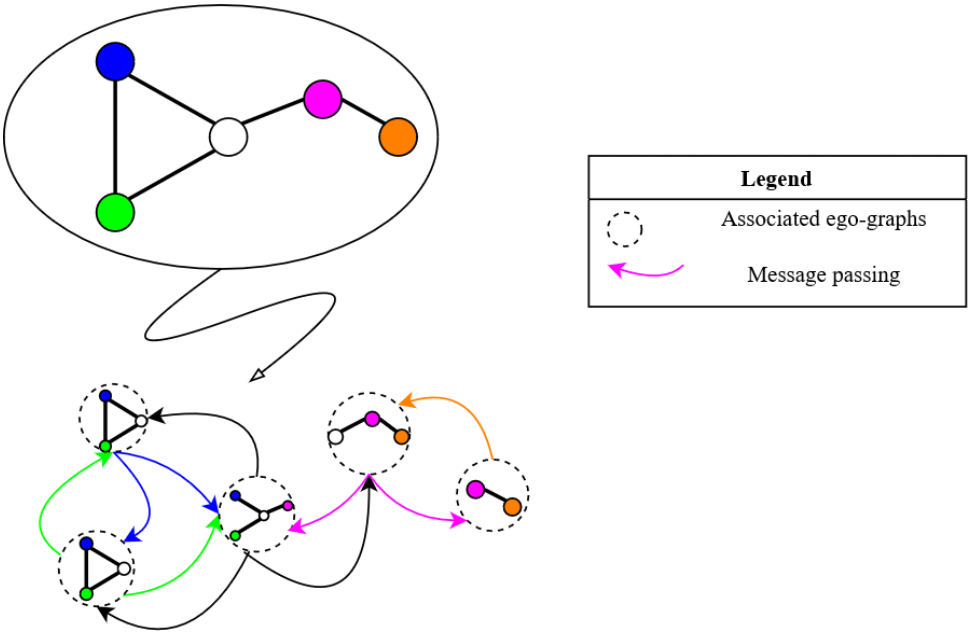


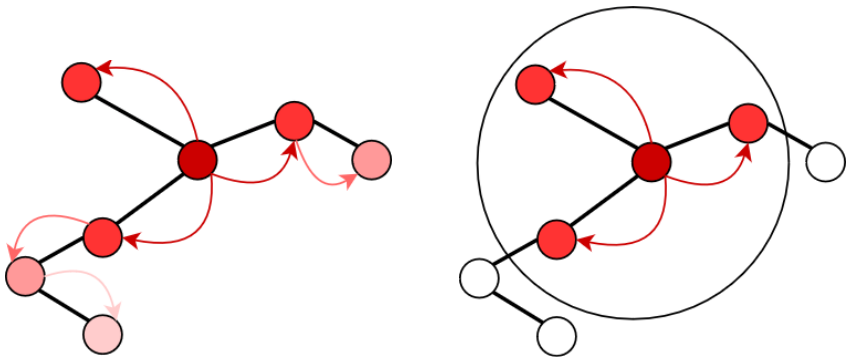
Figure 6.5: Message passing in an ego-GNN layer. The messages are passed inside of the ego-graph and then passed between them.

6.5 Diffusion Convolutional Neural Networks

Diffusion Convolutional Neural Networks [1] (DCNNs) are based on the idea of extending convolutional neural networks to apppropriate graph data using a diffusion-convolution operator. This operator builds a latent representation by calculating the diffusion of information across nodes in the graph, using a diffusion kernel, which is a measure of the level of connectivity between any two nodes in the graph when considering all paths between them.

We define D to be the maximal path length between two nodes, as calculating all possible paths would be computationally intensive, and a function $\text{path}(x, y, d)$, which returns all paths between x and y of length d. The function $f(p)$ aggregates all messages along the given path.

$$h_n^{l+1} = \sigma(\sum_{d=1}^{D} \sum_{u \in V} \sum_{p \in \text{path}(n,u,d)} W_d^l f(p) + W_0^l h_n^l) \tag{6.5}$$



(a) Message passing with unbounded path length.

(b) Message passing with bounded path length of one hop.

Figure 6.6: Message passing in a DCNN layer. This is a representation of how messages from one node (center node in this example) gets transmitted to others.

6.6 Cellular Weisfeiler-Lehman Networks

Cellular Weisfeiler-Lehman Networks [2] are based on algebraic topology, with message passing done across so-called regular cell complexes. In essence, cells represent topological objects, such that each n-cell is composed of a closed space defined by the $n-1$-cells.

Vertices are represented as 0-cells, from which edges or 1-cells are derived by enclosing the space between two vertices. From edges, 2-cells are derived as cycles of any size, which enclose a space between some number of edges. Cells of dimension higher than 2 are not covered, as they are generally not useful for our purpose.

The models utilizing the aforementioned structure are called CW networks, where the C stands for “closure-finite”, and the W for “weak” topology. There are two types of messages defined, where the first one passes information from lower to higher dimensional cells. The second one passes information between the cells which are a part of the same higher-dimensional cell. In this message, the information from the higher-dimensional cell is also aggregated.

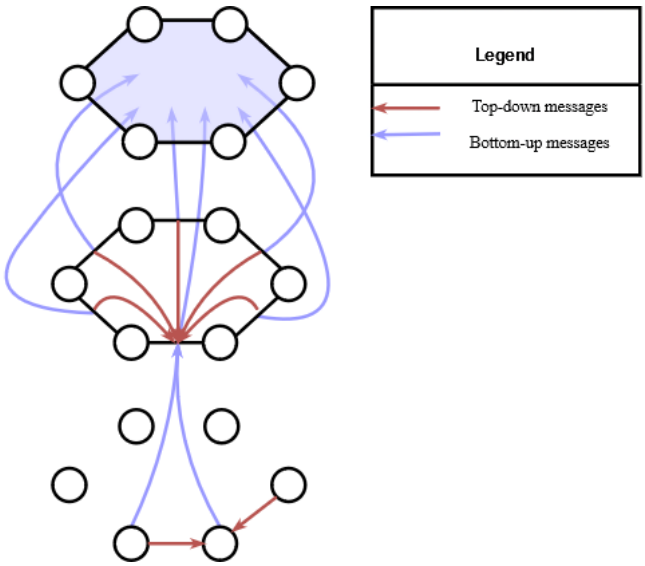


Figure 6.7: Message passing in an CW-net layer. There are two types of messages, the "top-down" messages are passed between the cells which are a part of the same higher-dimensional cell, while "bottom-up" messages pass information from lower to higher dimensional cells. In the original paper these messages were called "upper" and "boundary" messages respectively.

6.7 Subgraph Networks

Understanding the subghraph structure of networks can be an efficient way to bring important insight for determining the properties of such networks. In order to capture the subgraph patterns and enforce the interactions between these patterns, subgraph networks [40] (SGNs) were developed.

The simplest subgraph pattern is an edge, and the first-order SGN extracts these patterns from original graph as sets of two connected nodes. A new graph is constructed, where the extracted sets are new nodes, and edges between them are constructed if they share a common node in the original graph. The message passing is then done between these structures in the new graph.

This process can be iterated to the nth-order, where each next order processes the graph from the previous order. Second-order SGNs will induce open and closed triangles from the first-order SGNs, and so forth.

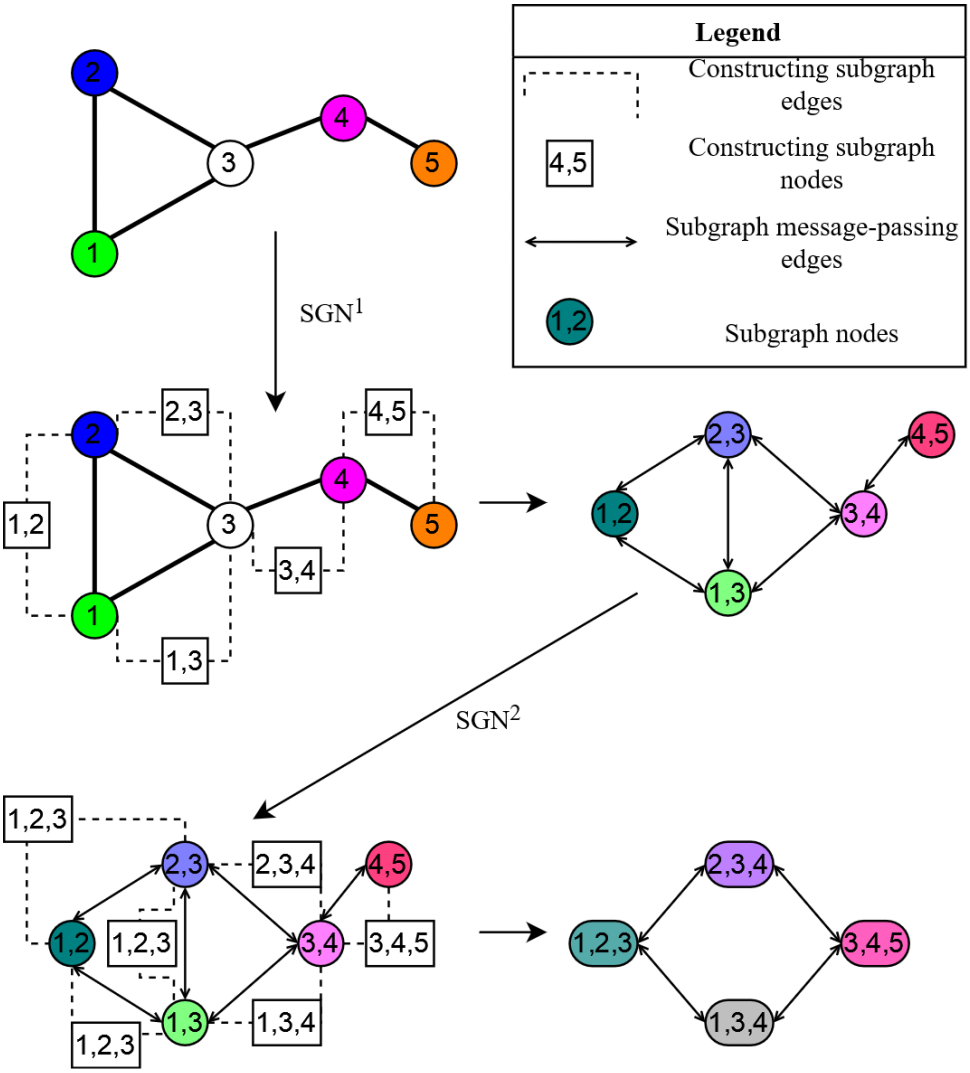


Figure 6.8: Message passing in an SGN layer. The process of constructing an first- and second-order SGN is shown. The first-order SGN is constructed by creating new nodes from edges in the original graph, and merging their node representations. Then a second-order SGN is constructed analogously from the first-order SGN.

Part III

Implementation

Chapter 7

NeuraLogic framework

The framework NeuraLogic [14] is used, which is written in Java, but has an API in Python [18]. Due to Python's ease of use, it is preferred for this purpose.

7.1 Syntax

The rule structure that was explained in Section 3.3 is the basis upon which the programs are built, and will be used throughout the examples in this thesis. However, the framework has its own syntax, which is based on this rule structure, adapted to use in Python.

Predicates are defined using the `Relation` module, and variables using `Var` module, which are abbreviated to `R` and `V` respectively. An example would be the following:

```
R.predicate_name(V.Var_name)
```

The naming conventions apply, meaning that variables should be capitalized, and predicate names and constants should be lowercase. The implication, instead of the Datalog symbol `:-`, is written as `<=`. The body of the rule is surrounded by parentheses, and weights are defined with braces after the predicate name. The dimension of the weights is given by a tuple

$(d_1, d_2, ..., d_n)$, where n is the number of dimensions and each d_i represents the size of ith dimension.

```
R.h(V.X)[1,] <= (R.b_1(V.X)[1,], R.b_2(V.X)[1,])
```

7.2 Computational graphs

The first step in the interpretation of the logical programs using this framework is to ground them and determine the Herbrand model, from which a computational graph is constructed in the following manner:

Logical construct	Node type
Ground fact	Fact node
Ground atom	Atom node
Ground rule's head	Aggregation node
Ground rule	Rule node

Table 7.1: Node types in the computational graph and their logical counterparts.

Example 1. An example of a one-layer GNN will be presented as defined in Section 6.1. The example graph has node features, and edges given by predicates `feature(X)` and `edge(X, Y)`. Then the propagation rules are defined as:

```
R.layer_1(V.X)[1,] <= (R.edge(V.Y, V.X), R.feature(V.Y)[1,])
R.layer_1(V.X)[1,] <= R.feature(V.X)[1,]
```

To connect the GNN layer with the output, another rule is defined:

```
R.predict <= R.layer_1(V.X)[1,]
```

The model template is presented in Figure 7.1 and the computational graph for a given example in Figure 7.2.

The ground template shows the relationships between the defined predicates. On the bottom layer, the two predicates are the ones defined in the examples.

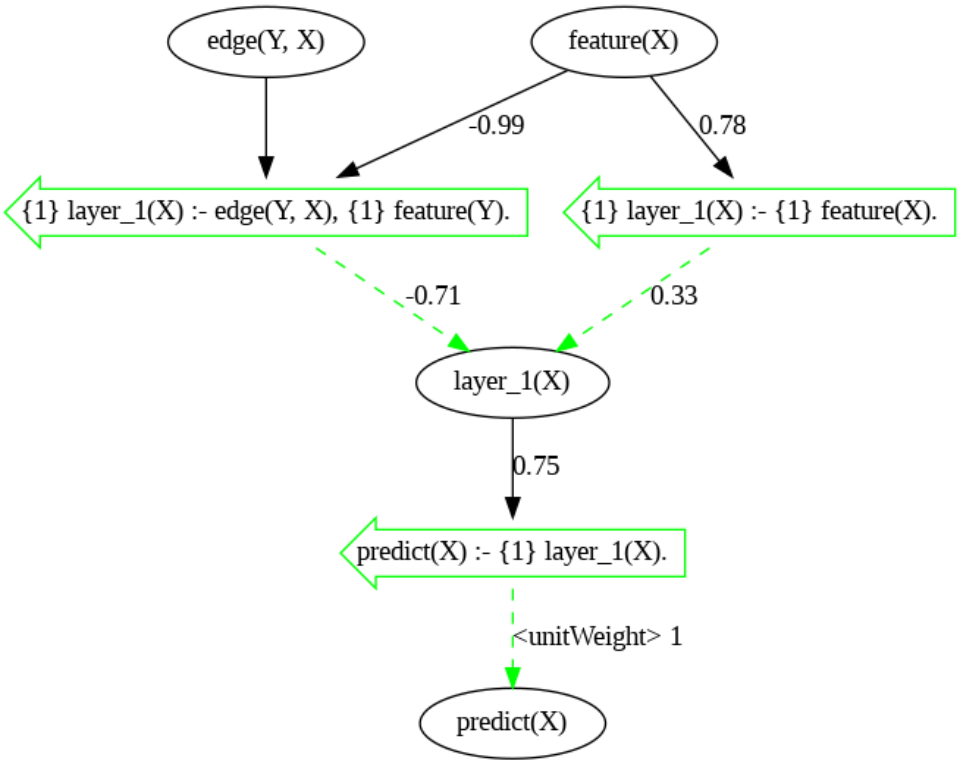


Figure 7.1: Template for the given model.

Then they are used by two rules defining the GNN propagation, which are further aggregated into the prediction. The numerical values that are seen between nodes are the weights, which are all scalars in this case.

The computational graph in Figure 7.2 is analogous to the ground template. It is, however, the representation of how the template is grounded on a specific example. Additionally, gradients, activation functions and other information can be seen in this graph.

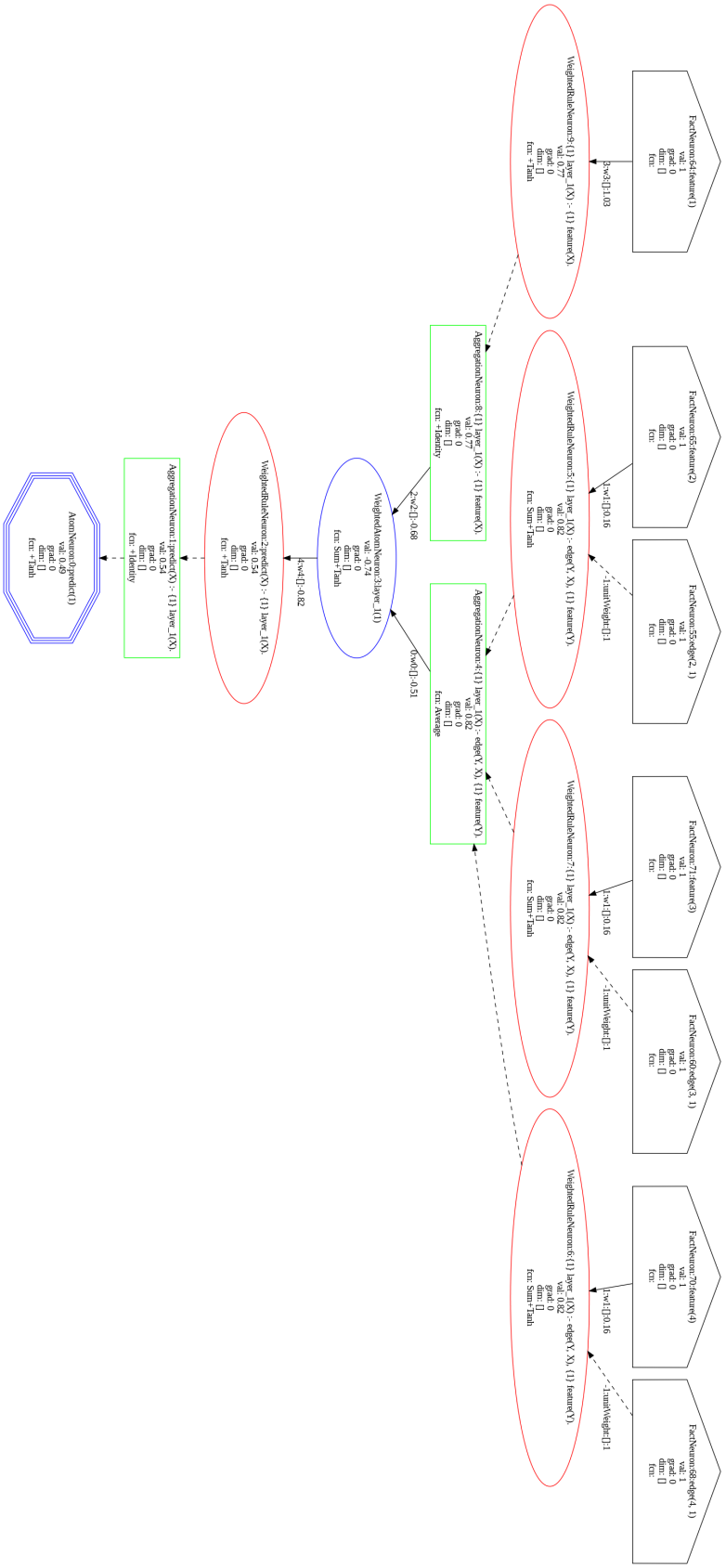


Figure 7.2: Computational graph for the given example. The node types as shown in Table 7.1 are visible.

Chapter 8

Dataset preprocessing

The datasets are assumed to be comprised of three main predicates, `node_embed/1` and `edge_embed/1`, which denote the feature embedding for nodes and edges, as well as `connection/3`, which is true when two nodes are connected via the edge given in the arguments. For example, if nodes `a1` and `a2` are connected by an edge `b`, the predicate `connection(a1, a2, b)` will be present.

For example, in the MUTAG dataset (Chapter 5), a typical entry is defined with predicates referring to the atom type by its chemical symbol (`c(a1)` for carbon, `h(a2)` for hydrogen, etc), and the predicate `bond(a1, a2, b1)` for connection. The predicates `b_1(b1)` define the bond type as single bond, and analogously `b_2` for double bond, etc. An ethylene molecule (Figure 8.1) in the MUTAG dataset will be encoded as follows:

```
c(c1), c(c2), h(h1), h(h2), h(h3), h(h4),
bond(c1, c2, b1), bond(c1, h1, b2), bond(c1, h2, b3),
bond(c2, h3, b4), bond(c2, h4, b5),
b_2(b2), b_1(b2), b_1(b3), b_1(b4), b_1(b5).
```

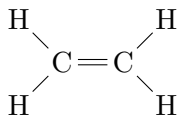


Figure 8.1: Ethylene molecule.

These requirements are assured in the preprocessing stage for each dataset, as well as extracting the node and edge types to their own predicates. Each

dataset has their own encoding for types of atoms and bonds, but they have to be unified. We can see an example below, where the left side are the predicates from a dataset, and on the right side the final unified representation:

```
atom_0(A1) ⇒ carbon(A1)
atom_5(A4) ⇒ oxygen(A4)

bond_0(A1, A4) ⇒ bond(A1, A4, B1), single_bond(B1)
...
```

The weighted predicates will be denoted with `W` before them, but each of these weights is different and unique for the predicate they belong to.

Model implementations can be found in Appendix B and chemical rules can be found in Appendix C. The entirety of the code used for these implementations will be available in the GitHub repository *DifferentiableChemistry* [11].

Chapter 9

Subgraph patterns

Aside from the models and chemical patterns that were implemented (please refer to Appendices B and C for the implementation details), there are some structural patterns or motifs, that can be deduced from molecules.

9.1 Basic patterns

Basic patterns are general for any graphs, and they include cycles, representing rings in molecules (Figure 9.1), and paths, representing chains of arbitrary length. Both patterns are defined with respect to a trainable parameter, which determines the maximum path length and maximum ring size. Respective indicator predicates are also defined, to determine if two nodes are in the same cycle or if there exists a path between two nodes, with given length constraints.

```
cycle(A, B) :- cycle(A, B, C).
cycle(A, B, C) :- connection(A, B, X1), connection(B, C, X2),
    connection(C, A, X3),
    W::node_embed(A), W::node_embed(B), W::node_embed(C),
    W::edge_embed(X1, W::edge_embed(X2), W::edge_embed(X3).
cycle(A, B, C) :- cycle(A, B, C, D).
...
```

```
path(A, B) :- path(A, B, max_depth).
path(A, B, 0) :- connection(A, B, X), edge_embed(X).
```

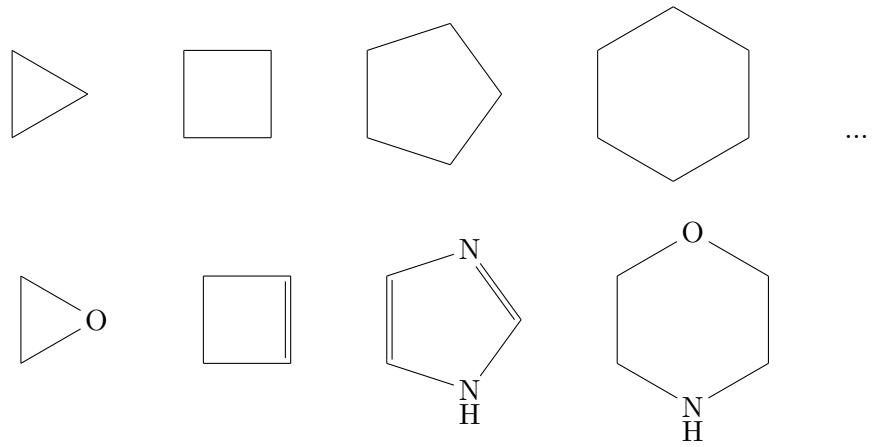


Figure 9.1: The most common ring sizes in organic chemistry are 3, 5, 6 atoms, but often up to 14 or more atoms. The examples in the bottom row are: ethylene oxide, cyclobutene, imidazole and morpholine. All of them, aside from cyclobutene, are heterocycles.

```
path(A, B, T) :- connection(A, C, X), edge_embed(X),
    path(C, B, T-1).
```

9.2 Y-shaped patterns

In molecules, there seems to be a significant recurring pattern of an atom connected to three neighbors in some kind of a Y shape (Figure 9.2). This is apparent in carbonyls, where the carbon atom is the center of this pattern, as well as imines, nitro compounds and others.

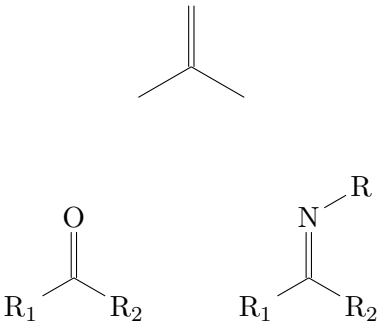


Figure 9.2: The proposed Y shape, as well as an example of carbonyls and imines.

The shape can further be abstracted to contain any type of bonds, not necessarily a double bond, which will then also include amines and other

similar structures.

```
y_subgraph(X1, X2, X3, X4) :-
    connection(X1, X2, B1), connection(X1, X3, B2),
    connection(X1, X4, B3),
    edge_embed(B1), edge_embed(B2),
    edge_embed(B3),
    node_embed(X1), node_embed(X2),
    node_embed(X3), node_embed(X4).

y_bond(X1, X2, X3, X4) :- y_subgraph(X1, X2, X3, X4),
    connection(X1, X2, B1), double_bond(B1).

y_group(X1, X2, X3) :- y_bond(Y1, Y2, X1, X2),
    y_bond(Z1, Z2, X2, X3).
```

9.3 Neighborhood patterns

The notion of neighborhoods is essential in GNNs, so a step further is taken in defining predicates which explicitly aggregate the messages. Carbon, as the main building block of organic compounds, usually has four neighbors, and to address this, we define a predicate. Another case is for atoms which usually have three neighbors, such as nitrogen.

```
four_nbhood(X) :-
    connection(X, X1, B1), connection(X, X2, B2),
    connection(X, X3, B3), connection(X, X4, B4),
    edge_embed(B1), edge_embed(B2),
    edge_embed(B3), edge_embed(B4),
    node_embed(X1), node_embed(X2),
    node_embed(X3), node_embed(X4),
    node_embed(X).


three_nbhood(X) :-
    connection(X, X1, B1), connection(X, X2, B2),
    connection(X, X3, B3),
    edge_embed(B1), edge_embed(B2), edge_embed(B3),
    node_embed(X1), node_embed(X2), node_embed(X3),
    node_embed(X).
```

Chirality is a concept in chemistry that refers to the property of a molecule

or a part of a molecule that cannot be superimposed on its mirror image. This means that it has a specific three-dimensional shape or orientation. In other words, it is not symmetric or identical to its mirror image. This is most commonly caused by the presence of one or more asymmetric carbon atoms (carbon atoms that are attached to four different groups), and these carbons are called chiral centers.

```
chiral_center(C) :- carbon(C),
    W::atom_type(X1), W::atom_type(X2),
    W::atom_type(X3), W::atom_type(X4),
    connection(C, X1, B1), connection(C, X2, B2),
    connection(C, X3, B3), connection(C, X4, B4),
    W::edge_embed(B1), W::edge_embed(B2),
    W::edge_embed(B3), W::edge_embed(B4),
    W::node_embed(X1), W::node_embed(X2),
    W::node_embed(X3), W::node_embed(X4).
```

An example would be a molecule called glyceraldehyde, where the hydroxyl group can either "pertrude" from the plane of the molecule, or "sink" into it, creating these non-superiposable mirror images, or enantiomers. This can be seen in Figure 9.3.

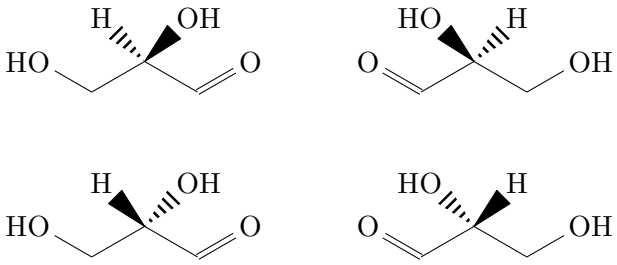


Figure 9.3: Glyceraldehyde molecule enantiomers on the left and their mirror images on the right.

9.4 Cyclic patterns

One interesting cyclic substructure is a four-member ring of alternating single and double bonds. The inspiration is taken from the molecule cyclobutadiene, see Figure 9.4, which is antiaromatic, despite sharing a similar bond arrangement as benzene.

```
brick(X) :- W::node_embed(Y1), W::edge_embed(B1),
    connection(X, Y1, B1), single_bond(B1),
    W::node_embed(Y2), W::edge_embed(B2),
    connection(Y1, Y2, B2), double_bond(B1),
```

```
    W::node_embed(Y3), W::edge_embed(B3),
    connection(Y2, Y3, B3), single_bond(B1),
    W::node_embed(X), W::edge_embed(B4),
    connection(Y3, X, B4), double_bond(B1).
```

Heterocycles, as cyclic patterns, are simply represented as a cycle where there exists a non-carbon atom.

```
heterocycle(X) :- W::cycle(X, C),
    carbon(C), ~carbon(X).
```

Figure 9.4: Cyclobenzadiene and the "brick" structure.

9.5 Collective patterns

The reason behind the naming of this group is that the patterns in this division are thought of as patterns which connect other patterns. The first example is the aliphatic chain, which serves as the scaffolding for any other pattern.

```
aliphatic_chain(X, Y) :- aliphatic_chain(X, Y, max_depth).

aliphatic_chain(X, Y, 0) :-connection(X, Z, B),
   carbon(X), carbon(Y),
   aliphatic_bond(B),
   W::edge_embed(B).


aliphatic_chain(X, Y, T) :- connection(X, Z, B),
    carbon(X),
    W::edge_embed(B),
    aliphatic_chain(Z, Y, T-1),
    W::aliphatic_bond(B).
```

The bridge pattern (Figure 9.5) is based on an atom which connects two different rings, neither of which it is a part of. A common example of these kinds of molecules is benzophenone, which serves as a building block for many other pharmaceutical compounds.

```
bridge(X) :-
    connection(X, Y, B1), connection(X, Z, B2),
    ~cycle(X, X1), ~cycle(Y, Z),
    W::cycle(Y, Y1),
    W::cycle(Z, Z1),
    W::edge_embed(B1), W::edge_embed(B2),
    W::node_embed(X).
```

Finally, there are compounds which have two rings which share one or more atoms.

```
shared_atom(X) :-
    connection(X, Y, B1), connection(X, Z, B2),
    W::cycle(X, Y), W::cycle(X, Z),
    ~cycle(Y, Z),
    W::edge_embed(B1), W::edge_embed(B2),
    W::node_embed(X).
```

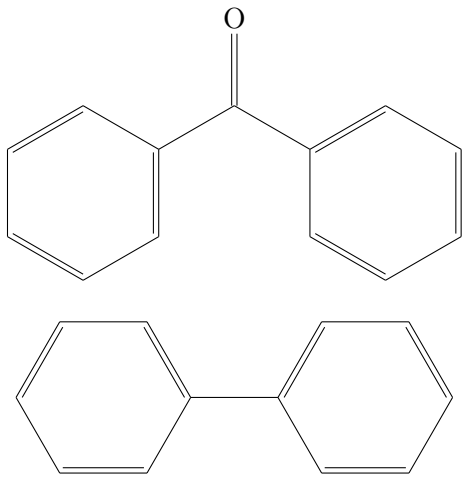


Figure 9.5: Benzophenone molecule with the bridging carbon atom and biphenyl, which has a shared atom between rings.

Part IV

Testing and results

Chapter 10

Testing pipeline

The testing was done separately for each dataset, such that the first run is a baseline for all models, without any chemical or subgraph rules added. This is to determine the best general hyperparameters, as well as the model-specific hyperparameters such as the number of layers, parameter size and depth of some models (meaning recurrence depth for diffusion GCNs, max ring size for CW networks and max order for SGNs). After the best values are found, the best combination of chemical and subgraph rules are found using these values.

Chapter 11

Results

The performance of each model on the MUTAG dataset was improved with the addition of some combination of chemical and subgraph rules, as opposed to the the models with optimized parameter, but no added rules. We can see this in Figure 11.1.

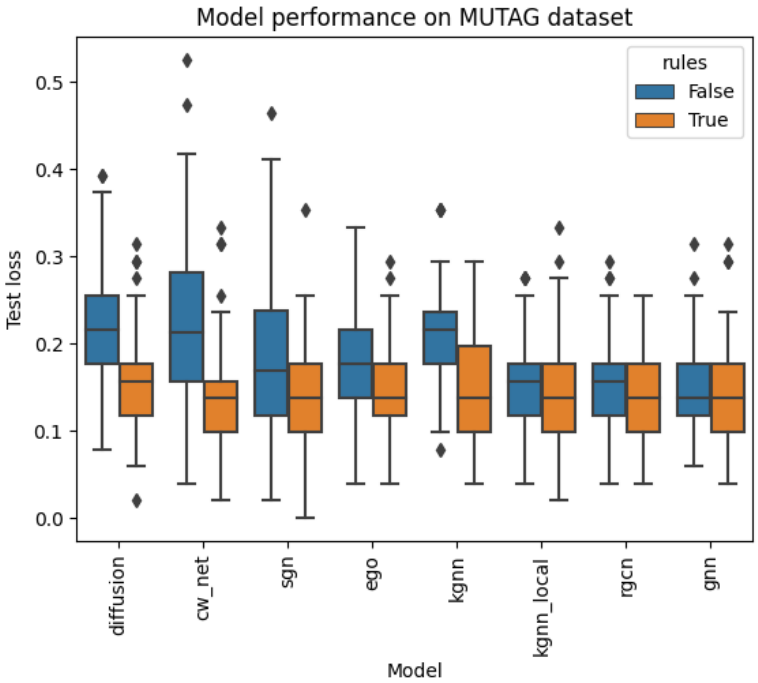


Figure 11.1: Benchmark performance across models with and without added chemical and subgraph rules on MUTAG dataset.

In the best performing models with added rules, the most common one seems to be global k-GNNs, followed by local k-GNNs, RGCNs and GNNs, which seems to be surprising, as these are the simplest of the implemented models. Probably because of the simplicity of the models, they were able to utilize the added rules the fullest. However, the distribution changes a bit when looking closer at the top circa 2% of the models, as seen in Figure 11.6. Here we can see that the best performing ones are actually local k-GNNs and SGNs, followed by GNNs and others.

The best performing rules are analogously extracted, and we can see that, on average, the chemical rules are more common than the subgraph rules (Figure 11.7). However there are two peaks at path and neighborhood subgraph rules. In the closer look we see that the most common rule set is the relaxations. Another point to make here is that the nitrogen rules are performing well because of the structure of the dataset, as it is comprised of primarily nitrogen-containing molecules. On the other hand, the sulfuric rules are not performing that well because sulfur is not common or present in most molecules of the dataset.

1	Hydrocarbons, nitrogen groups, relaxations, cycles, neighborhoods, circular patterns
2	Oxygen groups, nitrogen groups, sulfur groups, relaxations, cycles, collective patterns
3	Hydrocarbons, sulfur groups, relaxations, paths, neighborhoods, circular patterns, collective patterns
4	Hydrocarbons, oxygen groups, relaxations, paths, neighborhoods, circular patterns
5	Nitrogen groups, cycles
6	Hydrocarbons, oxygen groups, relaxations, paths, neighborhoods
7	Hydrocarbons, nitrogen groups, relaxations, cycles, paths, y-shapes, neighborhoods, circular patterns, collective patterns
8	Hydrocarbons, oxygen groups, relaxations, cycles, collective patterns
9	Hydrocarbons, oxygen groups, nitrogen groups, sulfur groups, relaxations, paths, neighborhoods, circular patterns, collective patterns
10	Oxygen groups, nitrogen groups, sulfur groups, relaxations, cycles, collective patterns

Table 11.1: Best performing rule combinations on MUTAG dataset.

Performance on the PTC MR dataset, however, doesn't seem to be enhanced by the addition of chemical and subgraph rules. The results are shown in Figure 11.2.

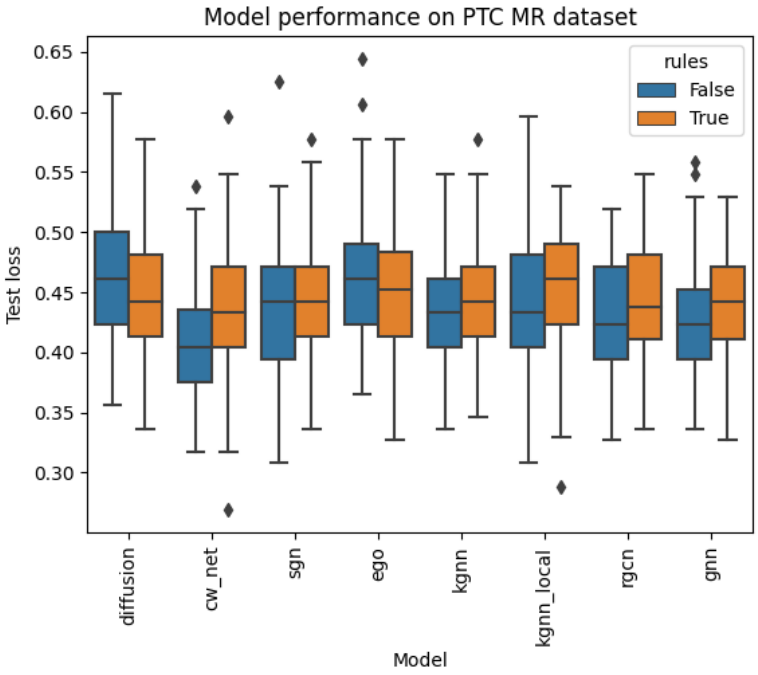


Figure 11.2: Benchmark performance across models with and without added chemical and subgraph rules on PTC MR dataset.

In the same manner, the best performing models and rules are extracted, and the distribution can be seen in Figures D.4 and D.5. The CW networks seem to perform quite well on this dataset, and this is probably due to the large presence of rings, since these models inherently take advantage of these structures. Standard GNNs and Diffusion GCNs are also very common in the best performing models, and this probably due to the fact that the neighborhoods of each node, both immediate and extended, hold enough important information for the models to extract meaning.

Meanwhile, we can see a different rule distribution in the best performing models, as opposed to the MUTAG dataset, where, for example, the Y-shaped subgraphs were not important in regards to performance, on this dataset they are one of the best performing. We can also see that the subgraph patterns, in general, seem to be much more present than chemical rules.

The PTC FR dataset shows similar results as PTC MR, where the rules do not improve the performance, as can be seen in Figure 11.3. Interestingly, the most successful models in this dataset are the global k-GNNs, appearing at least twice as often as other models. The superiority of this model deserves more attention in future work, to determine how can it be utilized, and maybe extrapolated to other datasets or models. As for the chemical and

subgraph rules, probably due to the structure of the dataset, the nitrogen and circular rules are the most often chosen ones, meaning that the presence of rings that contain functional groups based on nitrogen are very important in determining the target labels.

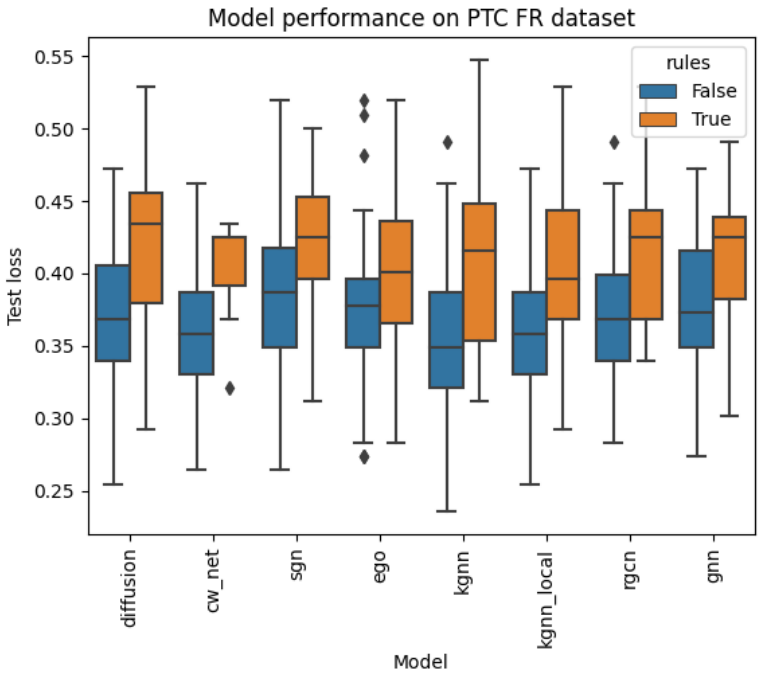


Figure 11.3: Benchmark performance across models with and without added chemical and subgraph rules on PTC FR dataset.

Again, for the PTC FM dataset, the performance is not being improved by the rules (Figure 11.4). There is an interesting superiority of the CWN model on this dataset, where it occupies almost half of the most successful models (Figure D.8). This suggests that the inherent ability of the model to recognize and pass messages in cycles is greatly beneficial to the performance. This should be investigated in future work. However, one would assume that cycles and circular patterns will be more present in the chosen rules, which is not the case for the latter, as seen in Figure D.9. Collective patterns also seem to play a great role in the accuracy of the models. This should probably indicate the presence of connected rings in the dataset, which affect the target label.

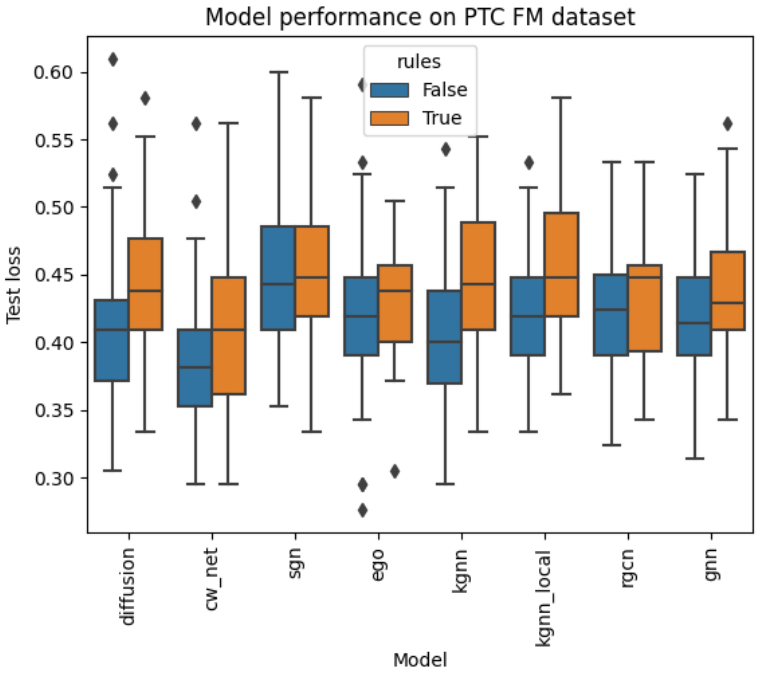


Figure 11.4: Benchmark performance across models with and without added chemical and subgraph rules on PTC FM dataset.

Finally, the performance on the PTC MM dataset is similar to the other PTC datasets, as seen in Figure 11.5. The most successful datasets seem to be RGCNs and GNNs (Figure D.10), which probably means that they can utilize the added rules better than other models, due to the simplicity. Also, according to the performance of these models, it may be assumed that the dataset holds more important information in the edges, rather than nodes, and RGCNs are able to make more use of that information, due to their ways of computation. The global higher-order GNNs are also successful in this dataset, and it may be the case that, as in PTC FM dataset, there are longer chains in this dateset, for which the shape plays a big role in its properties. This hypothesis seems to be backed by the fact that the most commonly chosen rules are hydrocarbon rules, as shown in Figure D.11. This set of rules are designed to recognize double and triple bonds in aliphatic chains, which occupy a larger region of space compared to a single bond, resulting in increased electron density between the bonded atoms. This increased electron density affects the repulsion between electron pairs, causing them to spread out more and influence the molecular shape.

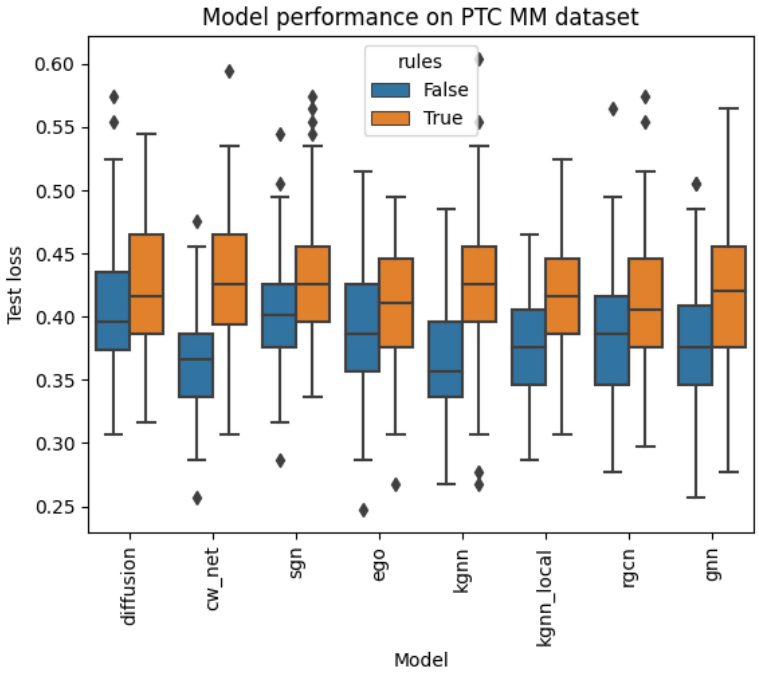


Figure 11.5: Benchmark performance across models with and without added chemical and subgraph rules on PTC MM dataset.

In the Table 11.2, we can see a breakdown of the test losses for each model and dataset.

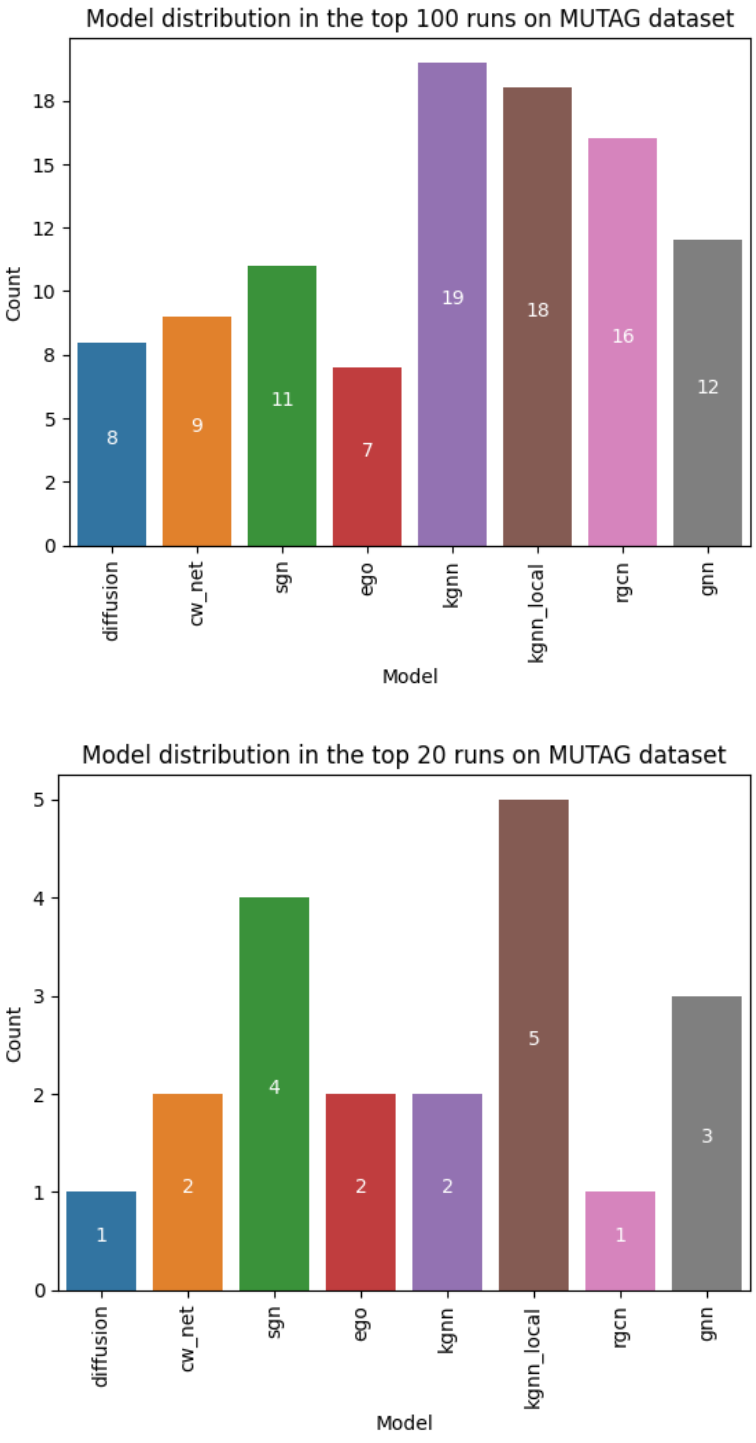


Figure 11.6: Highest performing models on MUTAG dataset.

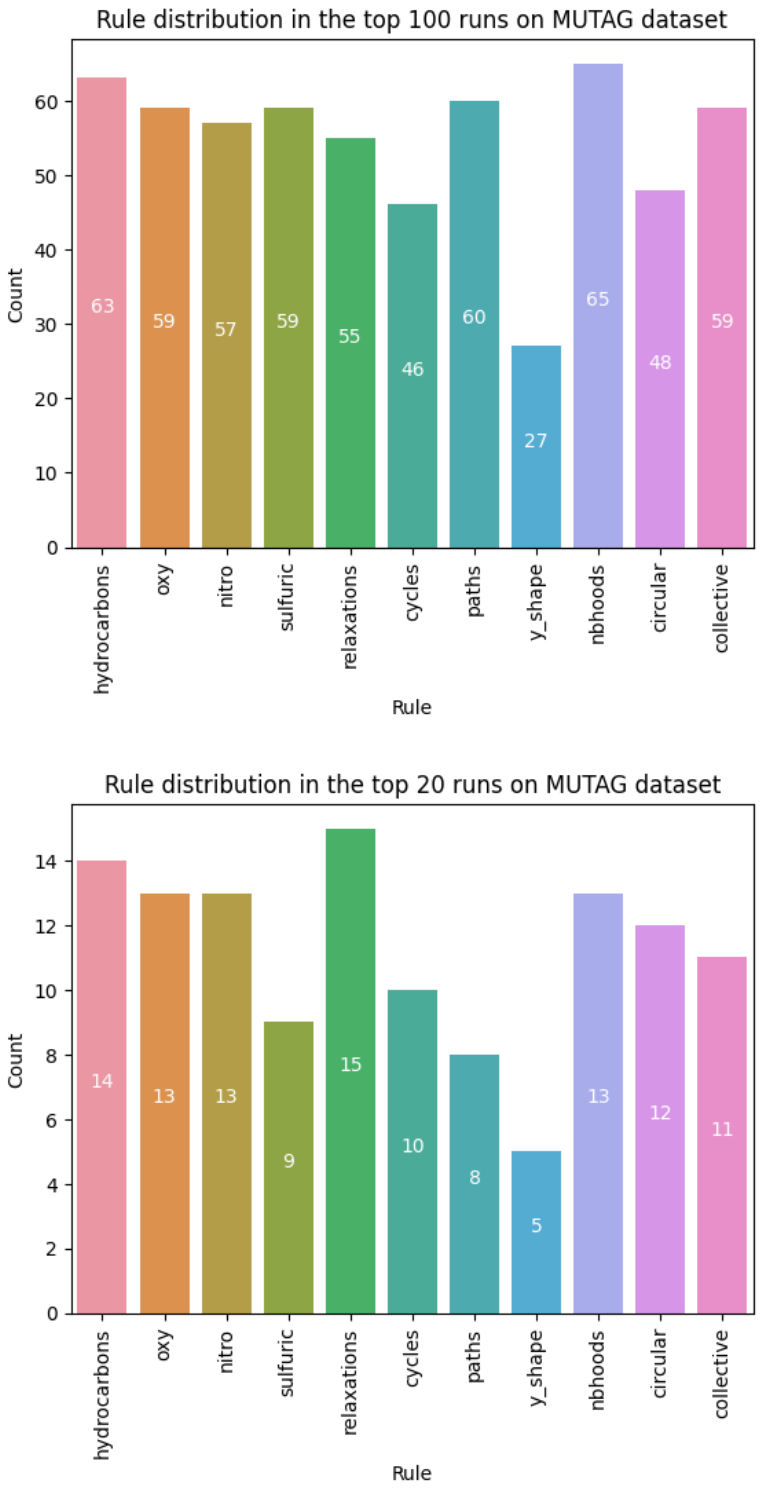


Figure 11.7: Highest performing rules on MUTAG dataset.

	Rules	MUTAG	PTC MR	PTC FR	PTC FM	PTC MM
GNN	No	$14.63 \pm 4.80\%$	$42.88 \pm 4.61\%$	$37.75 \pm 4.50\%$	$41.71 \pm 4.56\%$	$37.98 \pm 4.83\%$
	Yes	$14.19 \pm 5.24\%$	$44.11 \pm 4.66\%$	$41.21 \pm 4.72\%$	$43.57 \pm 4.66\%$	$41.70 \pm 5.26\%$
RGCN	No	$15.02 \pm 5.42\%$	$42.96 \pm 4.67\%$	$36.98 \pm 4.51\%$	$41.94 \pm 4.56\%$	$38.54 \pm 5.06\%$
	Yes	$13.92 \pm 4.74\%$	$44.26 \pm 4.69\%$	$41.58 \pm 5.51\%$	$43.03 \pm 4.84\%$	$41.09 \pm 5.32\%$
k-GNN	No	$20.92 \pm 5.23\%$	$43.69 \pm 4.45\%$	$35.34 \pm 5.23\%$	$39.96 \pm 4.96\%$	$36.30 \pm 4.15\%$
	Yes	$14.72 \pm 6.43\%$	$44.23 \pm 4.52\%$	$40.38 \pm 6.67\%$	$44.95 \pm 4.92\%$	$42.13 \pm 5.66\%$
k-GNN local	No	$14.98 \pm 5.07\%$	$43.91 \pm 5.59\%$	$36.02 \pm 4.43\%$	$41.72 \pm 4.43\%$	$37.88 \pm 3.96\%$
	Yes	$14.30 \pm 5.78\%$	$45.55 \pm 5.00\%$	$40.63 \pm 5.30\%$	$45.59 \pm 4.89\%$	$41.36 \pm 5.09\%$
ego-GNN	No	$17.94 \pm 5.31\%$	$46.32 \pm 5.16\%$	$37.31 \pm 4.38\%$	$42.22 \pm 5.08\%$	$38.75 \pm 5.16\%$
	Yes	$14.89 \pm 5.08\%$	$44.74 \pm 5.11\%$	$39.90 \pm 4.72\%$	$43.24 \pm 4.78\%$	$40.83 \pm 5.35\%$
DCCN	No	$21.48 \pm 6.44\%$	$46.39 \pm 5.27\%$	$37.26 \pm 4.75\%$	$40.91 \pm 5.35\%$	$40.22 \pm 5.09\%$
	Yes	$15.65 \pm 5.27\%$	$44.73 \pm 5.14\%$	$41.27 \pm 7.12\%$	$44.46 \pm 5.68\%$	$42.09 \pm 5.12\%$
CWN	No	$17.69 \pm 5.64\%$	$40.70 \pm 4.70\%$	$35.94 \pm 3.96\%$	$38.51 \pm 4.64\%$	$36.37 \pm 4.07\%$
	Yes	$14.18 \pm 5.37\%$	$43.46\% \pm 5.06\%$	$40.47 \pm 3.65\%$	$40.95 \pm 5.67\%$	$42.87 \pm 5.34\%$
SGN	No	$12.97 \pm 5.37\%$	$43.68 \pm 5.36\%$	$38.81 \pm 4.83\%$	$44.73 \pm 4.76\%$	$40.29 \pm 4.39\%$
	Yes	$13.67\% \pm 5.44\%$	$44.43 \pm 4.41\%$	$42.02 \pm 4.48\%$	$45.27 \pm 4.74\%$	$42.83 \pm 5.34\%$

Table 11.2: Table of test loss distribution details for every model on the datasets.

Chapter 12

Discussion

As already mentioned in Chapter 11, some models show severe superiority in some of the datasets. This is probably due to the inherent properties of the models, for example the ability of CW networks to recognize and pass messages between the nodes in cycles. This should also indicate a great presence of rings which affect the overall label of the molecule in the datasets where these models are most successful. In the PTC FR dataset, where the global k-GNNs were the most successful, there is probably some long-range dependencies which cannot be inferred by the other models, since they lack these "global" connections. This makes sense if the molecules are particularly large chains, such as in proteins, where the interactions of substructures which are far apart determine the shape of the molecule, which in turn determine its properties. Perhaps, testing this model on some datasets which deal with proteins (such as PPI [15]) can prove the hypothesis.

By the design of the testing pipeline, there was also no control over which rules were chosen, as this was given to a hyperparameter optimization framework to find the best combinations. Trying out all possible combinations would take too much time and computing power, as there is 11 different rules that can be chosen, resulting in $2^1 1$ combinations, and having in mind that one experiment could last up to 2 hours, it was infeasible to test all of them out. In the future, some time should definitely be dedicated to test these out on specific combinations of rules which are hand-chosen for each dataset, instead of blindly handing it out to a hyperparameter optimization framework.

Since MUTAG dataset is the simplest dataset out of the used ones, and also

it was the first one to explore, all the rules and models were based primarily on its structure. This probably posed an issue that the models, as well as rules, were overengineered to the design of this particular dataset, resulting in an inherent bias to the MUTAG dataset, explaining why the performance is so good, and also explaining at least a part of the decreased performance on the PTC MR dataset. There also might be a lack of understanding of the dataset, since there was no time to go into too much detail, while preparing the dataset to fit the required form. The differences between MUTAG and other datasets was apparent in how the connections are defined. While in MUTAG dataset, the connection predicate was a ternary `bond` predicate, the other datasets had a binary predicate which already included the type of bond (`bond_1` to denote a single bond, for example). These datasets then had to be preprocessed, as mentioned in Chapter 8, so that might have been an opportunity for some errors to arise.

Another reason for this might be that the optimized parameters for models without the added chemical or subgraph rules might be too strict and interfere negatively with the rules, when they are added. But the main issue seems to actually be overfitting, as we can see in Figure 12.1, which shows the severe difference between train loss and test loss on this dataset. The generalization error on the PTC FR, PTC FM and PTC MM datasets show similar characteristics (Figures D.1, D.2, D.3).

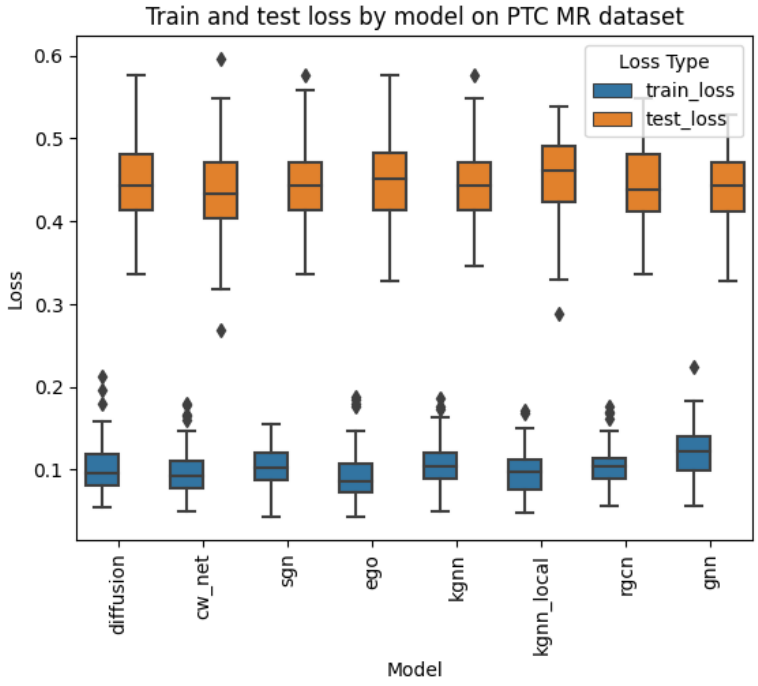


Figure 12.1: Train and test loss on the PTC MR dataset.

The templates overfits the MUTAG dataset as well, as seen in Figure 12.2,

but due to the simplicity of the dataset, it was actually beneficial to the overall accuracy of the models.

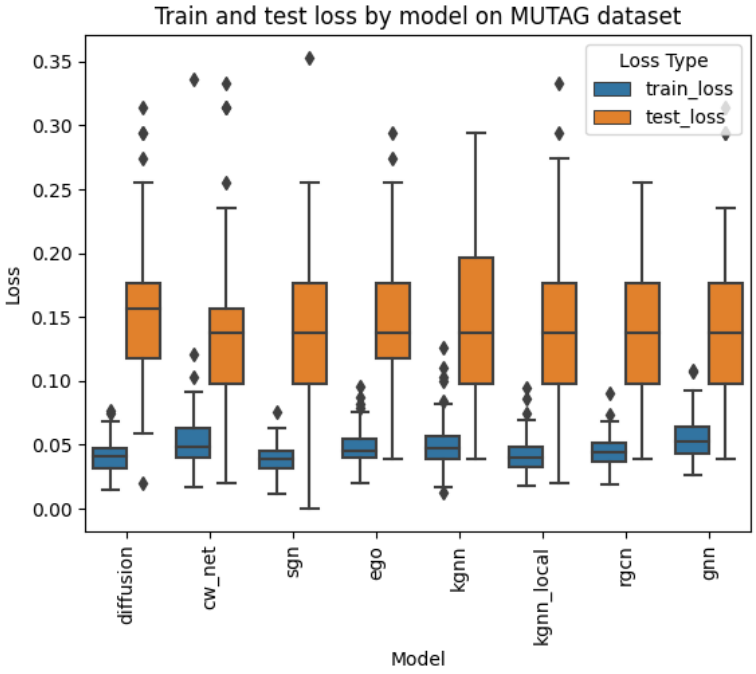


Figure 12.2: Train and test loss on the MUTAG dataset.

Due to the lack of time, the issues presented in this chapter have not been addressed, but it allows for an opportunity to investigate these points, and improve the generalization of the rules. Additionally, some other, more varied and bigger datasets can be used in future work to test the accuracy of this approach.

Chapter 13

Conclusion

The field of computational chemistry is undergoing a transformative phase with the emergence of specialized machine learning models designed to handle graph data. These new models hold great promise for advancing the accuracy and efficiency of machine learning models used for drug development, among other applications. As demonstrated in this thesis, the incorporation of a chemistry knowledge base has proven to be an effective approach for enhancing the performance of these models on some benchmark datasets. The results indicate that the inclusion of domain-specific rules has led to notable improvements across some presented models. These findings highlight the untapped potential for further advancements and innovations in this relatively unexplored area of research. Moving forward, continued exploration and refinement of such knowledge-based approaches have the potential to unlock new insights and propel the field of computational chemistry into even greater heights. While the initial results are not consistently excellent, there is significant optimism about the potential for improvement in this area.

In addition to the successful integration of a chemistry knowledge base into the presented machine learning models, further avenues for improving their accuracy and performance can be explored. One such direction involves the refinement and expansion of the existing rule set within the knowledge base. By incorporating additional chemical rules and constraints derived from expert knowledge and experimental data, the models can be fine-tuned to better capture the intricacies of chemical systems.

Furthermore, the utilization of larger and more diverse datasets can significantly enhance the generalizability and robustness of the models. By training

on a wider range of chemical structures, functional groups, and reaction types, the models can learn to recognize and predict a broader spectrum of chemical phenomena.

While the results of this thesis show some promise, it is important to acknowledge the limitations of the study. Due to time and computational constraints, the exploration of larger and more diverse datasets was restricted. Future research should aim to overcome these limitations by leveraging increased computational power and access to comprehensive chemical databases. Scaling up the models with extensive and diverse datasets will enable a more robust understanding of chemical systems and enhance the models' ability to generalize to new scenarios.

The collaboration between computational chemists, machine learning experts, and domain-specific chemists is crucial for advancing the field. The interdisciplinary nature of this research area necessitates the exchange of ideas, expertise, and resources between different communities. By fostering collaborations and knowledge-sharing, we can accelerate the progress and collectively push the boundaries of computational chemistry. The future holds great promise for the continued development and application of these models, enabling transformative advancements in drug discovery, materials design, and other areas of chemical research.

Appendices

Appendix A

Bibliography

[1] James Atwood and Don Towsley. Diffusion-convolutional neural networks, 2016.

[2] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montúfar, and Michael Bronstein. Weisfeiler and lehman go cellular: Cw networks, 2022.

[3] I. Bratko. *Prolog Programming for Artificial Intelligence*. International computer science series. Addison Wesley, 2001.

[4] Vollhardt K. Peter C. *Organic Chemistry: Structure and function*. W. H. Freeman, 2018.

[5] J. Clayden, N. Greeves, and S. Warren. *Organic Chemistry*. OUP Oxford, 2012.

[6] Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. Understanding convolutions on graphs, Feb 2022.

[7] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[8] Devendra Singh Dhami, Siwen Yan, and Sriraam Natarajan. A statistical relational approach to learning distance-based gcns, 2021.

[9] B. L. Douglas. The weisfeiler-lehman method and graph isomorphism testing, 2011.

[10] Varun Embar, Sriram Srinivasan, and Lise Getoor. A comparison of statistical relational learning and graph neural networks for aggregate graph queries. *Machine Learning*, 110(7):1847–1866, 2021.

[11] erhc. Differentiable chemistry. `https://github.com/erhc/DifferentiableChemistry`, 2023. Implementations of models and background chemical knowledge using the PyNeuraLogic framework.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[13] Garrett B. Goh, Nathan O. Hodas, and Abhinav Vishnu. Deep learning for computational chemistry, 2017.

[14] GustikS. Neuralogic: Deep relational learning through differentiable logic programming. `https://github.com/GustikS/NeuraLogic`, 2023. This is the official implementation of the Lifted Relational Neural Networks concept.

[15] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[16] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The Predictive Toxicology Challenge 2000–2001 . *Bioinformatics*, 17(1):107–108, 01 2001.

[17] Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, jun 2021.

[18] LukasZahradnik. Pyneuralogic. `https://github.com/LukasZahradnik/PyNeuraLogic`, 2023. PyNeuraLogic is a Python frontend for writing Differentiable Logic Programs.

[19] J.E. McMurry. *Organic Chemistry*. Cengage Learning, 2015.

[20] Christopher Morris, Matthias Fey, and Nils M. Kriege. The power of the weisfeiler-leman algorithm for machine learning with graphs, 2021.

[21] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2021.

[22] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, jan 2016.

[23] Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, and et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1), 2022.

[24] Claude Sammut and Geoffrey I. Webb. *Encyclopedia of Machine Learning*. Springer, 2010.

[25] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks, Sep 2021.

[26] Dylan Sandfelder, Priyesh Vijayan, and William L. Hamilton. Ego-GNNs: Exploiting ego structures in graph neural networks. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, jun 2021.

[27] Ryoma Sato. A survey on the expressive power of graph neural networks, 2020.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[29] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[30] R.M. Smullyan. *First-order Logic*. Dover books on advanced mathematics. Dover, 1995.

[31] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

[32] Gustav Šourek, Filip Železný, and Ondřej Kuželka. Beyond graph neural networks with lifted relational neural networks. *Machine Learning*, 110(7):1695–1738, jun 2021.

[33] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, Inc., USA, 1988.

[34] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, oct 1976.

[35] L.G. Wade. *Organic Chemistry*. New in Organic Chemistry Series. Prentice Hall PTR, 2011.

[36] Yuyang Wang, Zijie Li, and Amir Barati Farimani. Graph neural networks for molecules, 2023.

[37] Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao. *Graph Neural Networks: Foundations, Frontiers, and applications*. Springer, 2022.

[38] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[40] Qi Xuan, Jinhuan Wang, Minghao Zhao, Junkun Yuan, Chenbo Fu, Zhongyuan Ruan, and Guanrong Chen. Subgraph networks with application to structural feature space expansion. *IEEE Transactions on Knowledge and Data Engineering*, 33(6):2776–2789, jun 2021.

[41] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: A comprehensive review. *Computational Social Networks*, 6(1), 2019.

[42] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Appendix B

Model implementation

All models are defined such that the layer zero is simply the node embedding for each node, and all the layers take the output of the previous layer as node input. The output of the final layer is fed to the `predict` predicate.

B.1 Standard Graph Neural Networks

The GNN layer that was defined in Section 6.1 is implemented as follows:

```
gnn(X) :- W::node_embed(X), W::node_embed(Y),
    connection(X, Y, B), edge_embed(B).
```

Similarly to Example 1 in Section 7.2, this template will be grounded on a molecule from a dataset. Let's take a water molecule. The GNN layer computation for any of the hydrogens will be taking the node embeddings from itself and the oxygen atom, combining it with the edge embedding of the single bond by which it is bonded to the oxygen. For the oxygen atom, however, this will happen twice, once for each hydrogen atom. The results from the two computations will be combined to produce the final output for this node.

B.2 Relational Graph Convolutional Networks

This implementation is based RGCNs, defined in Section 6.2, with the exception that we have another weighed predicate defined, called `edge_types`, which aggregates the types of edges.

```
rgcn(X) :- W::node_embed(X), W::node_embed(Y),
    connection(X, Y, B), edge_embed(B),
    W::edge_types(B).
```

B.3 Higher-order Graph Neural Networks

Higher-order GNNs are defined with the distinction between the node embedding (or the output of the previous layer), and the output of the previous order. Global k-GNNs only aggregate the node embedding with the output of the $(k-1)$-GNN for each pair of nodes.

```
k_gnn(X) :- W::node_embed(X), W::previous_order(Y).
```

Local k-GNNs, on the other hand, aggregate the information only if the nodes are connected by an edge.

```
k_gnn(X) :- W::node_embed(X), W::previous_order(Y),
    connection(X, Y, B), W::edge_embed(B).
```

B.4 Ego Graph Neural Networks

For ego-GNNs (Section 6.4), we define a predicate which constructs an ego-graph of the node, and then we aggregate it to a layer based on the connectivity in the original graph.

```
ego_graph(X) :- connection(X, Y, B),
    W::edge_embed(B), W::node_embed(Y).

ego_gnn(X) :- connection(X, Y, B), W::ego_graph(Y).
```

B.5 Diffusion Convolutional Neural Networks

Diffusion operator (Section 6.5) is based on finding a path between two nodes. In this implementation, we are trying to find a path up to some maximum depth, aggregating information from all edges on the way.

```
path(X, Y, 0) :- W::edge_embed(B), connection(X, Y, B).
path(X, Y, T) :- W::edge_embed(B), W::path(Z, Y, T-1),
    connection(X, Z, B).

path(X, Y) :- path(X, Y, max_depth)
```

The path is then used as a condition for message passing between nodes, while the node's own embedding is reinforced.

```
diffusion(X) :- path(X, Y), W::node_embed(Y).
diffusion(X) :- W::node_embed(X).
```

B.6 Cellular Weisfeiler-Lehman Networks

In the definition of CW netorks from Section 6.6, the message passing is done in two phases, bottom-up and top-down.

```
cw(X) :- W::bottom_up_feaures(X).
cw(X) :- W::top_down_feaures(X).
```

In every layer, the induced node and edge features are saved and are used in the calculation of the output for the next layer.

```
node_features(X) :- W::node_embed(X).
node_features(X) :- W::cw(X).
node_features(X) :- W::previous_node_features(X).

edge_features(X) :- W::edge_embed(X).
edge_features(X) :- W::cw(X).
edge_features(X) :- W::previous_edge_features(X).
```

Bottom-up messages aggregate node to edge embeddings, and edge to cycle

embeddings. Cycle features are aggregated into a common predicate for each cycle with length up to a given maximum cycle size.

```
bond_features(B) :- connection(X, Y, B),
    W::previous_node_features(X),
    W::previous_node_features(Y).


cycle_features :- connection(X1, X2, B1),
    connection(X2, X3, B2), ...
    W::previous_edge_features(B1),
    W::previous_edge_features(B2), ...


bottom_up_feaures(B) :- W::bond_features(B).
bottom_up_feaures(X) :- W::cycle_features.
```

Top-down features are aggregated from cycles to edges and edges to nodes. The cycle messages are similarly constructed for all cycles up to the given maximum size, and they are aggregated for each edge. Edges which are a part of the same cycle will share messages, as well as the nodes which are a part of the same edge.

```
cycle_message(B1) :- connection(X1, X2, B1),
    connection(X2, X3, B2), ...
    W::previous_edge_features(B1),
    W::previous_edge_features(B2), ...

atom_nbhood(X) :- connection(X, Y, B),
    W::previous_node_features(Y), bond_nbhood(B).

bond_nbhood(B) :- W::cycle_message(B).

top_down_feaures(B) :- W::bond_nbhood(B).
top_down_feaures(A) :- W::atom_nbhood(A).
```

B.7 Subgraph Networks

Subgraph networks are defined in Section 6.7. The first-order SGN aggregates messages from three nodes and two edges connecting them. The higher-order SGNs simply aggregate the messages from the previous order.

```
sgn_1(B1, B2) :- connection(X, Y, B1),
    connection(Y, Z, B2),
    W::edge_embed(B1),
    W::edge_embed(B2),
    W::node_embed(X),
    W::node_embed(Y),
    W::node_embed(Z).

sgn_2(X, Z) :- W::sgn_1(X, Y), W::sgn_1(Y, Z).
sgn_3(X, Z) :- W::sgn_2(X, Y), W::sgn_2(Y, Z).
...
```

Appendix C

Chemical patterns

The chemical patterns were divided into several groups which include, aside from the ones already discussed, some general patterns as well as relaxations. All of the classes of patterns are grouped into `functional_group`, aside from relaxations, which have their own predicate `relaxed_functional_group`. The two are then aggregated into a predicate `chem_rules`, which is used to calculate the prediction.

C.1 General patterns

Certain chemical patterns or structures may not fit neatly into the predefined categories, or they are used to define more complex patterns.

To facilitate message passing, a predicate `bond_message` is defined, which aggregates atom and bond embeddings.

```
bond_message(X, Y, B) :- W::node_embed(X), W::node_embed(Y),
    W::edge_embed(B).
```

Furthermore, based on the types of bonds, four predicates are defined to include both the connection and the connection type. This will be useful to shorten the more complicated predicates.

```
single_bonded(X, Y, B) :- connection(X, Y, B), single_bond(B).
```

```
double_bonded(X, Y, B) :- connection(X, Y, B), double_bond(B).
triple_bonded(X, Y, B) :- connection(Y, X, B), triple_bond(B).
aromatic_bonded(X, Y, B) :- connection(X, Y, B),
    aromatic_bond(B).
```

An important feature of carbon atoms is saturation, which refers to the number of bonds that the carbon atom has. A saturated carbon atom is one that is connected to four other atoms, only by single bonds.

```
saturated(X) :- carbon(X),
    single_bonded(X, Y1), single_bonded(X, Y2),
    single_bonded(X, Y3), single_bonded(X, Y4).
```

Similarly, the halogen group is defined as any atom that is connected to a halogen atom, such as F, Cl, Br, or I.

```
halogen_group(Y) :- halogen(X), single_bonded(X, Y, B),
    bond_message(X, Y, B).
```

In the same manner, the hydroxyl (-OH) and carboxyl (carbon atom double bonded to oxygen) groups are defined.

```
hydroxyl(O) :- oxygen(O), hydrogen(H),
    single_bonded(O, H, B), bond_message(O, H, B).

carbonyl_group(C, O) :- carbon(C), oxygen(O),
    double_bonded(O, C, B), bond_message(O, C, B).

carbonyl_group(C, O, R1, R2) :- carbonyl_group(C, O),
    single_bonded(C, R1, B1), single_bonded(C, R2, B2),
    bond_message(C, R1, B1), bond_message(C, R2, B2).
```

C.2 Hydrocarbons

Benzene rings (Figure 4.2) are an important substructure in organic chemistry, consisting of six carbon atoms bonded aromatically. The rule is composed such that the messages between all atoms are passed. Additionally, an indicator rule is created to check if two atoms are in the same benzene ring.

```
benzene_ring(A, B) :- benzene_ring(A, B, C, D, E, F).

benzene_ring(A, B, C, D, E, F) :-
    aromatic_bonded(A, B, B1), aromatic_bonded(B, C, B2),
    aromatic_bonded(C, D, B3), aromatic_bonded(D, E, B4),
    aromatic_bonded(E, F, B5), aromatic_bonded(F, A, B6),
    carbon(A), carbon(B), carbon(C),
    carbon(D), carbon(E), carbon(F),
    bond_message(A, B, B1), bond_message(B, C, B2),
    bond_message(C, D, B3), bond_message(D, E, B4),
    bond_message(E, F, B5), bond_message(F, A, B6).
```

Another valuable substructure in organic chemistry is the presence of double and triple bonds between carbon atoms (Figure 4.1). These bonds play a crucial role in the reactivity and properties of organic compounds, as well as the shape in bigger molecules, such as proteins.

```
alkene_bond(C1, C2) :-  carbon(C1), carbon(C2),
    double_bonded(C1, C2, B), bond_message(C1, C2, B).
alkyne_bond(C1, C2) :- carbon(C1), carbon(C2),
    triple_bonded(C1, C2, B), bond_message(C1, C2, B).
```

C.3 Oxygen-containing compounds

Alcohols are implemented on the carbon atom connected to the hydroxyl group (Figure 4.3). One important note is that this carbon atom has to be saturated, lest it be confused with carboxylic acids.

```
alcoholic(C) :- saturated(C), hydroxyl(O),
    single_bonded(C, O, B1), bond_message(C, O, B1).
```

Ketones, aldehydes and acyl halides are defined on the central carbon of the carbonyl group, and they are distinguished depending on the substituents. See Figure 4.4.

```
ketone(C) :- carbonyl_group(C, O, R1, R2),
    carbon(R1), carbon(R2).

aldehyde(C) :- carbonyl_group(C, O, R, H),
    carbon(R), hydrogen(H).
```

```
acyl_halide(C) :- carbonyl_group(C, O, R, X),
    carbon(R), halogen(X).
```

Carboxylic acids are also defined on the central carbon of the carbonyl group, and has the hydroxyl group attached to itself. The central carbon atom in this case is not saturated, due to the double bond in the carbonyl group, and therefore cannot be confused with alcohols. Acid anhydrides are defined on the two central carbonyl carbon atoms. These structures can be seen in Figure 4.5

```
carboxylic_acid(C) :- carbonyl_group(C, O, R, O1),
    carbon(R), hydroxyl(O1).

carboxylic_acid_anhydride(C1, C2) :-
    carbonyl_group(C1, X1, O12, Y1),
    oxygen(O12), carbonyl_group(C2, X2, O12, Y1).
```

Similarly, ethers and esters are defined on the substituent carbons, see Figure 4.6.

```
ester(R1, R2) :- carbonyl_group(C, X, R1, O),
    carbon(R1), oxygen(O), carbon(R2),
    single_bonded(O, R2, B), bond_message(O, R2, B).

carbonate_ester(R1, R2) :- carbonyl_group(C, X, O1, O2),
    oxygen(O1), oxygen(O2),
    carbon(R1), single_bonded(R1, O1, B1),
    carbon(R2), single_bonded(R2, O2, B2),
    bond_message(O1, R1, B1), bond_message(O2, R2, B2).

ether(R1, R2) :- carbon(R1), oxygen(O), carbon(R2),
    single_bonded(R1, O, B1), single_bonded(O, R2, B2),
    bond_message(R1, O, B1), bond_message(R2, O, B2).
```

C.4 Nitrogen-containing compounds

We begin by defining the amino group, which is a nitrogen atom with three substituents, at least one of them is a carbon. This is used, in turn, to define quaternary ammonium ions, amines and amides. Carbamates are essentially a modified amide group. For reference, see Figure 4.7.

```
amino_group(N, R1, R2, R3) :- carbon(R1), nitrogen(N),
    single_bonded(N, R1, B1),
    single_bonded(N, R2, B2),
    single_bonded(N, R3, B3),
    bond_message(N, R1, B1),
    bond_message(N, R2, B2),
    bond_message(N, R3, B3).

quat_ammonion(N) :- nitrogen(N), carbon(C),
    amino_group(N, R1, R2, R3),
    single_bonded(N, C, B), bond_message(N, C, B).

amine(N) :- ~carbonyl(C, O), amino_group(N, C, R1, R2).

amide(R, R1, R2) :- carbonyl_group(C, O, R, N),
    amino_group(N, C, R1, R2).

carbamate(C1, C2, C3) :- oxygen(O), amide(O, C2, C3),
    single_bonded(O, C1, B), bond_message(O, C1, B).
```

Imines and imides are similarly defined for the substituents on the nitrogen and carbons from carbonyl groups (Figure 4.8).

```
imine(R, R1, R2) :- carbon(C), nitrogen(N),
    double_bonded(C, N, B), single_bonded(C, R1, B1),
    single_bonded(C, R2, B2), single_bonded(N, R, B3),
    bond_message(N, C, B), bond_message(C, R1, B1),
    bond_message(C, R2, B2), bond_message(N, R, B3).

imide(R, R1, R2) :- carbon(C1), nitrogen(N), carbon(C2),
    carbonyl_group(C1, O1, R1, N),
    carbonyl_group(C2, O2, R2, N),
    single_bonded(N, R, B), bond_message(N, R, B).
```

Azides, azo compounds and cyanates are defined on the terminal carbons. Compare to the structure in the Figure 4.9.

```
azide(C) :- carbon(C1), nitrogen(N1),
    nitrogen(N2), nitrogen(N3),
    single_bonded(C, N1, B1),
    double_bonded(N1, N2, B2), double_bonded(N2, N3, B3),
    bond_message(C, N, B1),
    bond_message(N1, N2, B2), bond_message(N2, N3, B3).
```

```
azo(C1, C2) :- carbon(C1), nitrogen(N1),
    nitrogen(N2), carbon(C2),
    single_bonded(C, N1, B1), single_bonded(N2, C2, B3),
    double_bonded(N1, N2, B2),
    bond_message(C, N, B1), bond_message(N2, C2, B3),
    bond_message(N1, N2, B2).

cyanate(R) :- carbon(C), nitrogen(N), oxygen(O), carbon(R),
    triple_bonded(C, N, B1),
    single_bonded(C, O, B2), single_bonded(O, R, B3),
    bond_message(C, N, B1),
    bond_message(C, O, B2), bond_message(O, R, B3).

isocyanate(R) :- carbon(C), nitrogen(N), oxygen(O), carbon(R),
    double_bonded(C, N, B1), double_bonded(C, O, B2),
    single_bonded(N, R, B3),
    bond_message(C, N, B1), bond_message(C, O, B2),
    bond_message(N, R, B3).
```

Nitro compounds and nitrates (Figure 4.10) are based off of the nitro group, which is a nitrogen atom connected to two oxygens, and a substituent, upon which the distinction is made.

```
nitro_group(R) :- nitrogen(N), oxygen(O1), oxygen(O2),
    single_bonded(R, N, B1), single_bonded(N, O2, B3),
    double_bonded(N, O1, B2),
    bond_message(R, N, B1), bond_message(N, O2, B3),
    bond_message(N, O1, B2).

nitro(C) :- carbon(C), nitro_group(C).

nitrate(C) :- carbon(C), oxygen(O), nitro_group(O),
    single_bonded(C, O, B), bond_message(C, O, B).
```

Azidrines are heterocyclic groups and they are defined on the carbon atoms, which can be seen on the Figure 4.11.

```
aziridine(C1, C2) :- carbon(C1), carbon(C1),
    nitrogen(N), hydrogen(H),
    single_bonded(C1, C2, B1), single_bonded(N, H, B4),
    single_bonded(N, C1, B2), single_bonded(N, C2, B3),
    bond_message(C1, C2, B1), bond_message(N, H, B4),
    bond_message(N, C1, B2), bond_message(N, C2, B3).
```

C.5 Sulfur-containing compounds

Similar to the previous implementations, thiocyanates (Figure 4.12) and thiols (Figure 4.13) are defined on the substituent carbon atom.

```
thiocyanate(R) :- carbon(C), sulfur(S), oxygen(O), carbon(R),
    triple_bonded(C, S, B1),
    single_bonded(C, O, B2), single_bonded(O, R, B3),
    bond_message(C, S, B1),
    bond_message(C, O, B2), bond_message(O, R, B3).

isothiocyanate(R) :- carbon(C), sulfur(S),
    oxygen(O), carbon(R),
    double_bonded(C, S, B1), double_bonded(C, O, B2),
    single_bonded(S, R, B3),
    bond_message(C, S, B1), bond_message(C, O, B2),
    bond_message(S, R, B3).

thiol(C) :- carbon(C), sulfur(S), hydrogen(H),
    single_bonded(C, S, B1), single_bonded(S, H, B2),
    bond_message(C, S, B1), bond_message(S, H, B2).
```

Sulfides and disulfides are defined on the terminal carbon atoms, see Figure 4.13 for reference.

```
sulfide(C1, C2) :- carbon(C1), sulfur(S), carbon(C2),
    single_bonded(C1, S, B1), single_bonded(S, C2, B2),
    bond_message(C1, S, B1), bond_message(S, C2, B2).

disulfide(C1, C2) :- carbon(C1), sulfur(S1),
    sulfur(S2), carbon(C2),
    single_bonded(C1, S1, B1),
    single_bonded(S2, S2, B12), single_bonded(S2, C2, B2),
    bond_message(C1, S1, B1),
    bond_message(S2, C2, B2), bond_message(S1, S2, B12).
```

C.6 Relaxations

So far we have been dealing with strictly defined chemical structures, however, sometimes, the strict rules governing chemical structures can be limiting.

This is where the concept of relaxations comes in.

The relaxations of the chemical rules are first done on the basis of types of atoms and types of bonds. New predicates are defined, such that they encompass aliphatic types of bonds (single, double triple bonds), and aromatic types, as well as some notion of key atoms, which should include oxygen, nitrogen and other commonly occuring atom types.

From this, two connection predicates are defined, relaxed aliphatic and aromatic bond, with weighed bond messages.

```
relaxed_aliphatic_bonded(X, Y, B) :- connection(X, Y, B),
    aliphatic_bond(B), W::bond_message(X, Y, B).

relaxed_aromatic_bonded(X, Y, B) :- connection(X, Y, B),
    aromatic_bond(B), W::bond_message(X, Y, B).
```

Carbonyl group can be relaxed, such that the rule triggers for any combination of connected key atoms, while preserving the substituents.

```
relaxed_carbonyl_group(X, Y) :- W::key_atom(X), W::key_atom(Y),
    relaxed_aliphatic_bonded(X, Y).

relaxed_carbonyl_group(C, O, R1, R2) :-
    relaxed_carbonyl_group(C, O),
    relaxed_aliphatic_bonded(C, R1),
    relaxed_aliphatic_bonded(C, R2).
```

Similarly, a relaxed benzene ring would be any six atoms bonded in an aromatic ring. This representation would also include some common heterocyclic compounds, along with other hexacycles.

```
relaxed_benzene_ring(A) :- relaxed_benzene_ring(A, B).
relaxed_benzene_ring(A, B) :-
    relaxed_benzene_ring(A, B, C, D, E, F).
relaxed_benzene_ring(A, B, C, D, E, F) :-
    relaxed_aromatic_bonded(A, B),
    relaxed_aromatic_bonded(B, C),
    relaxed_aromatic_bonded(C, D),
    relaxed_aromatic_bonded(D, E),
    relaxed_aromatic_bonded(E, F),
    relaxed_aromatic_bonded(F, A).
```

The arrangement of any carbon which connected to a non-carbon has a potential to be a functional group, so a very broad relaxed predicate is defined.

```
potential_group(C) :- W::relaxed_aliphatic_bonded(C, X),
    W::noncarbon(X), carbon(C).
```

Appendix D

Result details

Figures and tables which are referenced in Chapter 11 and Chapter 12 will be presented here.

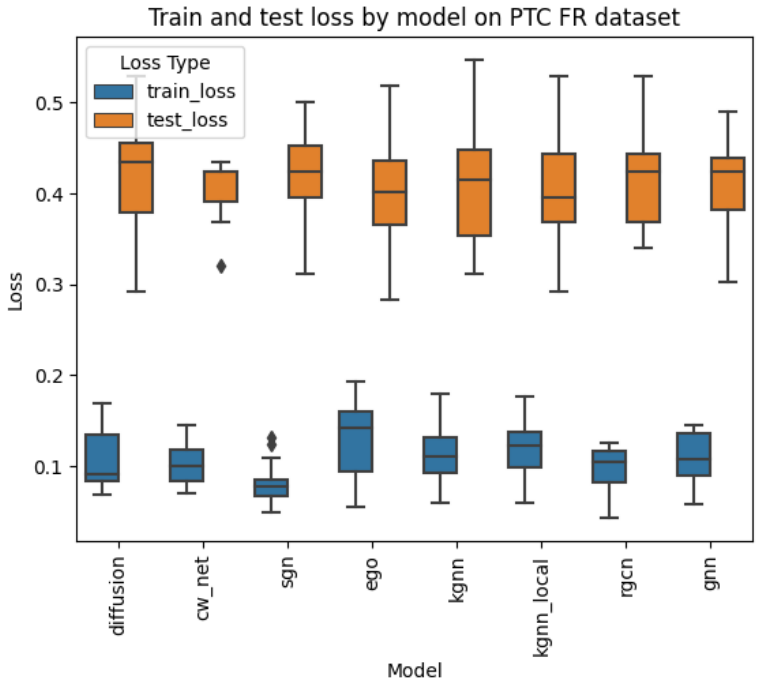


Figure D.1: Train and test loss on the PTC FR dataset.

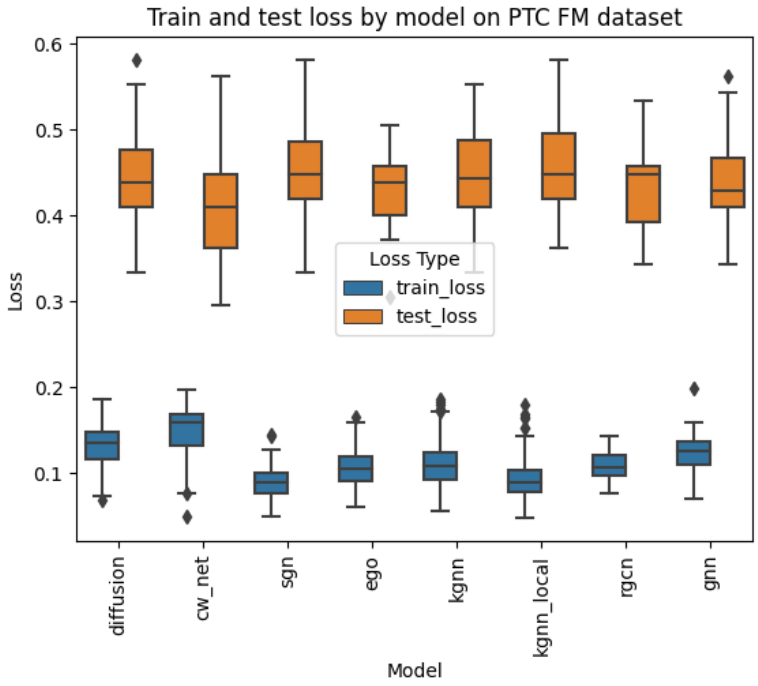


Figure D.2: Train and test loss on the PTC FM dataset.

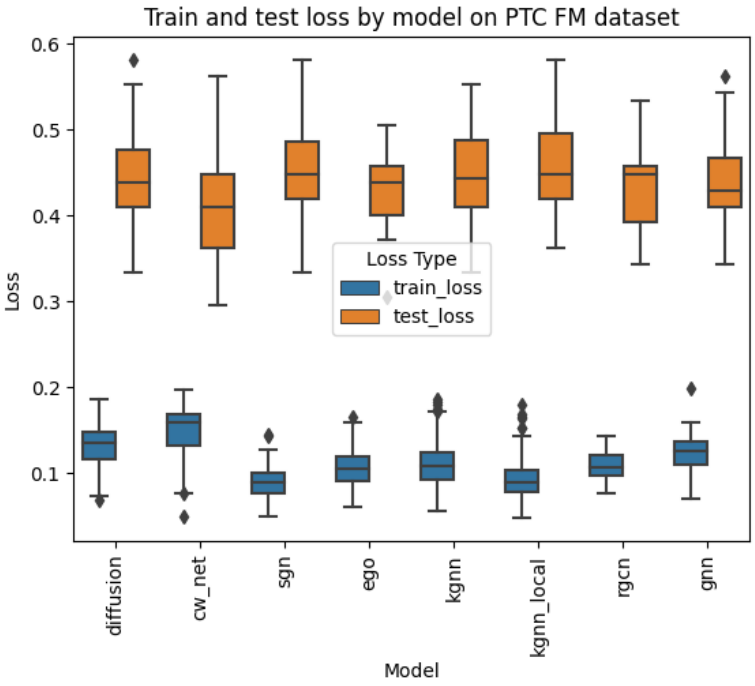


Figure D.3: Train and test loss on the PTC MM dataset.

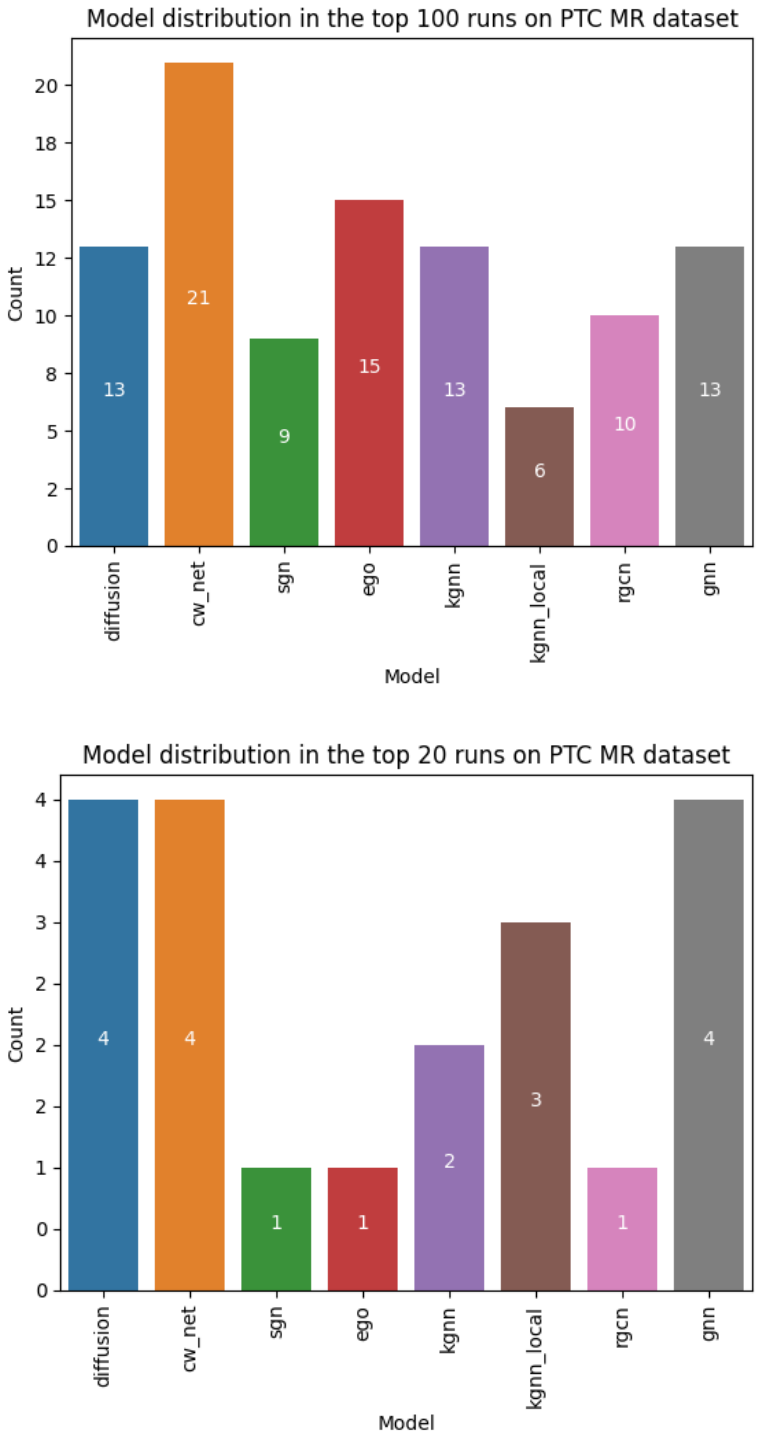


Figure D.4: Highest performing models on PTC MR dataset.

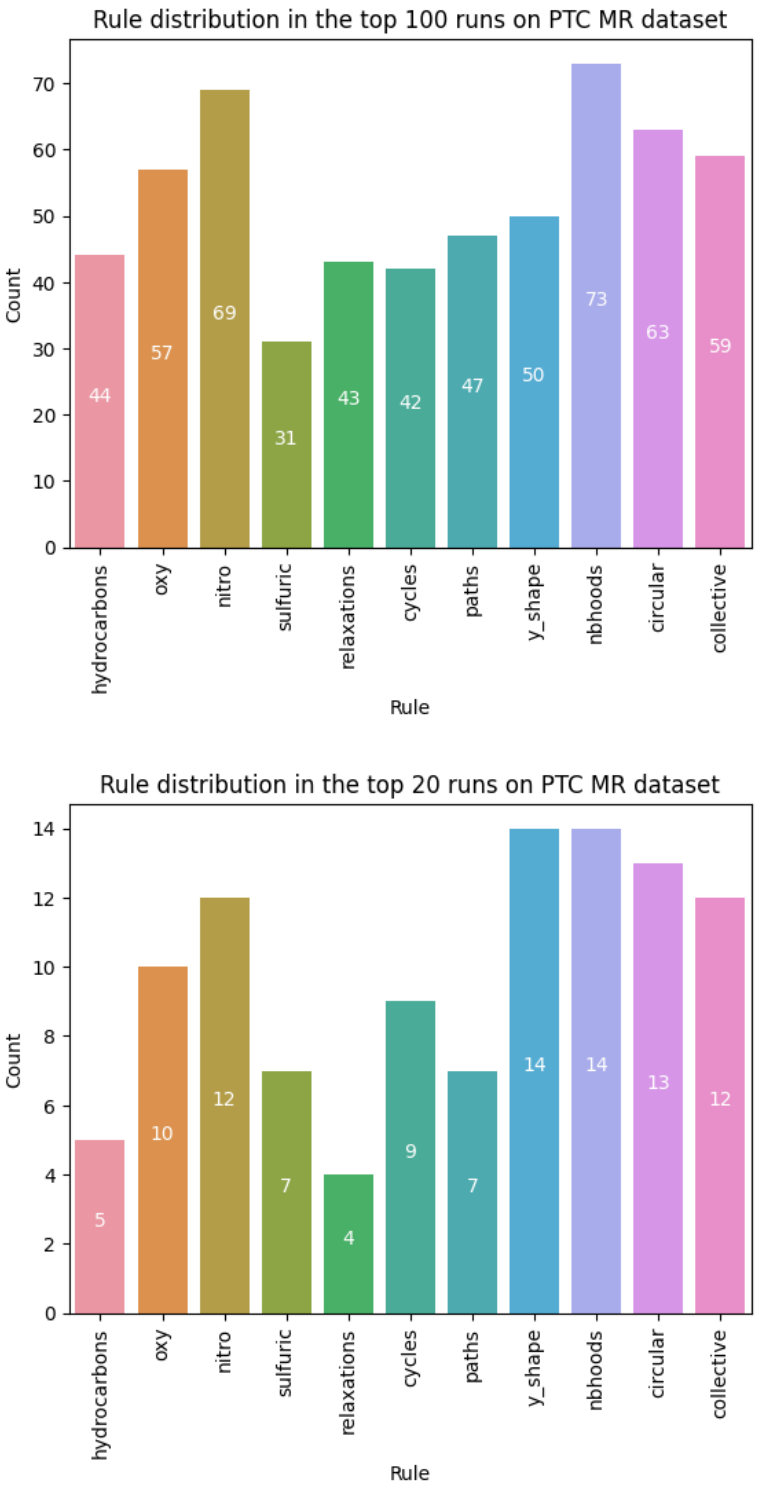


Figure D.5: Highest performing rules on PTC MR dataset.

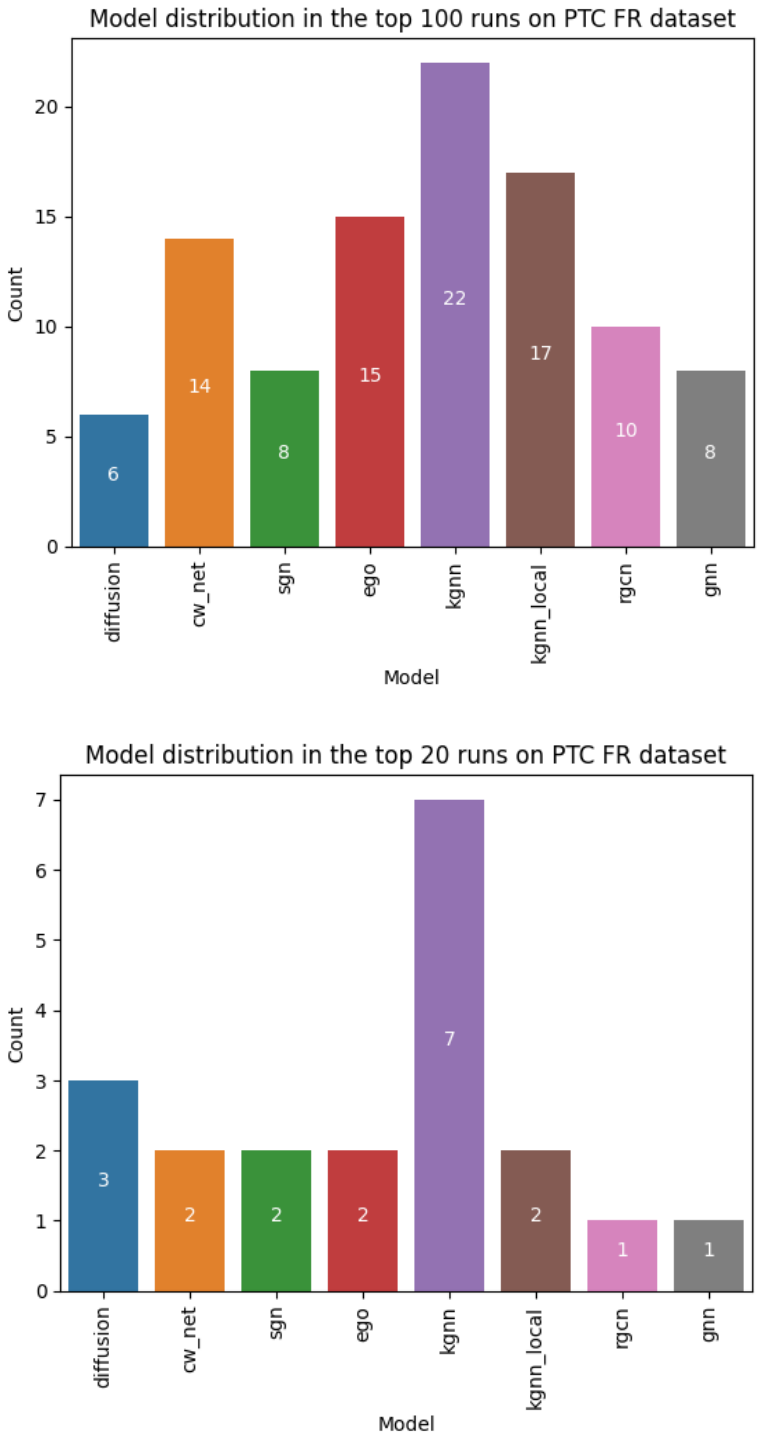


Figure D.6: Highest performing models on PTC FR dataset.

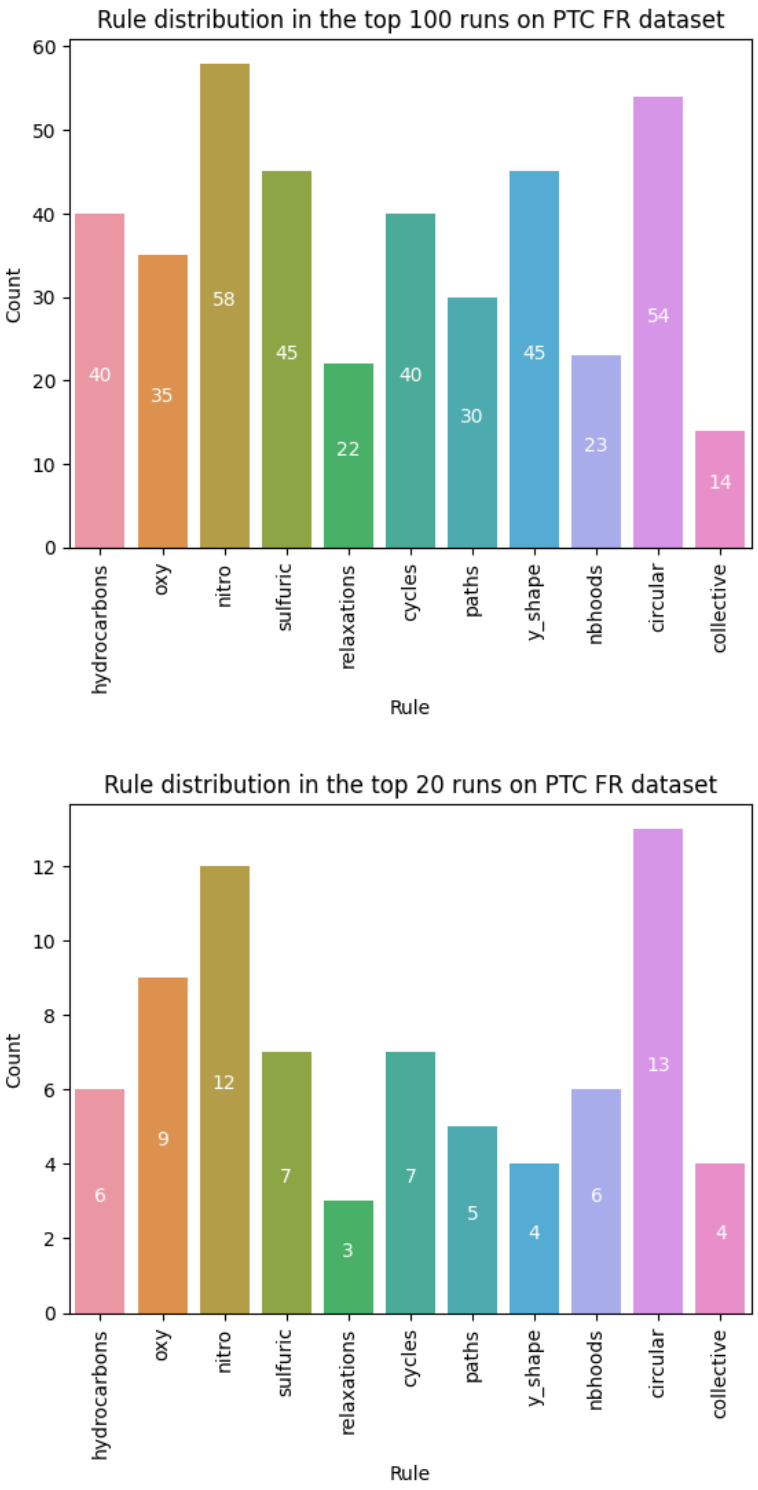


Figure D.7: Highest performing rules on PTC FR dataset.

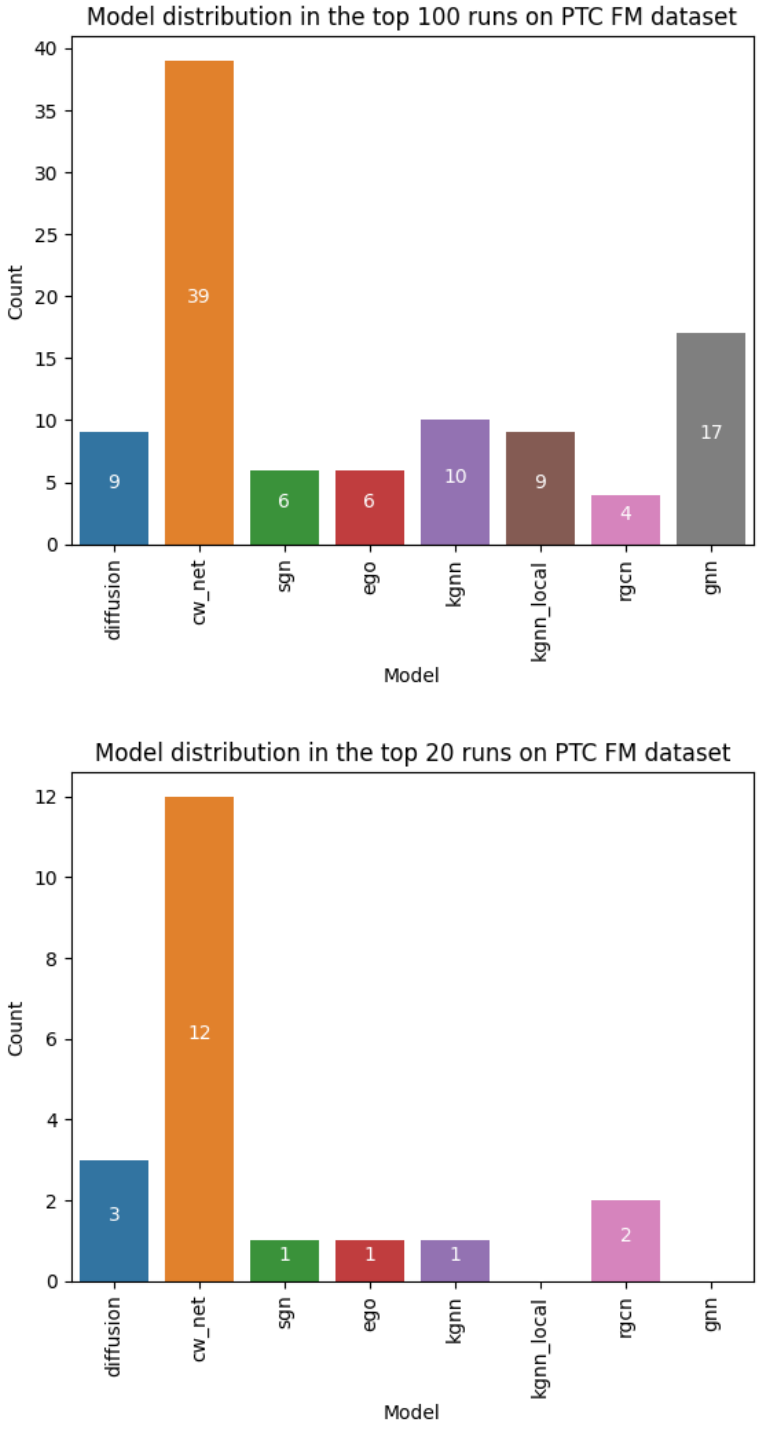


Figure D.8: Highest performing models on PTC FM dataset.

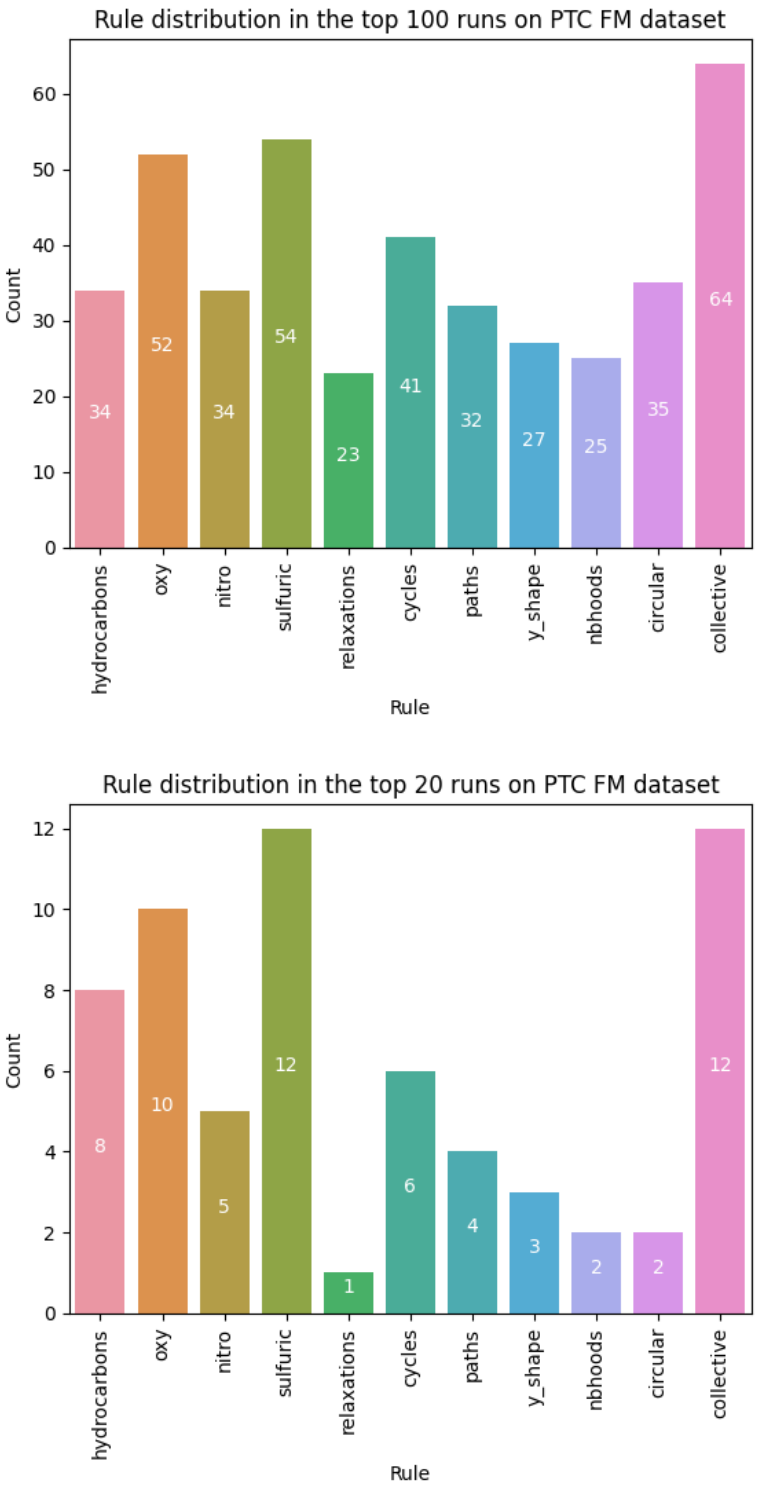


Figure D.9: Highest performing rules on PTC FM dataset.

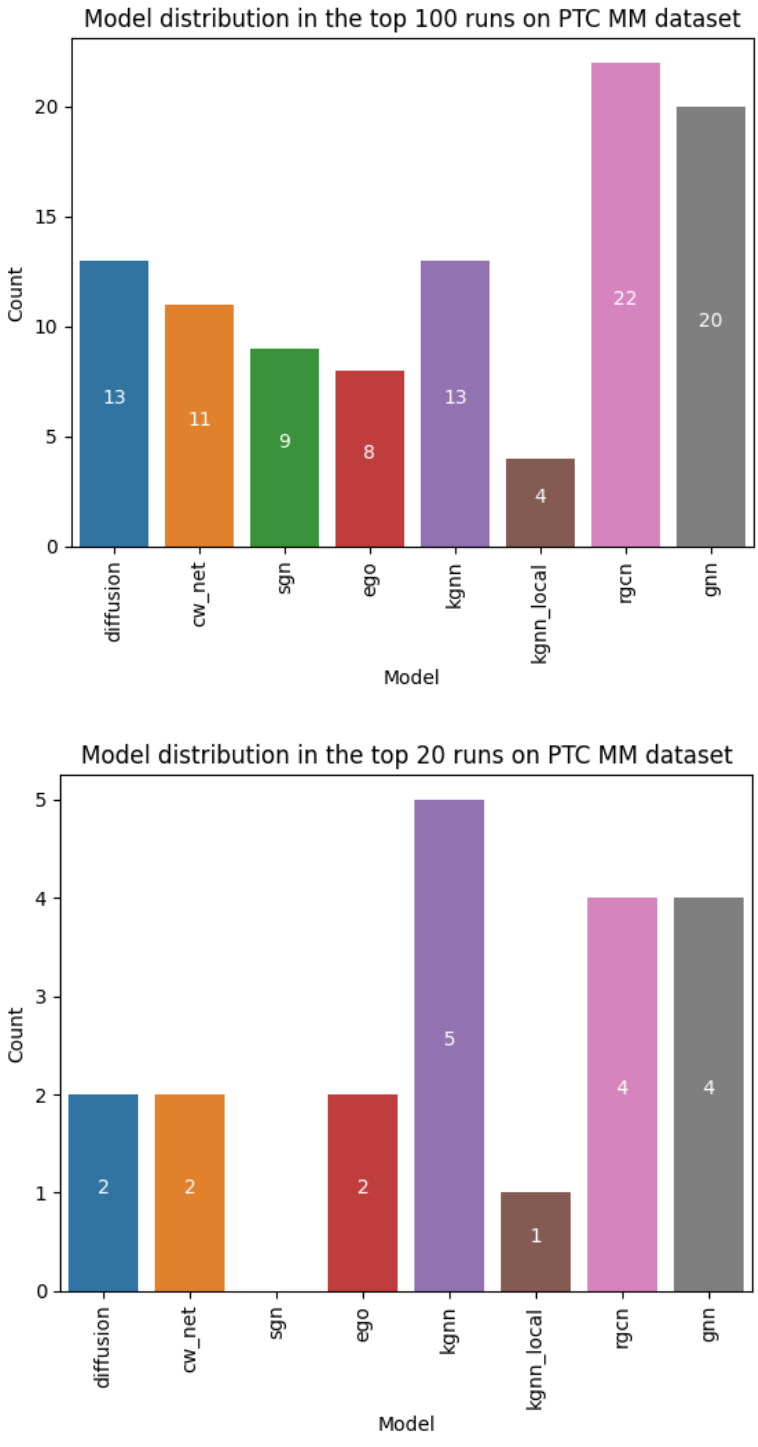


Figure D.10: Highest performing models on PTC MM dataset.

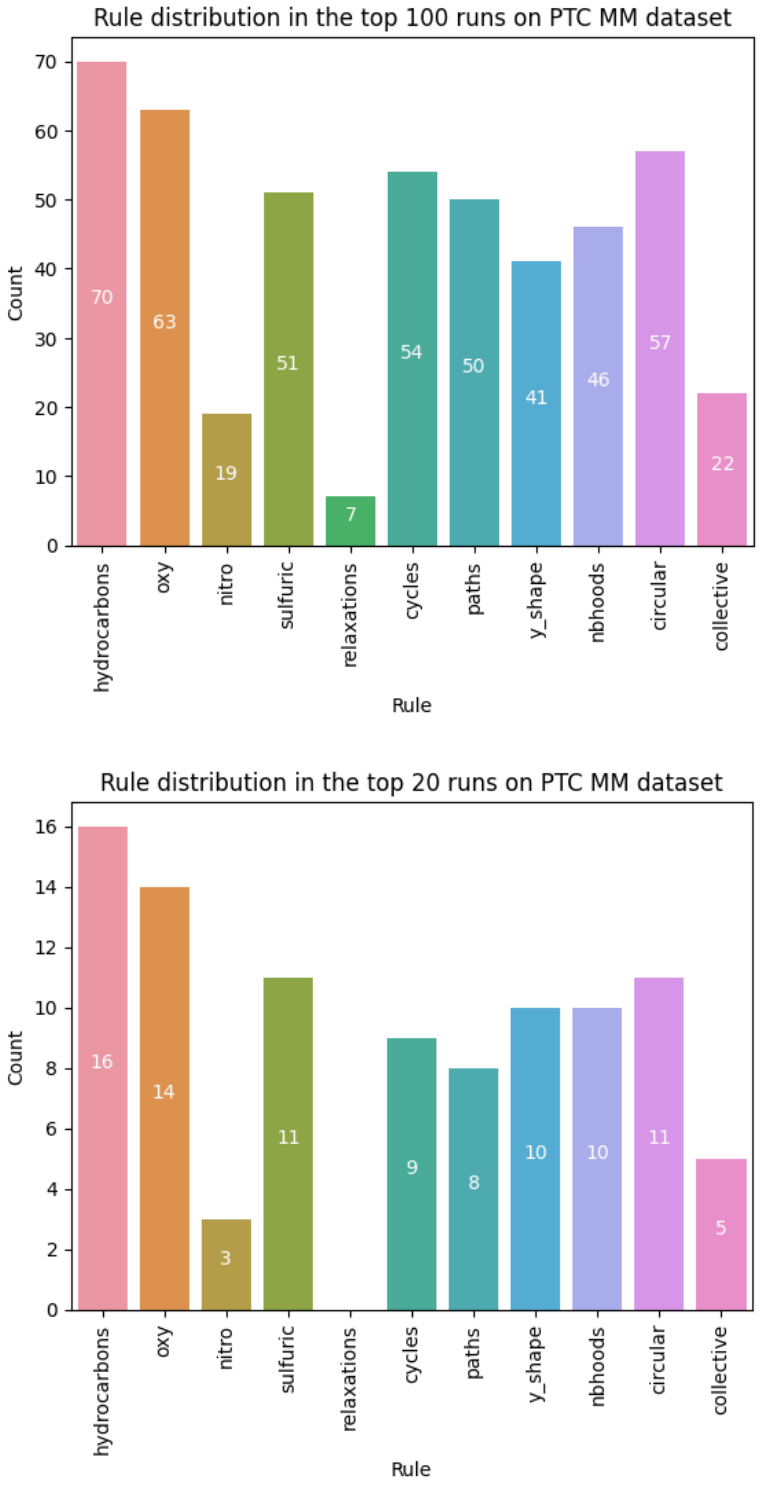


Figure D.11: Highest performing rules on PTC MM dataset.