Universiteit Leiden

Master Computer Science

AI-Driven Synthetic Route Length Prediction – Benchmarking Graph Neural Networks Against Traditional Machine Learning in *de novo* Drug Discovery

Name:	Pengxu Zheng
Student ID:	s2917211
Date:	06/04/2025
Specialisation:	Advanced Computing & Systems
1st supervisor:	Dr. Preuss, M.
2nd supervisor:	MSc. Hassen, A.K.

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Abstract

Computer-Aided Synthesis Planning (CASP) has revolutionized retrosynthetic analysis through computational approaches, yet challenges remain in high computation cost and synthetic feasibility. This study investigates machine learning for predicting synthetic route lengths directly from molecular structures, circumventing time-intensive CASP search methods. We implement graph neural networks (GNN) and XGBoost models trained on *1k*, *10k*, and *50k* bioactive molecules from the CASPYrus dataset, with comprehensive evaluation on *200k* **ChEMBL** test molecules. Our results demonstrate that GNN-based multiclass classification achieves 62.8% accuracy and 0.625 $F1$ *Score* in generalizability testing, significantly outperforming regression approaches (best R^2=0.29). The *50k* GNN model shows particular effectiveness in predicting shorter routes ($1-4$ steps) that are most relevant to chemists, with hyperparameter optimization confirming parameter robustness. While XGBoost exhibited overfitting tendencies, GNNs proved superior at capturing structural relationships in molecular graphs. This work establishes the feasibility of machine learning for synthetic complexity prediction, providing a foundation for future applications in *de novo* drug design and synthetic route optimization.

KEYWORDS: computer-aided synthetic planning, retrosynthesis, GNN, cheminformatics, high-performance computing, synthetic route scoring, *de novo* drug discovery

Contents

Chapter 1

Introduction

1.1 Subject overview

Retrosynthesis has long been the core of the *de novo* drug discovery process [51]. With each new type of drug developed, hundreds of millions of lives can be saved. Thus, empowering such processes with new technologies to push the frontier of that subject matter has significant value in unfolding. With computer-aided synthetic planning (CASP), high-performance computing, and predictive modeling based on machine learning with an available chemical knowledge base, retrosynthesis as a traditional chemical process is nowadays being revolutionized. Corey and Wipke envisioned such a bright future of computer-assisted synthetic planning as early as 1969 [10], followed by a 50-year-long endless discovery. From the earliest vacuum tube computers that solved hard-coded chemical reactions to modern scientists utilizing deep neural networks and symbolic AI to plan, evaluate, and optimize synthetic routes, CASP as an active research area has achieved remarkable progress [52, 59, 36]. Yet, with reaction templates representing the collection of known chemical properties and reactions of available chemical compounds, the computation and time cost of such calculations sometimes still impose limitations on the use of digital synthetic plans in laboratory practices. On top of that, synthetic routes, as computed by mainstream CASP software, are not always optimal, cost-effective, or environmentally friendly. Ertl and Schuffenhauer's seminal work in 2009 inaugurated the field of synthetic accessibility scoring by establishing a robust quantitative framework for evaluating the synthetic feasibility of drug-like molecules, thereby setting the stage for subsequent advances in synthetic accessibility scoring [14]. In recent years, researchers have begun to deep dive into the feasibility and scoring of synthetic routes solved by CASP software, such as *Retrosynthetic accessibility score* (*RAscore*) by Thakkar et al. in 2021 and the synthetic route scoring framework in the most recent release of the CASP tool *AiZynthFinder 4.0* by Saigiridharan et al. in 2024 [59, 48]. Following such achievements, this project aims to explore a different approach to speed up the *de novo* drug discovery process. By estimating synthetic route lengths using AI on molecular structures of target molecules, we discover the possibility of more efficient synthetic planning and synthetic routes scoring that offers feasible chemical insights in laboratory settings, which further enables use cases for CASP-powered chemical laboratory practices.

1.2 Experiment overview

To explore the feasibility of predicting synthetical properties using machine learning on only chemical structures, the experiment is designed to carry out large-scale data generation, processing, model training, and evaluation in a modern, academic high-performance supercomputing cluster [34]. As there were limited ref-

erence materials on such novel endeavors, it is yet not clear which specific machine learning paradigms tend to achieve the most convincing results. Considering the scope of this master's thesis project, we decided to structure the experiments on supervised learning methods of multiclass classification and regression, as illustrated in detail. Two machine learning algorithms, *Graph Neural Networks* (**GNN**) and *eXtreme Gradient Boosting* (**XGBoost**) are used to train machine learning models. Data sets containing synthetic route properties, including route lengths, are generated by open-source CASP software **AiZynthFinder** from a curated bioactive molecular data set jointly developed by *Amsterdam University Medical Center* and *Leiden University* [18, 6]. Size partitions of the training data are applied to explore the scaling effect on the models' performance metrics [1]. Generalizability evaluations are carried out on all machine learning models trained on the aforementioned data sets. Finally, hyperparameter optimization experiments are performed to verify the robustness of our chosen and default training hyperparameters.

[1] We've adopted different training and evaluation objectives for the multiclass classification and regression experiments as detailed in Chapter 4.4.

Chapter 2

Backgrounds and related work

2.1 Machine Learning

Machine learning, as first introduced as a terminology by Arthur Lee Samuel in 1959 [49], is now an active field in artificial intelligence that focuses on developing systems capable of approximating patterns from input and generate anticipatory outcomes. Similar to a mathematical function $\vec{y} = f(\vec{x})$, the foundation of machine learning revolves around the functional relatiionship between an independent variable $\vec{x}$ and a dependent variable $\vec{y}$. In other words, a dependent variable $\vec{y}$'s value can be approximated with a trained machine learning model based on the value of the independent variable $\vec{x}$. With such capability of function approximation, machine learning has now become a key source of automation in many research or industrial scenarios where repetitive statistical estimation is needed. From image identification to natural language processing, from self-driving vehicles to predicting the outcome of chemical synthesis, machine learning finds a broad spectrum of use cases.

2.2 Machine learning paradigms

Consists of many paradigms, machine learning tasks can be carried out via various methods characterizing each of their unique structures and objectives. Machine learning paradigms are the foundation of modern data analysis and predictive modeling, where each paradigm and the respective algorithm exhibit distinct properties, enabling them target different scenarios and use cases.

2.2.1 Supervised learning

A fundamental method known as "supervised learning" entails training an algorithm on a labeled dataset, associating each training example with an output label. After training, the model learns a mapping $\vec{x} \mapsto \vec{y}$ from inputs to outputs based on its learning objective function and generalize this mapping to new, unseen data. Such paradigm of labeling-train-predict enables supervised learning to replace repetitive manual labor in numerous unique scenarios, from financial risk estimation to medical imaging scans [57, 1].

Classification vs. Regression

Classification and regression fall into the category of supervised learning. Classification involves the prediction of input data into pre-defined groups or classes. For example, the widely used image recognition technology is capable of identifying cats and dogs from a collection of unordered pictures [33]. Similarly, email spam filters can prevent messages with a high likelihood of being malicious from entering our inbox [20].

Regression focuses on predicting continuous numerical outcomes from input data rather than classifying observations into discrete categories. In a regression task, the model learns a mapping that estimates the value of a continuous target variable based on input features, where the target variable can take any value within a range. For instance, in financial forecasting, a common regression task is the prediction of daily stock returns. Because the target variable of financial return is inherently continuous, regression is the natural modeling framework. In this scenario, regression models utilize historical prices, trading volumes, and various market indicators ($\vec{x}$) to forecast the continuous change in stock values for the next day ($\vec{y}$) [21].

Both methods have their unique pros and cons in a given machine learning task, which is why task formulation at the beginning of any machine learning project should be considered first. As this project focuses primarily on supervised learning methods, further details of task formulation and the comparison between classification and regression methods will be discussed in detail in Chapter 4.1.

2.2.2 Unsupervised learning

Unsupervised learning is a machine learning paradigm that discovers inherent patterns in unlabeled data through self-organization rather than predefined labels [63]. Primary examples of unsupervised learning include clustering for intrinsic grouping and dimensionality reduction for feature simplification. Clustering methods partition data based on similarity metrics, exemplified by *K-means* which minimizes intra-cluster variance through iterative centroid updates [38]. Density-based approaches like DBSCAN (Density-Based Spatial Clustering of Applications with Noise) overcome spherical cluster assumptions by identifying arbitrary-shaped groups through neighborhood connectivity analysis [15]. Dimensionality reduction preserves essential information while compressing features, as achieved by *Principal Component Analysis* (PCA) through orthogonal transformation to linearly uncorrelated components [30].

2.2.3 Reinforcement Learning

Reinforcement learning is a machine learning paradigm in which an agent learns optimal behavior through interactions with an environment by taking actions, observing the resulting states, and receiving rewards or penalties based on chosen actions. One famous example is the Monte Carlo Tree Search (**MCTS**). The theoretical foundations of MCTS trace their origins to the Upper Confidence Bounds applied to Trees (UCT) algorithm formalized by Kocsis and Szepesvári in 2006 [32]. Further developments emerged in game theory, particularly Coulom's 2006 pioneering application of MCTS to the game Go, demonstrating its effectiveness in combinatorial state spaces [12]. MCTS gained more formality in 2012 through Browne et al. in their comprehensive survey, which unified variant implementations and established parameter conventions [5]. David Silver et al. further refined MCTS's real-time planning capabilities with the power of deep neural networks with the well-acknowledged **AlphaGo** in 2016 [54]. Combined with random sampling, it is an ideal choice for automated planning where traditional machine learning algorithms are computationally too expensive to be carried out, allowing the agent to excel in coordinating complicated actions in unexplored environments.

Algorithm 1 demonstrates an abstract workflow of the MCTS algorithm [5]:

Algorithm 1 Monte Carlo Tree Search (MCTS) [5]

Require: Initial state s_0, maximum iterations K
Ensure: Optimal action a^*
Create root node v_0 with state s_0
for $k = 1$ **to** K **do**
  $v \leftarrow v_0$ ▷ Start from root
  Selection:
  while v is not leaf node **do**
    Select child v' using:

$$a^* = \arg\max_{a \in A(v)} \left[\frac{Q(v,a)}{N(v,a)} + C\sqrt{\frac{\ln N(v)}{N(v,a)}} \right]$$

    $v \leftarrow v'$
  end while
  Expansion:
  if v not fully expanded **then**
    Expand new node v_{new} with action a_{new}
    $v \leftarrow v_{new}$
  end if
  Simulation:
  Perform random rollout to terminal state, get reward Δ
  Backpropagation:
  while $v \neq$ null **do**
    Update visit count: $N(v) \leftarrow N(v) + 1$
    Update cumulative reward: $Q(v) \leftarrow Q(v) + \Delta$
    $v \leftarrow \text{parent}(v)$
  end while
end for
return $a^* = \arg\max_{a \in A(v_0)} N(v_0, a)$

The Upper Confidence Bound (UCB) formula

$$a^* = \arg\max_{a \in A(v)} \left[\underbrace{\frac{Q(v,a)}{N(v,a)}}_{\text{Exploitation}} + C \underbrace{\sqrt{\frac{\ln N(v)}{N(v,a)}}}_{\text{Exploration}} \right] \tag{2.1}$$

balances exploitation of known rewards $Q(v,a)$, from action a at node v, with exploration $C\sqrt{\frac{\ln N(v)}{N(v,a)}}$, where $N(v,a)$ denotes action a's selections and $N(v)$ the parent node's total visits. The exploration constant C regulates this balance (normally set to $\sqrt{2}$)), while the natural logarithm (ln) ensures exploration decays appropriately as $N(v)$ increases. The action set $A(v)$ defines available choices at node v that maximizes the total rewards [12].

Noticeably, the application of reinforcement learning has been extended by researchers to the realm of retrosynthetic analysis, as introduced in Chapter 2.6. The **MCTS** algorithm also plays a pivotal role in the retrosynthetic design package **AiZynthFinder** as the main searching algorithm [18], which will be introduced in Chapter 2.6.2.

2.2.4 Deep Learning

Deep learning is a specific paradigm of machine learning based on the significant usage of multi-layered neural networks to analyze complex patterns difficult to be picked up by basic machine learning models. In tasks like computer vision, natural language processing, and speech recognition, this development enables machines to reach high performance levels matching or even surpassing humans [54].

Multi-layer perceptron (**MLP**) is a fundamental building block of artificial neural networks, first introduced by Frank Rosenblatt in 1958 [46]. Artificial neural networks function as interconnected artificial neurons, transforming input data into multiple layers of abstraction through intermediate activation functions. Once the network completes the feature abstraction, it back-propagates the model weights to reflect the significance of the features in relation to the objective. Based on this mechanism, deep learning removes the need for manual feature engineering by automatically abstracting features from raw data through stacked layers of networks.

2.3 Machine learning algorithms

In this project, two algorithms were taken into account for comparison studies, which are Graph Neural Network (**GNN**) [50] and **eXtreme Gradient Boosting** (**XGBoost**) [9].

2.3.1 Graph Neural Networks

In December 2008, Franco Scarselli et al. proposed a type of artificial neural network named Graph Neural Network (**GNN**) [50]. The GNN models are capable of carrying out deep-learning in graph-like data structures directly, either acyclic, cyclic, directed, or undirected, in scenarios such as molecular chemistry and computer vision. In the scope of this project, we are primarily interested in **GNN**'s application towards *de novo* drug discovery. Below is a piece of pseudocode for a general GNN framework with forward propogation.

Algorithm 2 Graph Neural Network (GNN)

Input: Graph $G = (V, E)$ with node features $\mathbf{X} \in \mathbb{R}^{|V| \times d}$
Output: Updated node representations $\mathbf{H}^{(K)} \in \mathbb{R}^{|V| \times h}$
Initialize $\mathbf{H}^{(0)} \leftarrow \mathbf{X}$ ▷ Initial node features
for $k = 1$ **to** K **do** ▷ Message passing iterations
  for each node $v \in V$ **do**
    $\mathbf{m}_v^{(k)} \leftarrow AGGREGATE^{(k)}\left(\{\mathbf{H}_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)$
    $\mathbf{H}_v^{(k)} \leftarrow \sigma\left(COMBINE^{(k)}\left(\mathbf{H}_v^{(k-1)}, \mathbf{m}_v^{(k)}\right)\mathbf{W}^{(k)}\right)$
  end for
end for
return $\mathbf{H}^{(K)}$

The input graph $G = (V, E)$ consists of nodes V and edges E, where each node $v \in V$ has an initial feature vector stored in the matrix $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ with d denoting the feature dimension. The parameter K controls the number of message-passing iterations, corresponding to the depth of the GNN layers. During each iteration k, the *AGGREGATE* function collects transformed features from neighboring nodes $\mathcal{N}(v)$, while *COMBINE* integrates these aggregated messages with the node's previous state $\mathbf{H}_v^{(k-1)}$. The learnable weight matrix $\mathbf{W}^{(k)} \in \mathbb{R}^{h_{in} \times h_{out}}$ transforms node representations at layer k, where h_{in} and h_{out} define the input/output dimensions of the hidden states. The nonlinear activation σ (e.g., ReLU or sigmoid) enables

model expressiveness. The final output $\mathbf{H}^{(K)} \in \mathbb{R}^{|V| \times h}$ contains refined node embeddings after K propagation steps, suitable for downstream tasks like node classification or regression.

Chemprop

Developed and introduced by Esther Heid et al. in *Chemprop: A Machine Learning Package for Chemical Property Prediction* in 2023 [27] , **Chemprop** leverages directed message-passing neural networks (**D-MPNNs**), a variation of graph neural networks (**GNN**). This package is extensively utilized in this project, in which the **GNN** module is one of the two algorithms being experimented upon for comparison studies.

2.3.2 eXtreme Gradient Boost

Introduced in year 2016 by Tianqi Chen and Carlos Guestrin, the eXtreme Gradient Boost (**XGBoost**) is an open source library to perform efficient, portable, and flexible gradient boosting in machine learning [9]. The Python version of this package is extensively used in this project to carry out comparison experiments. The algorithm of **XGBoost** is as follows:

Algorithm 3 eXtreme Gradient Boost (XGBoost)

Require: Training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, iterations T, learning rate η
Ensure: Boosted model $F_T(x)$
1: Initialize model: $F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2: **for** $t = 1$ **to** T **do**
3: Compute gradients: $g_i = \partial_{F_{t-1}} L(y_i, F_{t-1}(x_i))$
4: Compute hessians: $h_i = \partial^2_{F_{t-1}} L(y_i, F_{t-1}(x_i))$
5: Build tree f_t to minimize:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

6: Update model: $F_t(x) = F_{t-1}(x) + \eta f_t(x)$
7: **end for**

XGBoost combines gradient boosting with advanced regularization techniques [9]. The core objective function $\mathcal{L}^{(t)}$ contains two components: the loss approximation using second-order Taylor expansion:

$$\mathcal{L}^{(t)} = \sum \left[g_i f_t + \frac{1}{2} h_i f_t^2 \right] \tag{2.2}$$

and the regularization term:

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \|w\|^2 \tag{2.3}$$

that penalizes tree complexity through leaf count T and weights w. The gradients g_i and hessians h_i enable precise updates by capturing both first and second derivatives of the loss function L. Each iteration t adds a new tree f_t scaled by learning rate η to prevent overfitting. The tree construction uses gain calculation:

$$\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \tag{2.4}$$

to determine optimal splits, where G and H represent gradient sums in left/right nodes. This combination of boosting with L1/L2 regularization and parallel tree construction enables superior performance on structured data.

2.4 Hyperparameter Optimization

The first literature appearance of the term *hyperparameter* traces back to the work by Lindley and Smith in 1972 in their paper *Bayes Estimates for the Linear Model*, in which *hyperparameters* are defined as a hierarchical representation where the parameters of a linear model are themselves governed by additional parameters [35]. Hyperparameter optimization as a modern machine learning technique was first introduced by James Bergstra and Yoshua Bengio in 2012 in their article *Random Search for Hyper-Parameter Optimization* [3]. The concept of hyperparameter optimization (denoted as **HPO** from now on) aims to provide a method to search for the best combination of hyperparameters to train machine learning models to reach optimal performance metrics. There are two prominent paradigms of HPO, namely, grid search and random search, as explained in detail by Matthias Feurer & Frank Hutter in 2019 in their publication *Hyperparameter Optimization* [16]. In this project, **HPO** is utilized as a robustness evaluation of the training hyperparameters of the machine learning models, which will be discussed later in Chapter 4.7.

2.5 High-Performance Computing

High-Performance Computing (HPC) refers to the aggregation of advanced processing resources—such as supercomputers, clusters, and cloud-based infrastructures—to solve problems that require substantial computational horsepower beyond that of conventional desktops [22]. Dated back to the year 1975 when Richard M. Russell introduced the first supercomputer, *CRAY-1* [47], scientific research has been revolutionized by the increasinly more available and affordable computation powers. Over the decades, HPC has become indispensable in academic research by enabling the simultaneous processing of vast datasets and complex scientific models.

Modern HPC systems integrate innovative technologies such as cloud computing, virtualization, and GPU-accelerated frameworks to optimize resource allocation and boost computational efficiency [41, 22]. Central to HPC is the practice of parallel computing, wherein large problems are decomposed into numerous smaller subtasks that run independent of each other. A particularly common paradigm is known as *embarrassingly parallel* [17]. In embarrassingly parallel computing, the overall computational task naturally splits into independent subtasks that require minimal or no inter-task communication, thereby simplifying algorithm design and allowing for nearly linear scalability in performance.

2.5.1 Academic Leiden Interdisciplinary Cluster Environment (ALICE)

The *Academic Leiden Interdisciplinary Cluster Environment*, known as **ALICE** for short [34], is one of these clusters for academic researchers where a majority of the scientific computation of this project took place. Equipped with `44` computing nodes and `68` GPUs, **ALICE** offers a considerable computation capability with its powerful infrastructure – `1109 TB` of storage, `17.9 TB` of RAM, `1234 TFlops`, and `1712 cores (3424 threads)`. ALICE is **CUDA**-capable and **Slurm**-enabled [29] that allow heavy computing tasks to be carried out in an embarrassingly parallel manner [17].

2.5.2 Simple Linux Utility for Resource Management (Slurm)

Developed by Morris A. Jette and Tim Wickberg [29], **Slurm** is an open-source workload manager designed to encompass tasks for high concurrency running in **HPC** clusters. By using its GNU Bash command `sbatch` in the format of `sbatch -array=1-9000 -i` $my\_in\_\%a$ `-o` $my_out\%a$ `my.program`, the user can

specify how many instances of the program should be spawned using the `-array` flag. On **ALICE** the maximum limit of the size of the `-array` flag is set to `1000` [34].

2.6 Computer-Aided Synthesis Planning and Retrosynthesis

Chemical synthesis is a process to experimentally assemble bigger molecules using known chemical reactions based on smaller, easy-to-obtain precursors. Retrosynthetic analysis, or retrosynthesis for short, is a process to trace back the target molecule from a chemical synthesis back to available precursors. Traditionally, planning for chemical synthesis involves primarily chemists' knowledge of the characteristics of the functional groups and various reactions, as detailed by Corey and Cheng in *The Logic of Chemical Synthesis* back in the year 1989 [11]. Nowadays, with the power of artificial intelligence and machine learning, the planning of chemical synthesis has been revolutionized with a new active field of research – *Computer-Aided Synthesis Planning* – denoted as **CASP** in short. Instead of exhausting hundreds of years of chemistry knowledge, chemists nowadays can rely on **CASP** to plan an optimal synthetical route to achieve their chemical synthesis objectives.

Pioneered by Corey and Wipke, the concept of *Computer-Assisted Design of Complex Organic Syntheses* was first introduced in 1969 [10]. In 2018, by training a neural network to recognize chemical reactions and later use it as the policy for **MCTS**, Segler, Preuss, and Waller demonstrated the feasibility of applying reinforcement learning on retrosynthetic tasks with deep neural networks and symbolic AI. [52]. **GNN**-based retrosynthetic analysis tooling has also gained popularity in recent years. In 2022, two groups of researchers demonstrated their advances on applying **GNN** to retrosynthetic planning. Presented by Liu el al, *RetroGNN* focuses on synthesizability[1] estimation in generative drug discovery pipelines [36]. Zhao et al. demonstrated *GNN-retro,* which, despite its similar name, specializes in providing better insights on cost estimation for candidate synthetic routes in a large search space to reach the target molecule during the *de novo* drug discovery process [23]. Being directly inspired by their prior work, **GNN** was adopted as one of the two algorithms being studied in this project.

2.6.1 Synthetic route

A synthetic route, as explained by Corey and Cheng in 1989, is a pathway derived by the retrosynthesis process from the precursor to be converted to the target molecule [11]. A synthesis task may have more than one synthetic route, which necessitates synthesis planning to select the most optimal synthetic route to carry out experiments. For chemists, the length of the synthetic route is the biggest decision factor that determines the difficulty of a synthesis task. Other factors such as reagent toxicity and cost are also important decision factors to determine an optimal synthetic route. Researchers in this novel field are motivated by such factors as the synthetic route scoring system *Routescore* by Seifrid et al. in 2022 and synthesizability estimation *RAScore* by Thakkar et al. in 2021 [53, 59].

[1] The ability of finding a synthetic route to a target molecule and ensuring such a molecule could be produced

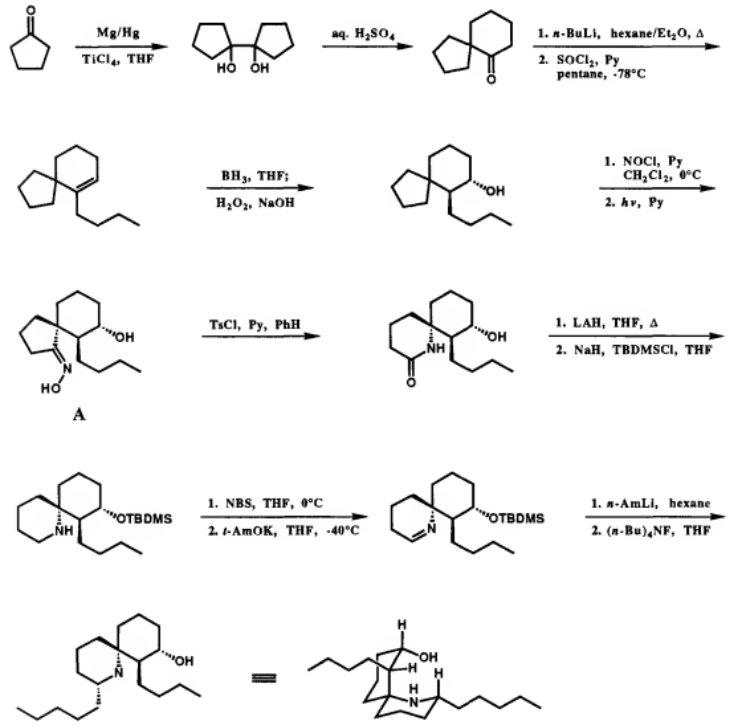

Figure 2.6.1.1: Synthetic route of *(±)-Perhydrohistrionicotoxin* from Corey and Cheng's *Logic of Chemical Synthesis* [11]

2.6.2 AiZynthFinder

AiZynthFinder is an open-source software dedicated to retrosynthetic planning [18]. As introduced by Genheden et al. in 2020, given a target molecule, **AiZynthFinder** leverages **MCTS** to recursively search for purchasable chemical compounds that can be used as a precursor for the retrosynthetic route being searched on. Since its first release, **AiZynthFinder** has gone through several upgrades that significantly enhanced its features and capabilities. In 2024, Saigiridharan and Hassen et al. introduced the *4.0* version of **AiZynthFinder**, incorporating new policies for filter reactions, support for any one-step retrosynthesis model and scoring framework, and additional search algorithms other than **MCTS** [48]. Details of some crucial features for of **AiZynthFinder** will be illustrated in detail in this section.

Building blocks and Stocks

Building blocks, as indicated in **AiZynthFinder**, represent the group of chemical compounds that meet the winning state of **MCTS** as available or purchasable, implying that those molecules could be obtained to carry out synthesis in a laboratory. The abstracted collection of the building blocks is denoted in **AiZynthFinder** as "**Stocks**". Additionally, stocks can be customized to fit specific rules, such as limiting the amount of carbon atoms in a compound [18].

In **AiZynthFinder**, the authors obtained a list of purchasable chemical compounds as building blocks, denoted as the **ZINC** stock that originated from the *ZINC 15 – Ligand Discovery for Everyone* database, proposed by Sterling and Irwin in 2015 [56]. Compounds in the **ZINC** stock were converted to the **SIMLES** format (introduced in Chapter 2.6.3) for better search compatibility [18].

Reaction templates

Genheden et al. categorized retrosynthesis planning algorithms as template-based and template-free in their publication *AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning* in 2020 [18]. For CASP algorithms based on templates, a reaction template is a collection of well-documented chemical reactions that serve as the knowledge base for retrosynthetic algorithms when planning a retrosynthesis, similar to the supervised learning approach introduced in Chapter 2.2.1. For template-free algorithms, the "knowledge base" is replaced with another machine learning model, usually Transformer models [61] that excel at pattern approximation in large amounts of data, which serves as the search policy. Examples in the latter category include a self-correcting transformer neural network for retrosynthetic prediction presented by Zheng et al. in 2019 and the augmented transformer by Tetko et al. in 2020 [65, 58].

AiZynthFinder implements a hybrid approach consisting of template-based strategies with Monte Carlo Tree Search (MCTS) guided by machine learning. The **AiZynthFinder** flavor of the MCTS consists of the following structures: *root* node denoting the target molecule at initialization of the search, *leaf nodes* representing available, purchasable compounds or unexplored branches that need further expansion, and edges represent applied reaction templates [18]. The **AiZynthFinder** flavor of the MCTS framework operates through four key phases: [2]

1. **Selection**: The algorithm traverses the synthetic tree from the *root* (target molecule) to the *leaf* nodes using the Upper Confidence Bound (UCB) scores outlined in Equation 2.1. The balance of UCB weighs the exploration of new routes against the exploitation of high-probability routes [5].

2. **Expansion**: At chemically viable leaf nodes, which are neither terminal nodes nor purchasable compounds, a multilayer perceptron neural network (MLP) trained on **USPTO** reaction data [18, 37] ranks the applicable reaction templates by predicted applicability ($P_{template} \in [0, 1]$). The upper k (k = 50 usually) templates generate precursor nodes through simulated retrosynthetic disconnections.

3. **Simulation**: Recursive roll-out continues until all precursors reach purchasable compounds, verified by the ZINC stock filters described in Chapter 2.6.2, or the maximum search depth ($d_{max} = 9$ in our case, as illustrated in Table 2.6.2.1) is reached [18]. Terminal states receive scores based on synthetic accessibility metrics.

4. **Backpropagation**: Pathway evaluation scores propagate backward to update route metrics based on precursor availability and route lengths [18].

AiZynthFinder parameters

AiZynthFinder has many parameters to adjust for to produce ideal retrosynthetic analysis. The table below shows the parameters chosen for the data generation process for this experiment, as explained in detail in Chapter 4.3. The description of the parameters is obtained from the official documentation of **AiZynthFinder** [18].

[2] According to the algorithmic workflow described in the documentation of **AiZynthFinder** by Genheden et al. [18].

Parameter	Value	Description
C	1.4	Balance factor of exploitation and exploration in MCTS
Cutoff cumulative	0.995	Accumulative probability of the suggested templates is capped at
Cutoff number	50	Maximum number of templates returned from the expansion policy
Max transforms	9	Maximum depth of the search tree
Default prior	0.5	Prior that is used if policy-provided priors are not used
Use prior	True	If True, priors from the policy are used instead of the *default_prior*
Return first	True	If True, the tree search will be terminated as soon as one solution is found
Iteration limit	1000	The maximum number of iterations for the tree search
Time limit	900	The maximum number of seconds to complete the tree search
Exclude target from stock	True	If True, the target in stock will be broken down
Template column	retro_template	The column in the template file that contains the templates
Filter cutoff	0.05	The cut-off for the quick-filter policy
Prune cycles in search	True	If True, prevents the MCTS from creating cycles by recreating previously seen molecules when it is expanded
Additive expansion	False	Expand routes additively
Search algorithm	mcts	Search algorithm for AiZynthFinder
Post processing	min_routes: 5, max_routes: 25, all_routes: False,	

Table 2.6.2.1: Parameters of **AiZynthFinder** being applied during this project for training data generation

2.6.3 Simplified Molecular-Input Line-Entry System (SMILES)

Introduced by David Weininger in 1988, the **SMILES** notation has paved the way for modern chemical information processing in CASP. Designed according to molecular graph theory, **SMILES** is capable of abstracting complex chemical structures into machine-compatible strings with minimal lexical effort [62]. By specifying and standardizing chemical notations for atoms, bonds, cyclic and disconnected structures,

and aromaticity, the **SMILES** notations excel at denoting and distinguishing unique chemical properties of different molecules. For example, the molecule *ethanol*'s molecular formula is CH_3CH_2OH. In **SMILES** notation, such a molecular formula is denoted as "CCO" that preserves the fundamental atoms and functional groups that consistitute the molecule and omits lengthy `C-H` and `O-H` bonds in the expression.

2.6.4 Data conversion and encoding

In the fields of machine learning and data science, the first step in data preparation is always to convert raw data into compatible vectors that models can process and perform actions on. In this project, a more specific use case for such a vectorization process is to convert the **SMILES** strings to vectors that represent certain molecular and chemical properties. In general, the process works as follows: first, a molecular fingerprint is chosen based on the availability and reliability of the conversion so that the converted vectors are model-compatible; next, based on molecular properties taken into account by the molecular fingerprint, the **SMILES** strings are converted into machine-readable vectors. Finally, the converted features encoded by the molecular fingerprints are sent as inputs to the machine learning models[3].

In this project, it is worth mentioning that some specific molecular fingerprints were attempted or considered during the experimentation phase. In the beginning, the molecular fingerprint of `Molecular ACCess System` (**MACCS**) [44], as a part of the CASP library `RDKit`, was attempted to encode the molecules. It was later discovered that this molecular fingerprint lacks expressibility in a machine learning task due to its limited ability[4] to capture only certain chemical characteristics but not the whole molecular structure. A second attempt with a different molecular fingerprint called `Extended Connectivity Fingerprints` [45], or **ECFP** in short, succeeded. Introduced by David Rogers and Mathew Hahn in 2010, the **ECFP** molecular fingerprints specialize in identifying circular atomic environments and encoding them with variable fingerprint lengths that offer a wide range of flexibility and compatibility [45]. On the other hand, due to its high flexibility in fingerprint lengths, it sometimes incurs a rather high computational cost as a trade-off. Below is a picture illustrating the workflow of creating an **ECFP** molecular fingerprint[5] [45]:

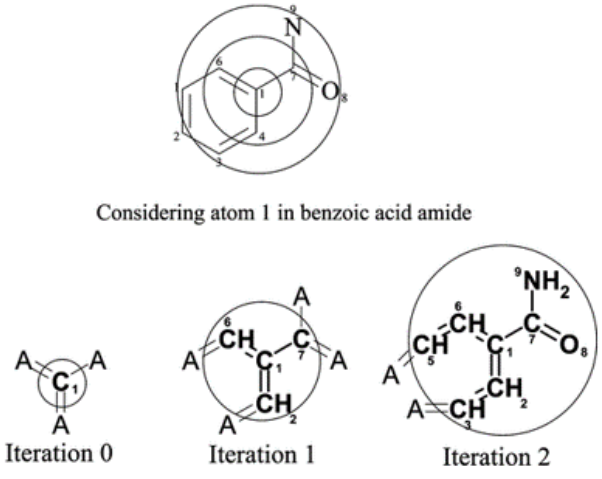

Figure 2.6.4.1: Illustration of the fingerprint creation process in **ECFP** for benzoic acid amide by Rogers & Hahn

[3]For Chemprop GNN models, the molecular fingerprint conversion step is not needed due to its message-passing capability on the molecular graphs [27].

[4]Only **ECFP** is compatible with the Python package of **XGBoost** to encode the **SMILES** strings. For the **Chemprop GNN** package encoding is not needed, as explained later in Chapter 6.2.1.

[5]In this project, we adopted the 2,048-bit encoding and 3 iterations of the **ECFP** encoding to process **SMILES** strings.

2.7 Related Work on Synthesizability Scoring

Ertl and Schuffenhauer gave the definition of *synthetic accessibility score* (denoted as **SAScore**) in 2010 [14], which quantified the molecular synthetic difficulty level for the first time. In 2021, Thakker et al. leveraged machine learning to perform binary classification of the possibility of a molecule's synthesizability[6] as *Retrosynthetic accessibility score* (**RAScore**) [59]. In 2022, Liu el al. introduced a regression **GNN** model as the synthesizability predictor [36]. Cost estimation in synthetic route scoring has been carried out by Seifrid et al. in 2022 with their work *Routescore: Punching the Ticket to More Efficient Materials Development* and by Han and Peng et al. in the paper *GNN-Retro: Retrosynthetic Planning with Graph Neural Networks* in 2021 [53, 24]. One of the most recent advances in such endeavor is *Estimating the synthetic accessibility of molecules with building block and reaction-aware SAScore* by Chen and Jung in 2024 that further expanded the scope of application of **SAScore** [8]. These research works paved the way to approximate synthetic route planning based in CASP.

[6]The ability to find a synthetic route in a CASP software

Chapter 3

Research question

As being motivated by the prior researches based on synthesizability estimation [59, 36] and synthetic route scoring [14, 24, 53, 8] introduced in Chapter 2.7, we aim to determine the feasibility of using machine learning to predict the synthetic route properties calculated by the CASP tool **AiZynthfinder** [18]. Specifically, experiments will be carried out to investigate if the shortest synthetic route lengths of molecules solved by **AiZynthfinder** could be predicted directly from **SMILES** strings with machine learning models trained using algorithms **XGBoost** and **GNN**, which is the core research question of this thesis.

Chapter 4

Methodology

4.1 Task formulation

The research question presented in Chapter 3 necessitates the need for a formulation of the machine learning task. Conventionally, a supervised machine learning task can be carried out in either classification or regression based on the nature of the task. In this thesis, with consideration of data interpretability and chemical intuition, both machine learning paradigms are experimented with.

4.1.1 Multiclass classification for synthetic route length prediction

The multiclass classification, as introduced in Chapter 2.2.1, takes the molecular structure in the format of an encoded **SMILES** [62] string as input, which was covered in Chapter 2.6. It then outputs the predicted class that the target molecule belongs to with the highest level of probability. The class segregation is designed to incorporate the synthetic difficulty of the input target molecule based on chemical convention. Generally, being able to synthesize a molecule in 1-2 steps is considered easy. Completing the synthesis in 3-5 steps is considered a normal difficulty level. And for a synthetic route longer than 6 steps, it is perceived as a hard chemical practice that will likely negatively impact the output amount and quality of the final product. There is an additional "unsolved" class alongside those three synthesizable classes for target molecules that the CASP tool **AiZynthfinder** [18] is unable to find a route for, making the multiclass classification task have 4 classes in total.

4.1.2 Regression for synthetic route length prediction

Similar to the multiclass classification task, an encoded **SMILES** [62] string is also needed as the input for the regression task. Chemically speaking, in nature, organic molecules grow in size by establishing and concatenating carbon-carbon bonds continuously on top of a base molecule [10]. Hence, the growth of organic compounds could be seen as a continuous process based on discrete amounts of atoms, of which the length of the synthetic route could be predicted by machine learning using a regression model.

4.2 Experiment framework

In this section, the software framework, design patterns, and related technical details for this project are outlined. The development environment is configured leveraging the convenience of remote access to the **ALICE** HPC (as introduced in Chapter 2.5) cluster of Leiden University [34] via **Microsoft Visual Studio Code** [40], where all development work and experiments were completed. Development of the experiment framework[1] was achieved via **Python** 3.7.12 [43]. A virtual environment was created via **Anaconda** [2].

4.2.1 Design patterns

During the development, design patterns were followed such that modularity, code reuse, and automation were made possible for faster troubleshooting and easier functionality expansion. The parameter passing was configured through the instantiation of an **Experiment** object, where relevant parameters for an individual experiment to take place were passed and set, namely, the size of the training data set partitions (introduced in Chapter 4.3.4), the choice of machine learning model, the specification of multiclass classification and regression, etc. Crucial and repetitive steps in individual experiments were extracted out as encapsulated into helper functions. Additionally, data as a product of overlapping intermediate steps of multiple individual experiments were saved and reused to further reduce computation time and energy consumption. Below is a design diagram illustrating such considerations:

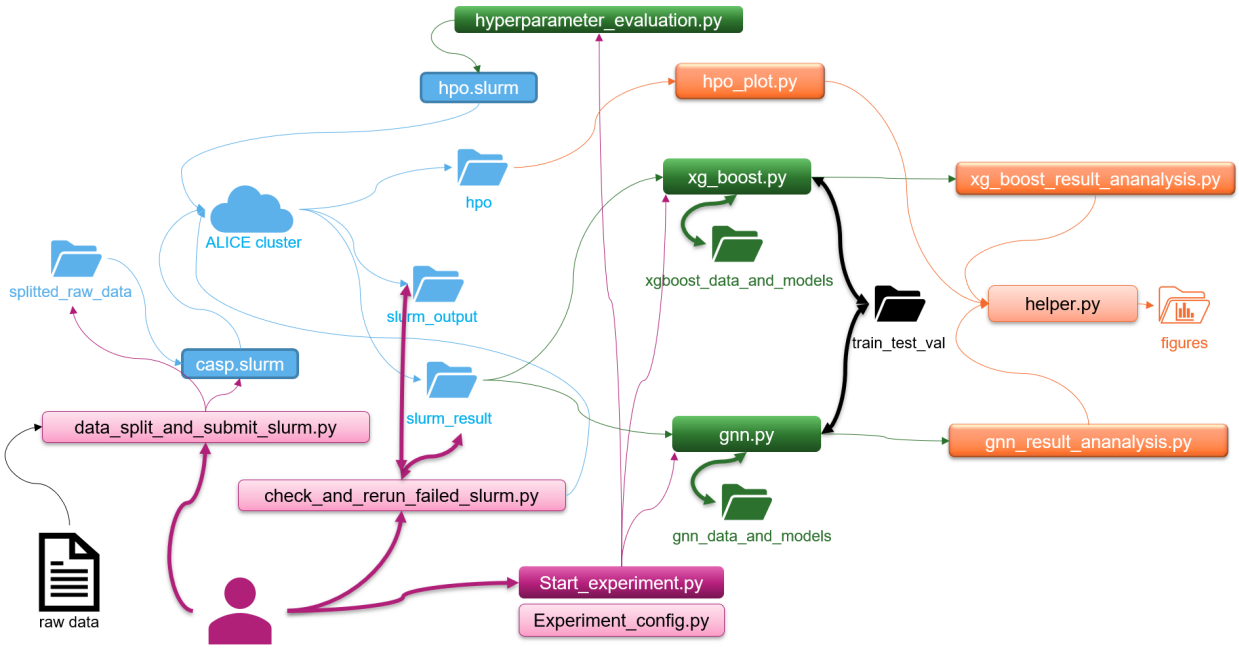

Figure 4.2.1.1: Flowchart of the design pattern and the interaction of modules. Arrows in this graph indicate the direction of flow of files and data from one module to another. Modules that serve the same purpose are labeled with the same color. This figure demonstrated the modular design patterns taken into account during the software development process and the flow of data among modules.

4.2.2 Experiment pipeline

As stated in Chapter 4.2.1, to achieve faster experiment completion, modular design and code reuse were incorporated into the experiment framework, which enables the implementation of experiment pipelines.

[1] Code is available at: `https://github.com/penguin9360/master_thesis`

Specifically, an experiment pipeline consists of modules of the experiment framework such that by specifying the experiment name, size, model name, learning task, and the option to enable or disable model training and data evaluation in file *start.py*, corresponding components of the framework are called and executed based on the user's selection. It enables one-click-run-everything with visualization of all individual experiment results in one go, which significantly accelerated the scaled-up experiments after the proof-of-concept (POC) stage. Below is an abstract flowchart of the experiment pipeline:

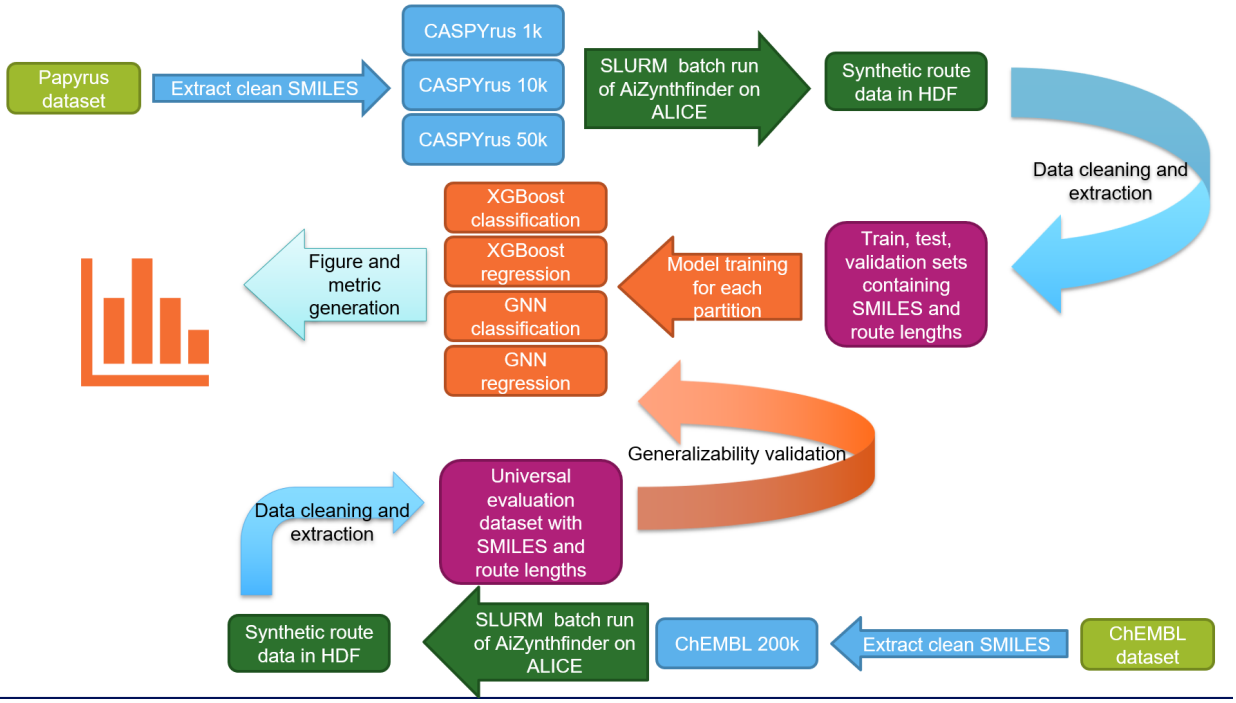

Figure 4.2.2.1: This figure shows the experiment pipeline abstracted as a flowchart, with modules that serve the same purpose labeled with the same color. From the top left and bottom right, raw **SMILES** data serving as the first layer of the input are processed to become training and generalizability evaluation data sets, thereby entering the corresponding pipelines of machine learning model training, testing, generalizability evaluation, figures plotting, and metrics exporting.

Similar to the experiment pipeline, at a certain point in the project, cross-validation of one machine learning model on another validation data set is needed. Thus, an expansion of the experiment framework is carried out, namely, the inference framework. Utilizing the modular design pattern, the inference framework enables any trained models to be evaluated for generalizability and performance on larger datasets than their own evaluation sets.

4.3 Data preparation

The nature of this project, prediction from raw **SMILES** [62] to the shortest synthetic route length computed by **AiZynthfinder** [18], demands a massive quantity of data for model training and evaluation. Therefore, data preparation is a key step to ensure data quality and thus lead to convincing performance metrics of machine learning models.

4.3.1 CASPYrus 50k

The **CASPYrus50k** dataset is a clustered data set constituting `50,000` bio-active molecules derived from *Papyrus: a large-scale curated dataset aimed at bioactivity predictions* [6], that has been further curated into a standalone CASP data set by Hassen et al. in 2025 in their publication *Generate what you can make: achieving in-house synthesizability with readily available resources in de novo drug design* [25].

In this project, the source of training data of all machine learning models were derived, either as a part or a whole, from the **CASPYrus50k** dataset.

4.3.2 ChEMBL 200k

The **ChEMBL 200k** is originated from the **ChEMBL database**, a database for drug-like, bio-active small molecules [64]. Hassen et al. converted the extracted `200,000` molecules and developed it into a CASP data set [25]. It consists of `200,000` bio-active molecules with calculated chemical properties and is used as the universal evaluation data set for all trained machine learning models for their generalizability and performance metrics.

4.3.3 Slurm job submission and automation

The **Slurm Workload Manager** was introduced in Chapter 2.5.2. Since this project requires a large amount of data to be generated for training, validating, and testing models, we adopted parallelism paradigm of *embarrasingly parallel* (introduced in Chapter 2.5) to speed up the process. Up to `1,000` instances of **AiZynthfinder** were created in parallel on the **ALICE** cluster, coordinated by **Slurm** [18, 34, 29].

To initiate a **Slurm** job, a script is needed to submit the job to the **ALICE** HPC cluster [34] as introduced in Chapter 2.5.1. Conventionally, to perform one batch experiment, the following steps are required. First, raw **csv** files consist of thousands of **SMILES** need to be split into groups of smaller chunks, with each chunk carrying `1/1,000` of the amount of the original **SMILES** entries[2]. Next, a modification of the **Slurm** file is needed to specify **Slurm** job properties such as file chunk location, the beginning and end indices, and the partitions.

This process, if done manually, is highly complex and error-prone. Thus, automation was applied to speed up the **Slurm** job submission workflow. **Python** scripts were developed to automatically allocate the size, location, and amount of file chunks before the first step of data splitting. Afterwards, a series of interactive prompts will be displayed in the command line to instruct the users with the choice of partitions, resource allocation, and time limits. Finally, to avoid manual overhead, it offers the option to overwrite the **Slurm** file and submit the batch job to the **ALICE** HPC cluster automatically via the **Python** script.

4.3.4 Training data partition and cleaning

As introduced in Chapter 4.3.1, smaller partitions of the **CASPYrus50k** data set are taken as the training data for smaller, proof-of-concept models. Experiments were conducted with smaller models first to verify feasibility and then moved on with scaled-up training data consisting of more **SMILES** entries.

[2]The amount of `1,000` concurrent instances was a design consideration due to the rules on the **ALICE** cluster [34].

CASPyrus 1k

The training of machine learning models began with extracting the first `1,000` **SMILES** entries of the **CASPYrus 50k** data set, hence the name of **CASPyrus 1k** for the data set. This data set is crucial for setting up the data processing pipeline and all proof-of-concept experiments for all modules of the experiment framework.

CASPyrus 10k

Similar to **CASPyrus 1k** introduced in Chapter 4.3.4, **CASPyrus 10k** constitutes the first `10,000` **SMILES** entries from **CASPyrus 50k**. The purpose of having a data set in such proportion is to establish an in-between argument of the models'performance metrics from **CASPyrus 1k** to **CASPyrus 50k**, as it is expected that the model's performance would increase as the size of the training data grows.

CASPyrus 50k

The vast majority of **CASPyrus 50k** is used as training data in the end to demonstrate the performance of machine learning models. It is worth mentioning that during the data generation phase using **Slurm**, as stated in Chapter 4.3.3, there were occasional unrecognizable molecules that prevented the execution of the **Slurm** batch process. Those incompatible **SMILES** were removed to ensure successful data generation, leading to a slight loss of training data for the models trained on **CASPyrus 50k**.

4.3.5 Split of train, test, and validation

The last step of data preparation is to split the training data, in every partition, into a training set, a validation set, and a test set, as a convention of machine learning tasks. All raw **CASPYrus** partitions was further divided into three data sets of train, test, and validation that are ready to be consumed by machine learning models in a ratio of:

$$\texttt{train} : \texttt{test} : \texttt{validation} = 0.8 : 0.1 : 0.1$$

Partition	**CASPYrus** Training	**CASPYrus** validation	**CASPYrus** Test	Generalizability Evaluation
CASPYrus 1k	800	100	100	ChEMBL 200k
CASPYrus 10k	8,000	1,000	1,000	ChEMBL 200k
CASPYrus 50k	40,000	5,000	5,000	ChEMBL 200k

Table 4.3.5.1: Splits of training, validation, testing, and additional generalizability evaluation of data sets. Numbers in the table denote how many molecules are used to formulate one data set.

4.4 Metrics

Before we proceed with result evaluation, we first determine what exact metrics should be used to evaluate the training, validation, and testing performances of the various models based on the learning tasks of multiclass classification and regression. Table 4.4.0.1 offers a brief overview of the choices of training objectives and evaluation metrics being used during the experiment.

Task	Model	Training Objective	Evaluation Metric
Regression	GNN	MSE	RMSE
Classification	GNN	Categorical Cross Entropy	F1 - MCC
Regression	XGBoost	RMSE	RMSE
Classification	XGBoost	Categorical Cross Entropy	F1 - MCC

Table 4.4.0.1: Overview of training and evaluation metrics

4.4.1 Training objectives

In this experiment, we have chosen different training objectives based on different types of learning tasks. For multiclass classification tasks, `categorical cross entropy` (**CCE**) is widely adopted as the default training objective in many open-source machine learning frameworks. **CCE** measures the discrepancy between the true label distribution and the predicted probabilities generated by the trained model. It penalizes deviations in predicted probabilities from ground-truth labels, making it effective for guiding optimization during training [4].

The categorical cross-entropy loss for a batch of N samples and C classes is computed as follows. For each sample i, let $\mathbf{y}_i \in \mathbb{R}^C$ represent the one-hot-encoded true label, and $\mathbf{p}_i \in \mathbb{R}^C$ denote the predicted class probabilities from the model. The **CCE** loss L is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c}), \tag{4.1}$$

where $y_{i,c}$ and $p_{i,c}$ correspond to the c-th element of $\mathbf{y}_i$ and $\mathbf{p}_i$, respectively. Minimizing this loss encourages the model to align its predictions with the true distribution [19].This loss is typically minimized using gradient-based optimization methods such as stochastic gradient descent (SGD) or Adam [31], where gradients are computed via backpropagation to update model parameters iteratively.

For regression tasks, the `Root Mean Squared Error` (**RMSE**) was initially selected as the primary evaluation metric due to its sensitivity to large errors and interpretability in the target variable's units. However, due to software compatibility constraints encountered during implementation, `Mean Squared Error` (**MSE**) was ultimately employed as the training objective for the regression component of the GNN experiments.

MSE measures the average squared difference between predicted and actual values, providing a differentiable loss surface that facilitates gradient-based optimization [26]. The RMSE counterpart represents the square root of MSE, maintaining identical unit dimensions as the target variable [7].

In the format of mathematical formula, the MSE is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.2}$$

For completeness, RMSE is just the square root of the MSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.3}$$

Where y_i represents the true value, $\hat{y}_i$ denotes the predicted value, and n is the number of observations.

4.4.2 Evaluation metrics

The metrics chosen for evaluation differ from the training objectives to analyze the performance of machine learning models. the model. For multiclass classification, two additional metrics are used for result evaluation: `F1` and `Matthews Correlation Coefficient` (**MCC**). RMSE remains the evaluation metric for the regression tasks.

The F1 score, first introduced by Cornelis Van Rijsbergen in 1979, is often used to evaluate classification models by combining precision and recall [60]. Given an input, a true prediction or a false prediction could be performed by a classification model, which are denoted by TP and FP, respectively. Precision assesses the proportion of correctly identified positive instances among all predicted positives. This metric reflects the model's ability to minimize false positives and ensure the reliability of its predictions. Recall, in contrast, evaluates the model's capacity to quantify the proportion of true positives that the model successfully detects out of the total actual positives. It emphasizes the effectiveness of the model in capturing the complete set of relevant outcomes. Together, these two metrics give the F1 score an advantage to be used in an unbalanced data set. The formula of F1 is as follows:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

where:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$

`The Matthews Correlation Coefficient` (**MCC**), first introduced by Brian W. Matthews in 1975, provides a comprehensive perspective using confusion matrix elements [39]. Its formula

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.5}$$

yields values in $[-1, 1]$, where 1 indicates perfect prediction and -1 completely false classification. Unlike F1, MCC considers true negatives and remains reliable for both balanced and unbalanced data.

4.5 Training

Chapter 4.3.5 outlined the data split for training, validation, and testing for each trained dataset-task combination. Chapter 4.1 discussed the intuition of the formulation of machine learning tasks that involve both multiclass classification and regression. With two machine learning algorithms of interest, **Graph Neural Network** from the package **Chemprop** [28] (referred to as "*GNN*" from now) and **XGBoost** [9] as introduced in Chapter 2.2, a comprehensive list of 12 training tasks is enumerated below:

1. Training on **CASPYrus1k** (referred to as "*1k*" from now)
   (a) `1k GNN Multiclass classification`
   (b) `1k GNN Regression`
   (c) `1k XGBoost Multiclass classification`
   (d) `1k XGBoost Regression`

2. Training on **CASPYrus10k** (referred to as "*10k*" from now)
   (a) `10k GNN Multiclass classification`
   (b) `10k GNN Regression`
   (c) `10k XGBoost Multiclass classification`
   (d) `10k XGBoost Regression`
3. Training on **CASPYrus50k** (referred to as "*50k*" from now)
   (a) `50k GNN Multiclass classification`
   (b) `50k GNN Regression`
   (c) `50k XGBoost Multiclass classification`
   (d) `50k XGBoost Regression`

After training, models are tested on their individual test sets created during the train-test-validation split from the raw data sets, as specified in Chapter 4.3.5. Performance metrics during training are collected, visualized, and presented in Chapter 5.

4.5.1 Initial Choice of hyperparameters

During initial training tasks, it is observed that most models in Chapter 4.5 exhibited the characteristics of underfitting to their designated training sets. Therefore, as a measure of improving training quality, the parameters were set higher than their default values before the final round of training for both **GNN** and **XGBoost** models.

Due to the intrinsic difference of architecture between **GNN** and **XGBoost** models, not all hyperparameters could be mapped in between the two models on a one-to-one basis. As such, only two hyperparameters are chosen as they are shared by both models.

Parameters	Chosen values	Default value for GNN	Default value for XGBoost
Number of epochs	**150**	50	100
Max depth	**6**	3	6

Table 4.5.1.1: Selected metrics for model training

4.6 Evaluation

The evaluation steps are carried out after the training tasks specified in Chapter 4.5 completed. The purpose of evaluation is to verify generalizability of the trained models. Two sets of evaluation tasks were performed to address both areas of concern: inference of the trained models on the **ChEMBL 200k** data set and smaller **CASPyrus** models on larger **CASPYrus** models' test sets. For the experiment setup, since the most relevant evaluation to the generalizability argument is the robustness of the model's performance on an unforseen data set with unknown data distributions, hence we only discuss the evaluation that took place on the **ChEMBL200k** dataset.

In order to preserve the authenticity of the performance metrics, an automated scan and removal of any potentially duplicated **SMILES** entries in training and inference sets were carried out before the evaluation tasks.

4.6.1 Inference on ChEMBL200k as the universal validation set

The **ChEMBL 200k** data set, as stated in Chapter 4.3.2, is used as a universal evaluation set for all trained models for their generalizability. As neither the **SMILES** were known to the models trained on **CASPYrus 50k** data sets nor the distribution of them, the generalizability of the model's performance metrics is thus ensured. Below is an enumeration of each trained dataset-task combination that are evaluated on the **ChEMBL 200k** data set:

1. Generalizability Evaluation of *1k* models
    (a) `1k GNN Multiclass classification` model inferred on **ChEMBL 200k**
    (b) `1k GNN Regression` model inferred on **ChEMBL 200k**
    (c) `1k XGBoost Multiclass classification` model inferred on **ChEMBL 200k**
    (d) `1k XGBoost Regression` model inferred on **ChEMBL 200k**
2. Generalizability Evaluation of *10k* models
    (a) `10k GNN Multiclass classification` model inferred on **ChEMBL 200k**
    (b) `10k GNN Regression` model inferred on **ChEMBL 200k**
    (c) `10k XGBoost Multiclass classification` model inferred on **ChEMBL 200k**
    (d) `10k XGBoost Regression` model inferred on **ChEMBL 200k**
3. Generalizability Evaluation of *50k* models
    (a) `50k GNN Multiclass classification` model inferred on **ChEMBL 200k**
    (b) `50k GNN Regression` model inferred on **ChEMBL 200k**
    (c) `50k XGBoost Multiclass classification` model inferred on **ChEMBL 200k**
    (d) `50k XGBoost Regression` model inferred on **ChEMBL 200k**

4.7 Hyperparameter Optimization

Hyperparameter optimization, as introduced in Chapter 2.4, is a key process for validating the effectiveness and robustness of hyperparameters applied to models during training [16]. In essence, hyperparameter optimization operates by repeatedly training a machine learning model with various combinations of hyperparameter values, with the aim of searching for the combination that provides the most robust performance. Two prominent search algorithms are: grid search and random search. Grid search intuitively operates by specifying candidate datapoints in a list for each hyperparameter that forms a "grid," where each grid point corresponds to a combination of different specified values of hyperparameters. Random search, in contrast, automatically selects random combinations of hyperparameter values from the given distribution within the chosen upper and lower boundaries for each hyperparameter.

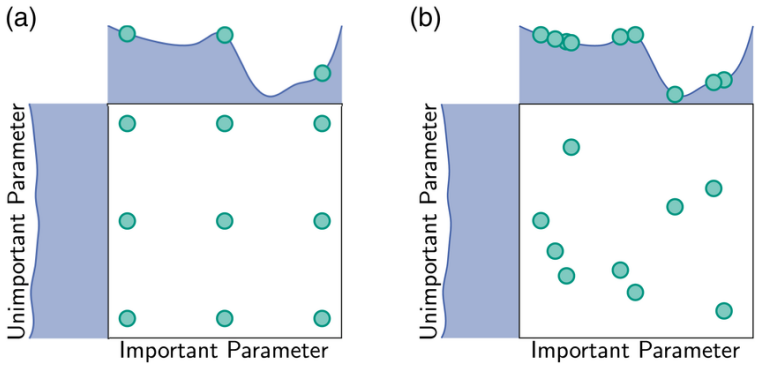

Figure 4.7.0.1: Illustration of Grid Search vs. Random Search by Feurer & Hutter in their publication *Hyperparameter Optimization*

In this project, we performed a hyperparameter optimization **random search** for the `50k Chemprop GNN` models to verify the robustness of our chosen (and package-default) hyperparameter values used to train the models. An embarrassingly parallel random search approach has been implemented using **Slurm** to accelerate the process, due to the nature of random searches and the fact that each individual evaluation is independent of another. Considering that hyperparameter optimization is an extremely time-consuming process, we had to make compromises. First, due to time constraints, the scale has been limited to `100` evaluations in total from 2 separate random search runs. Second, we do not seek to search for the best combination of hyperparameters that could potentially yield the best performance metrics, but rather demonstrate the robustness of the chosen and default hyperparameter values. The table below demonstrates the experiment setup for the hyperparameter optimization experiments:

Parameters	HPO run #1	HPO run #2
Number of parallel instances	5	10
Number of evaluations per instance	10	5
Epochs boundary	(50, 250)	(10, 200)
Depth boundary	(3, 9)	
Initial learning rate boundary	$(5 \times 10^{-5}, 1.5 \times 10^{-4})$	
Max learning rate boundary	$(7.5 \times 10^{-4}, 1.25 \times 10^{-3})$	
Batch size boundary	(48, 96)	

Table 4.7.0.1: Setup of parallel random search HPO experiments. Here we outline the training hyperparameters we used to perform random searches on, namely, epoch, depth, initial learing rate, maximum learning rate, and batch size. For each hyperparanmeter, a search boundary is defined to limit the scope of the search.

Chapter 5

Results

5.1 Training

This section presents all the figures and tables of metrics obtained during the **final**[1] training run of the machine learning models based on the **CASPYrus** datasets[2]. The configurations and partitions applied during training are shown in Table 4.3.5.1. This section presents graphic results of the prediction based on their learning tasks, namely, confusion matrices for the multiclass classification tasks and box plots for the regression tasks. In addition to the outcomes of the learning tasks, the metrics of the training objectives, as presented in Chapter 4.4.1, were collected and visualized throughout the training epochs. For each experiment, the graphs of the training results from both the **GNN** and **XGBoost** models will be shown first, followed by their corresponding learning curves reflecting the change in the values of the training objectives throughout the training.

5.1.1 Training results of Caspyrus 1k

The **CASPYrus 1k** partition is the smallest partition tested among all. The primary purpose of having such a partition is to make it a fail-fast, proof-of-concept experiment to verify the functionality of the machine learning and data processing pipelines. Given this configuration, 800 molecules were used to train the models, with 100 molecules reserved for the validatoin step and the last 100 serving as a test set. Note that all models from all partitions during experimentation went through self-testing and universal testing to ensure generalizability. Self-test uses the reserved 10% of the molecules in any partition, while the universal test utilizes the **ChEMBL 200k** data set that is introduced in detail in Chapter 4.3.2.

[1]Note that during the experimentation phase, a number of training and evaluation runs were performed for various troubleshooting reasons. All figures and tables presented in this section are obtained from the **final** run when reliable program execution was achieved from data parsing to result visualization. HPO experiments are carried out afterwards.

[2]Both models of multiclass classification and regression from the same size of the **CASPyrus** partition share the same training, validation, and test data sets for consistency. All trained models will be evaluated on the same **ChEMBL 200k** universal evaluation data set.

CASPYrus 1k multiclass

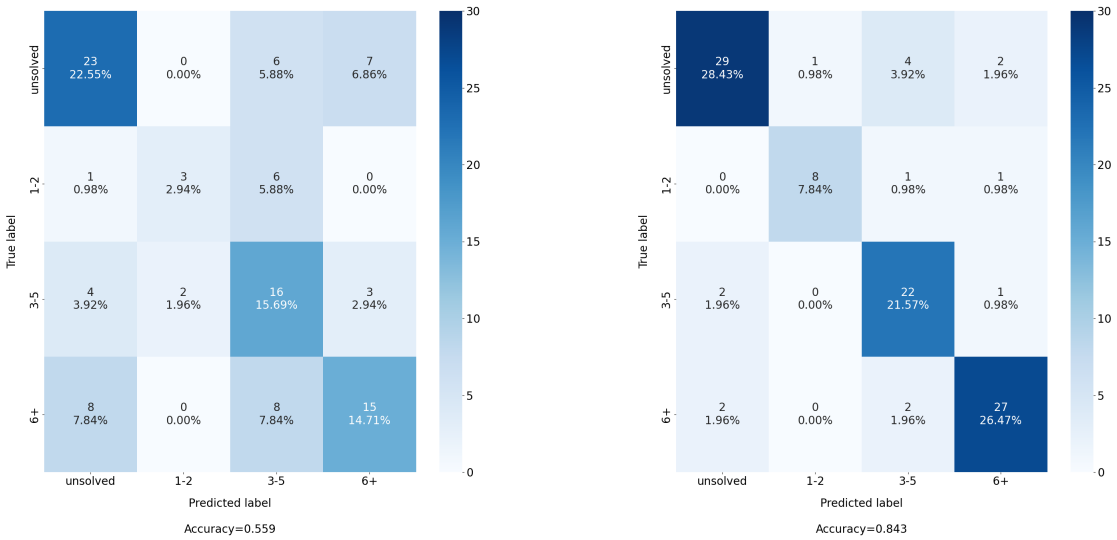

Figure 5.1.1.1: Confusion matrices of the *1k* multiclass Chemprop GNN (left) and XGBoost (right) models tested on **CASPYrus 1k** test set. These preliminary models, trained from 800 molecules and validated/tested by 100 molecules each, paved the way for further large-scale multiclass classification experiments.

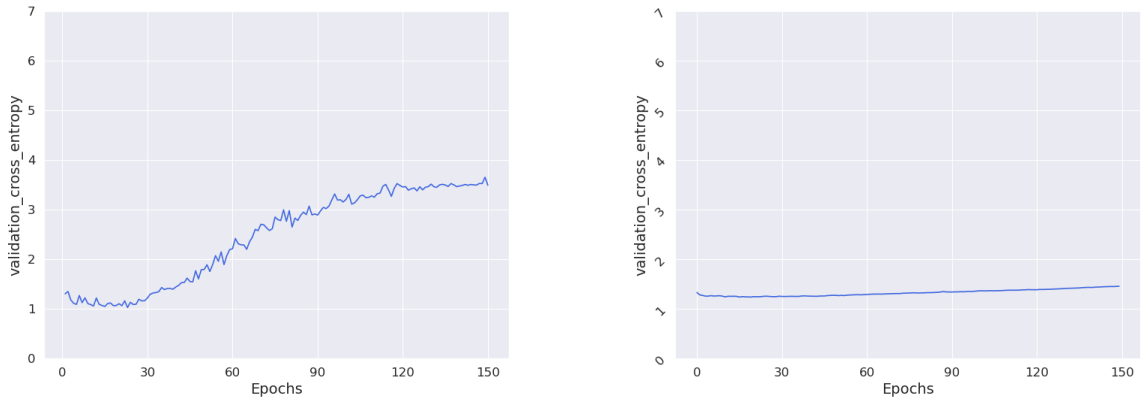

Figure 5.1.1.2: Validation cross-entropy vs. training epochs of **CASPYrus 1k** multiclass Chemprop GNN (left) and XGBoost (right) models validated on **CASPYrus 1k** validation set.

Metrics	GNN	XGBoost
Accuracy	0.559	0.843
F1	0.556	0.844
MCC	0.383	0.782
Precision	0.441	0.757
Recall	0.559	0.843

Table 5.1.1.1: Table of multiclass classification training metrics for **CASPYrus 1k** models

CASPYrus 1k regression

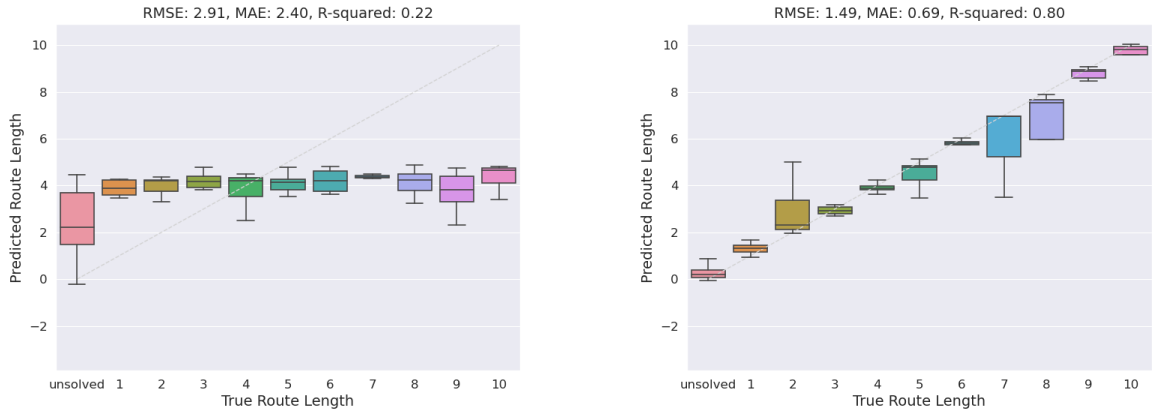

Figure 5.1.1.3: Box plots of the *1k* regression Chemprop GNN (left) and XGBoost (right) models tested on **CASPYrus 1k** test set, trained from 800 molecules and validated/tested by 100 molecules each. The dashed gray line indicates an ideal fit of route lengths predicted from SMILES.

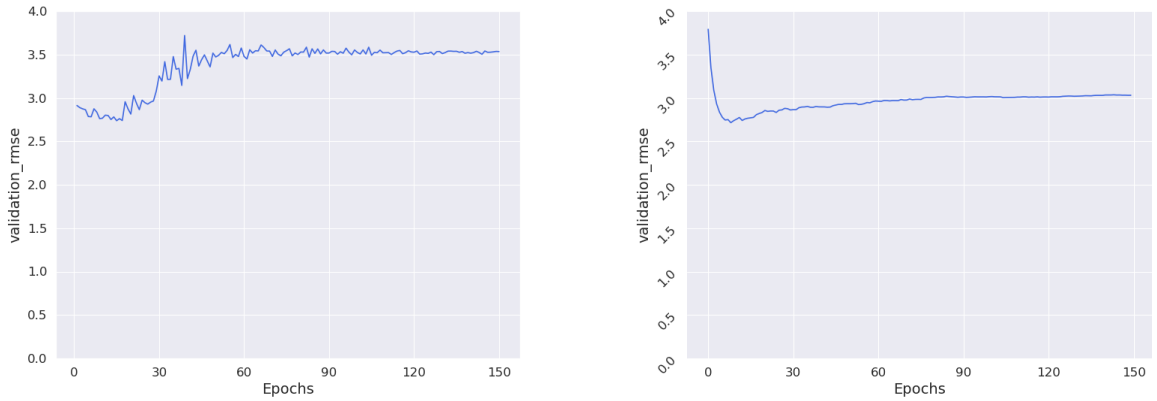

Figure 5.1.1.4: Validation RMSE vs. training epochs of **CASPYrus 1k** regression Chemprop GNN (left) and XGBoost (right) models validated on **CASPYrus 1k** validation set.

Metrics	GNN	XGBoost
RMSE	2.91	1.49
MAE	2.40	0.69
R-squared	0.22	0.80

Table 5.1.1.2: Table of regression training metrics for **CASPYrus 1k** models

The training results from **CASPYrus 1k** multiclass classification and regression models indicate that using machine learning to approximate the mapping from molecular structure to synthetic route lengths is feasible with multiclass classification tasks. In regression, only the **XGBoost** model seems to perform rather well after being trained on only 800 molecules. This, however, could also be an indicator of **XGBoost** models' vulnerability to overfitting.

5.1.2 Training results of Caspyrus 10k

After the experiments with the **CASPYrus 1k** partitions were completed, the feasibility of the concept was noticed, and a scaled-up experiment was planned as a follow-up. The result is the **Caspyrus 10k** partition introduced in Chapter 4.3.4. This partition has proven to be valuable as an in-between argument for the effectiveness of the experiment setup.

CASPYrus 10k multiclass

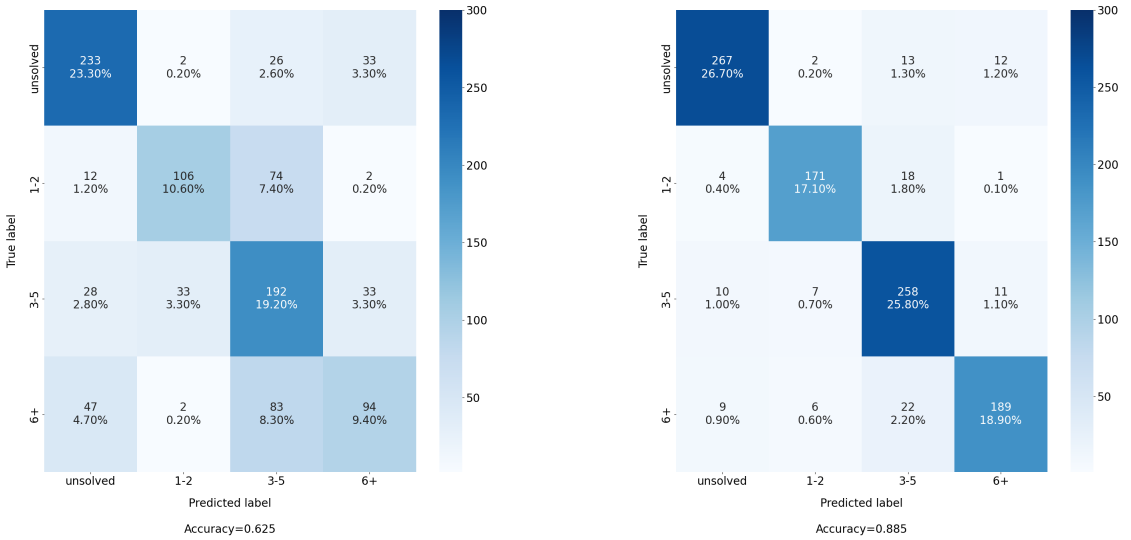

Figure 5.1.2.1: Confusion matrices of the *10k* multiclass Chemprop GNN (left) and XGBoost (right) models tested on **CASPYrus 10k** test set. These models serve as scaled-up multiclass classification from what's shown in Figure 5.1.1.1, are trained from 8,000 molecules, validated by 1,000, and tested by 1,000.

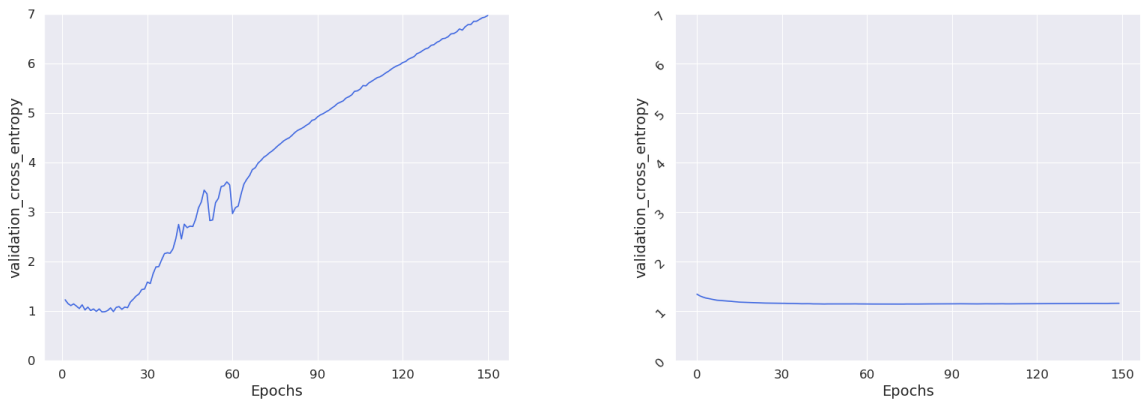

Figure 5.1.2.2: Validation cross-entropy vs. training epochs of **CASPYrus 10k** multiclass Chemprop GNN (left) and XGBoost (right) models validated on **CASPYrus 10k** validation set.

Metrics	GNN	XGBoost
Accuracy	0.625	0.885
F1	0.621	0.885
MCC	0.495	0.845
Precision	0.493	0.815
Recall	0.625	0.885

Table 5.1.2.1: Table of multiclass classification training metrics for **CASPYrus 10k** models

CASPYrus 10k regression

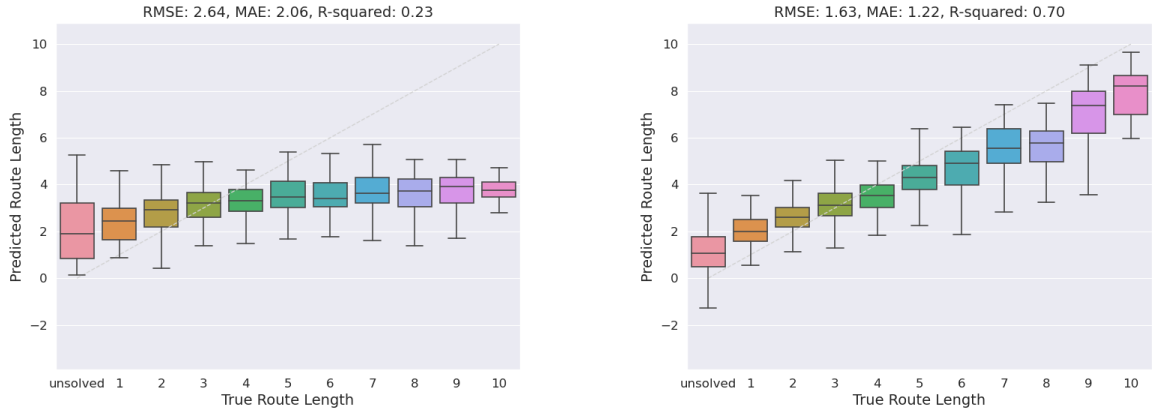

Figure 5.1.2.3: Box plots of the *10k* regression Chemprop GNN (left) and XGBoost (right) models tested on **CASPYrus 10k** test set, scaled up from Figure 5.1.1.3. This time the regression models are trained from 8,000 molecules, validated by 1,000, and tested by 1,000.

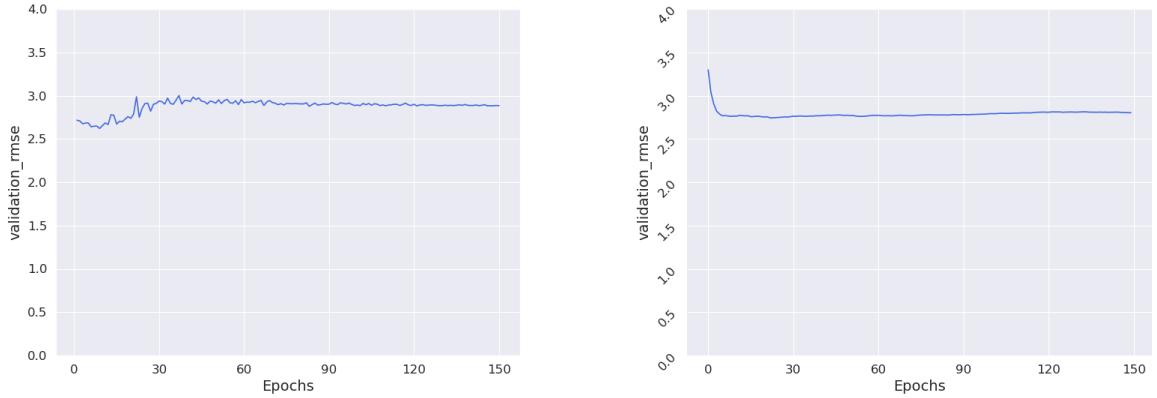

Figure 5.1.2.4: Validation RMSE vs. training epochs of **CASPYrus 10k** regression Chemprop GNN (left) and XGBoost (right) models validated on **CASPYrus 10k** validation set.

Metrics	GNN	XGBoost
RMSE	2.64	1.63
MAE	2.06	1.22
R-squared	0.23	0.70

Table 5.1.2.2: Table of regression training metrics for **CASPYrus 10k** models

From the results of **CASPYrus 10k** models, a positive effect is observable from scaled-up training with more data by improvement of training metrics. In multiclass classification, both models exhibit remarkably better accuracy, F1, and recall scores than those trained with **CASPYrus 1k** data set. As for regression, the GNN model in this partition is starting to recognize a pattern in the shorter route lengths, particularly from 1 to 4 (and approximating the inability for a CASP software to solve a molecule). The XGBoost models, both in multiclass classification and regression tasks, continue to yield remarkable training metrics, which will be later evaluated for their generalizability. It is noticeable that the learning objective of the GNN multiclass classification model starts to rise after being trained for around 25 epochs (see Figure 5.1.2.2). This is likely due to the limited diversity of data distribution on the training dataset **CASPYrus**, as the original data set *Papyrus* itself was a curated data set centralizing on bio-active molecules, as introduced in Chapter 4.3.1. This speculation will be discussed in detail in Chapter 6.6.

5.1.3 Training results of Caspyrus 50k

Here we present the result of the training based on the entirety of the **Caspyrus 50k** data set, as described in Table 4.3.5.1. This series of experiments marks the biggest and most sophisticated models trained during the experimentation phase of the project.

CASPYrus 50k multiclass

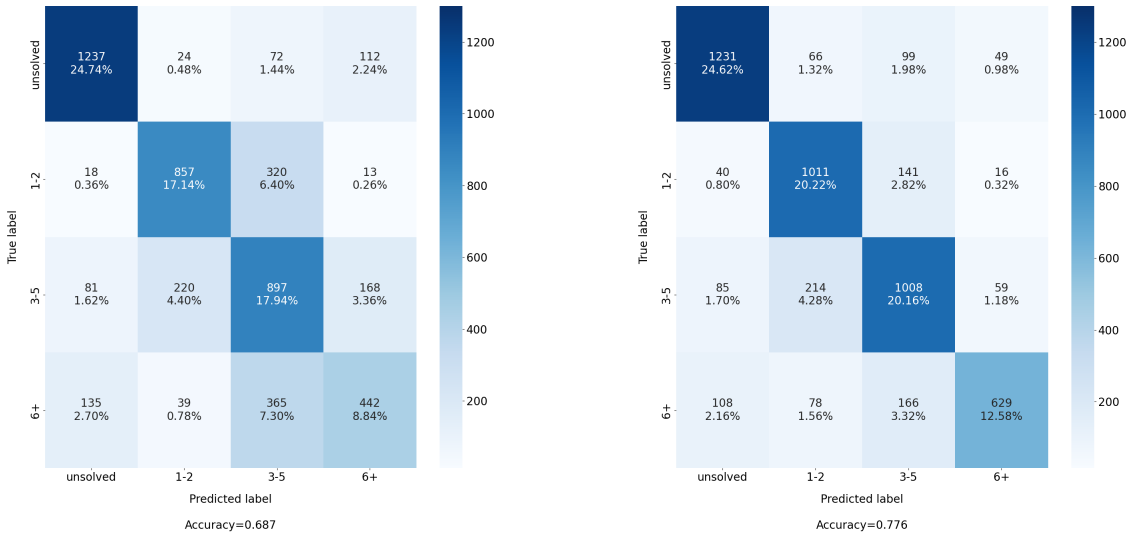

Figure 5.1.3.1: Confusion matrices of the *50k* multiclass Chemprop GNN (left) and XGBoost (right) models tested on **CASPYrus 50k** test set. These multiclass classification models were trained, validated, and tested utilizing the full **CASPYrus 50k** data set with 40,000 molecules for training and 10,000 each for validation and testing.

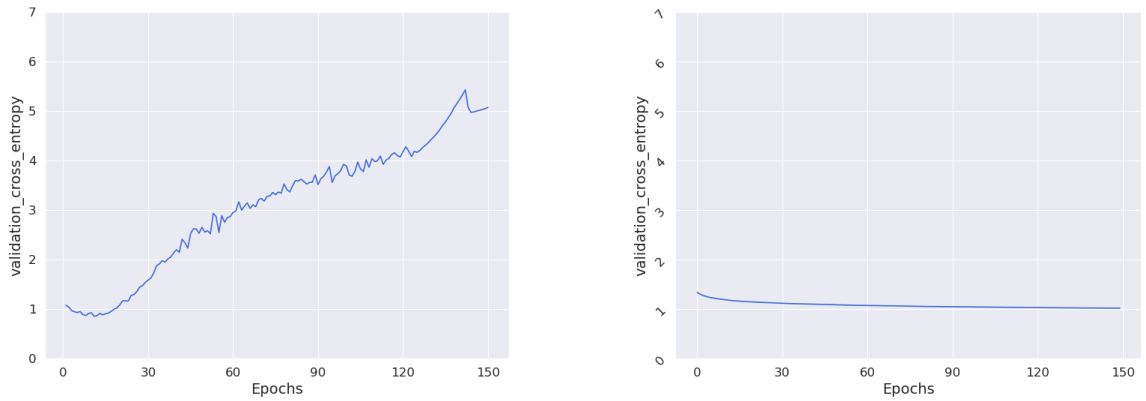

Figure 5.1.3.2: Validation cross-entropy vs. training epochs of **CASPYrus 50k** multiclass Chemprop GNN (left) and XGBoost (right) models validated on **CASPYrus 50k** validation set.

Metrics	GNN	XGBoost
Accuracy	0.687	0.776
F1	0.685	0.775
MCC	0.579	0.700
Precision	0.563	0.668
Recall	0.687	0.776

Table 5.1.3.1: Table of multiclass classification training metrics for *50k* models

CASPYrus 50k regression

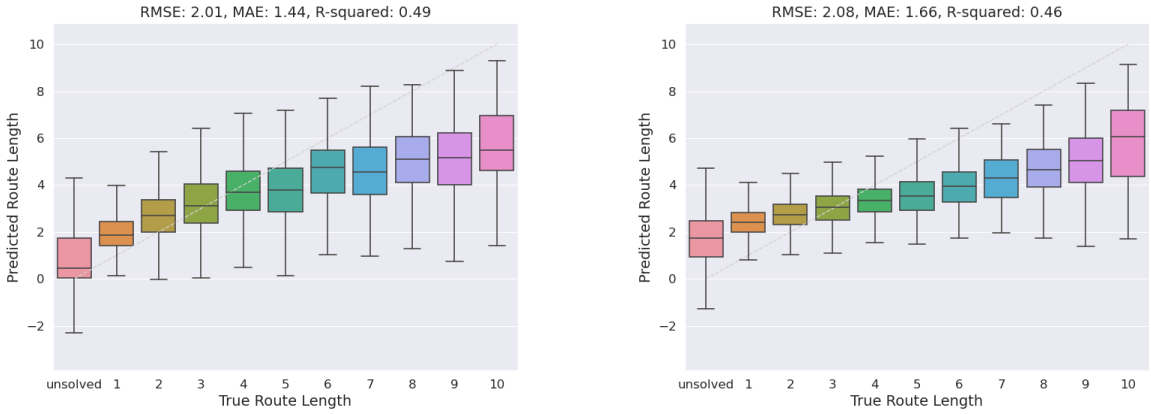

Figure 5.1.3.3: Box plots of the *50k* regression Chemprop GNN (left) and XGBoost (right) models tested on **CASPYrus 50k** test set. These regression models were trained, validated, and tested utilizing the full **CASPYrus 50k** data set with 40,000 molecules for training and 10,000 each for validation and testing.

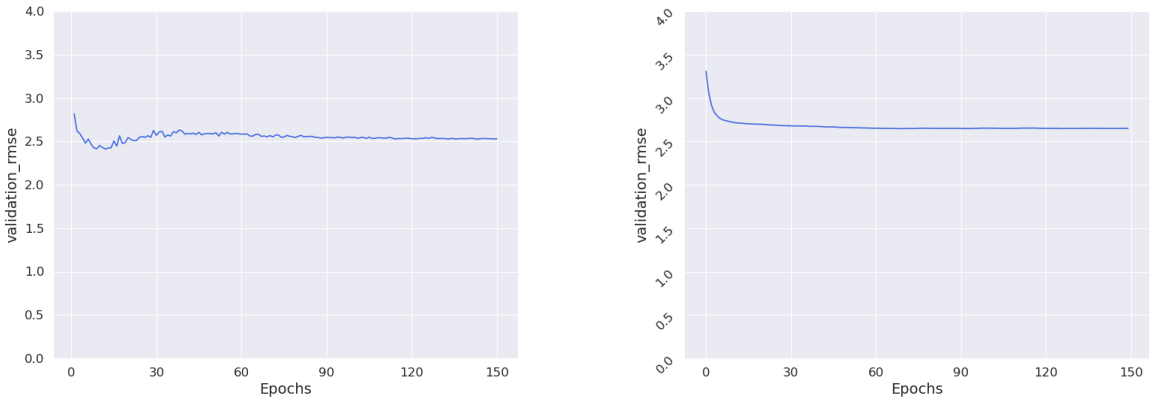

Figure 5.1.3.4: Validation RMSE vs. training epochs of **CASPYrus 50k** regression Chemprop GNN (left) and XGBoost (right) models validated on **CASPYrus 50k** validation set.

Metrics	GNN	XGBoost
RMSE	2.01	2.08
MAE	1.44	1.66
R-squared	0.49	0.46

Table 5.1.3.2: Table of regression training metrics for *50k* models

Training

Results in this partition demonstrate the best training performance we achieved utilizing the **CASPYrus 50k** data set. In multiclass classification tasks, both models now demonstrate good training metrics with F1 scores of 0.685 and 0.775. In regression tasks, the GNN model begins to outperform the **XGBoost** model by achieving a 0.07 lower RMSE and a 0.03 higher R-squared value than the XGBoost model.

5.2 Evaluation

This section details the generalizability experiments, namely, evaluations, for all **CASPYrus** models trained during the final training run, as described in Chapter 5.1. As the models have not been trained with the **ChEMBL 200k** data [64], and due to the amount of unseen molecules, the evaluation results aim to provide an argument for the generalizability of the predictive models.

In this section, the results are presented with respect to the size of the experiment partition – *1k, 10k* and *50k*. For each experiment, multiclass classification results from the **GNN** and **XGBoost** models will be presented first, followed by regression results. Since the evaluation procedures did not involve the training of new models, this section will not include learning curves as shown in Chapter 5.1.

5.2.1 CASPYrus 1k models evaluated on ChEMBL 200k evaluation set

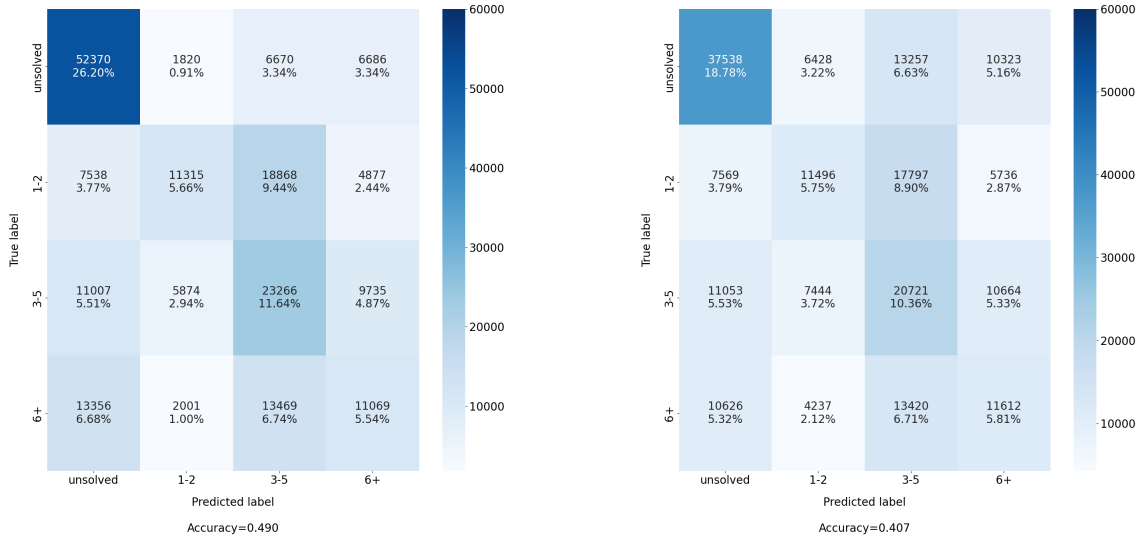

Figure 5.2.1.1: Confusion matrices of **CASPYrus 1k** multiclass models (trained from 800 molecules) from Chemprop GNN (left) and XGBoost (right) evaluated on **ChEMBL 200k** universal evaluation data set to verify the models' generalizability.

Metrics	GNN	XGBoost
Accuracy	0.490	0.407
F1	0.474	0.406
MCC	0.302	0.196
Precision	0.377	0.340
Recall	0.490	0.407

Table 5.2.1.1: Table of multiclass classification evaluation metrics for **CASPYrus 1k** models on ChEMBL 200k evaluation set.

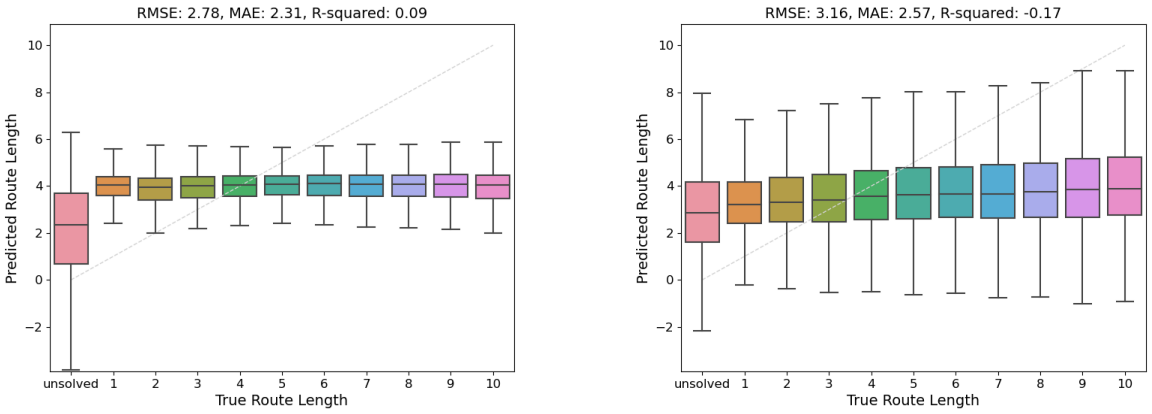

Figure 5.2.1.2: Box plots of **CASPYrus 1k** regression models (trained from 800 molecules) from Chemprop GNN (left) and XGBoost (right) evaluated on **ChEMBL 200k** universal evaluation data set to verify the models' generalizability.

Metrics	GNN	XGBoost
RMSE	2.78	3.16
MAE	2.31	2.57
R-squared	0.09	-0.17

Table 5.2.1.2: Table of regression evaluation metrics for **CASPYrus 1k** models on ChEMBL 200k evaluation set

Generalizability of the **CASPYrus 1k** models on the **ChEMBL 200k** data set generally did not yield promising results in either classification or regression tasks. Nonetheless, in confusion matrices shown in Figure 5.2.1.1, both GNN and XGBoost models could recognize the unsolvable class with a rather clear distinction from the rest. Besides, in both models, the true positive prediction of the route length class $3-5$ seems to stand out among all predicted classes.

5.2.2 CASPYrus 10k models evaluated on ChEMBL 200k evaluation set

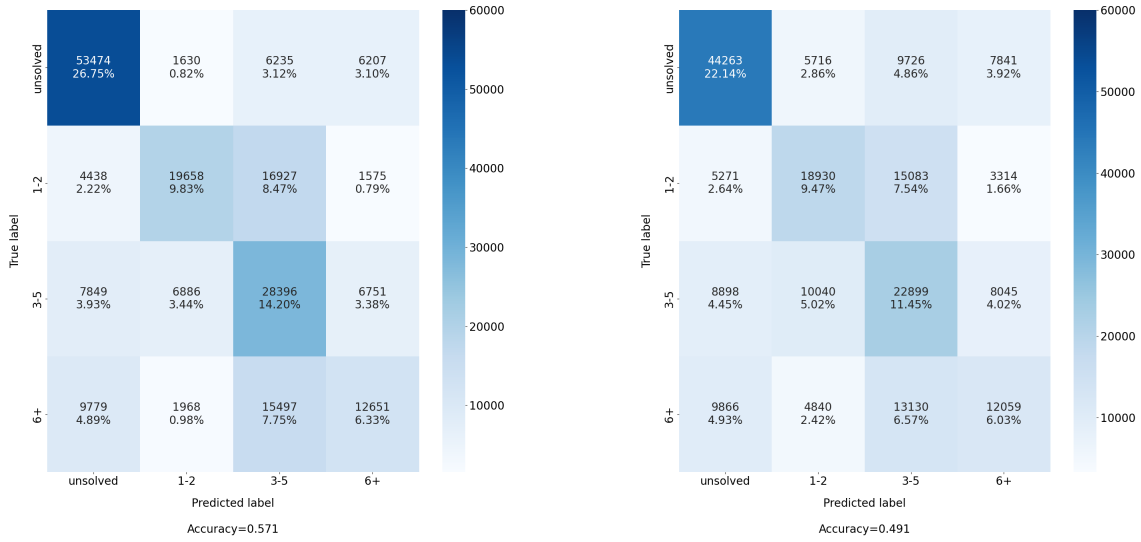

Figure 5.2.2.1: Confusion matrices of **CASPYrus 10k** multiclass models (trained from 8,000 molecules) from Chemprop GNN (left) and XGBoost (right) evaluated on **ChEMBL 200k** universal evaluation data set to verify the models' generalizability.

Metrics	GNN	XGBoost
Accuracy	0.571	0.491
F1	0.564	0.489
MCC	0.417	0.309
Precision	0.445	0.392
Recall	0.571	0.491

Table 5.2.2.1: Table of multiclass classification evaluation metrics for **CASPYrus 10k** models on ChEMBL 200k evaluation set

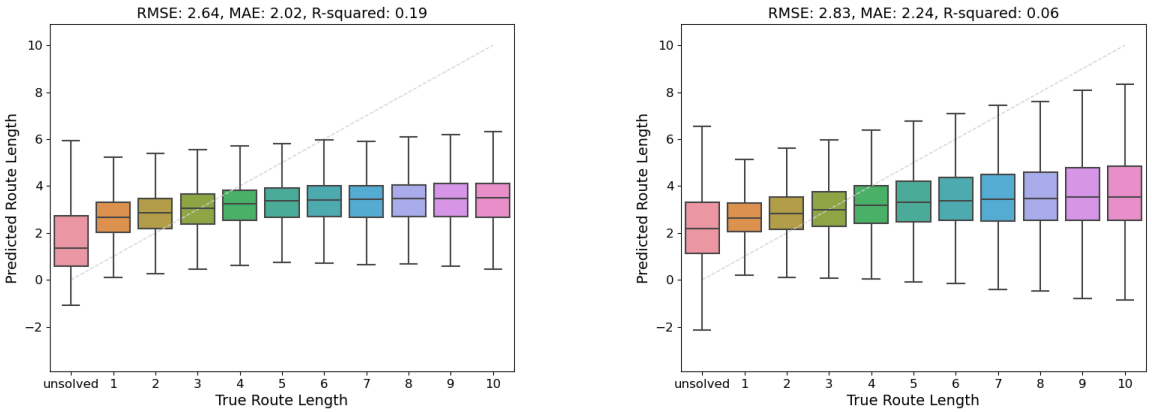

Figure 5.2.2.2: Box plots of **CASPYrus 10k** regression models (trained from 8,000 molecules) from Chemprop GNN (left) and XGBoost (right) evaluated on **ChEMBL 200k** universal evaluation data set to verify the models' generalizability.

Metrics	GNN	XGBoost
RMSE	2.64	2.83
MAE	2.02	2.24
R-squared	0.19	0.06

Table 5.2.2.2: Table of regression evaluation metrics for **CASPYrus 10k** models on ChEMBL 200k evaluation set

Following the findings from the generalizability evaluation experiments of the **CASPYrus 1k** models on the **ChEMBL 200k** data set, the pattern that the models are only capable of correctly identifying unsolvable molecules among all molecules evaluated continues to be observable from the results above. The **CASPYrus 10k** GNN multiclass classification model outperforms the **CASPYrus 10k** XGBoost multiclass classification model with a 16.3% higher classification accuracy. Nonetheless, in terms of regression, neither models appear to be able to effectively generalize their learned mapping between the molecular structure and synthetic route lengths from the training sets of **CASPYrus 10k** partition with only slightly improved regression RMSE.

5.2.3 CASPYrus *50k* models evaluated on ChEMBL 200k evaluation set

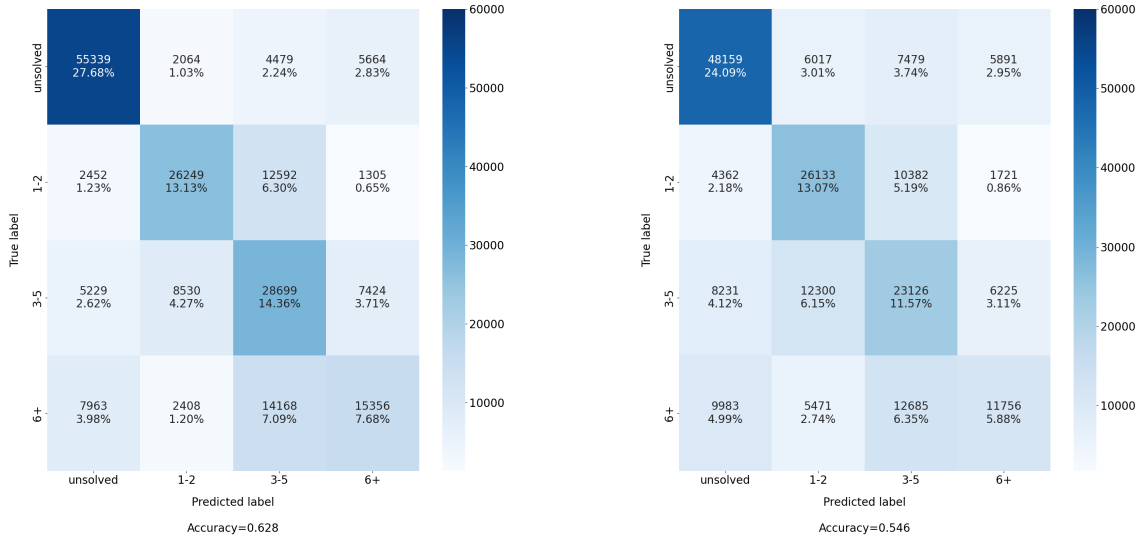

Figure 5.2.3.1: Confusion matrices of *50k* multiclass models (trained from 40,000 molecules) from Chemprop GNN (left) and XGBoost (right) evaluated on **ChEMBL 200k** universal evaluation data set to verify the models' generalizability.

Metrics	GNN	XGBoost
Accuracy	0.628	0.546
F1	0.625	0.539
MCC	0.496	0.384
Precision	0.501	0.442
Recall	0.628	0.546

Table 5.2.3.1: Table of multiclass classification evaluation metrics for *50k* models on ChEMBL 200k evaluation set

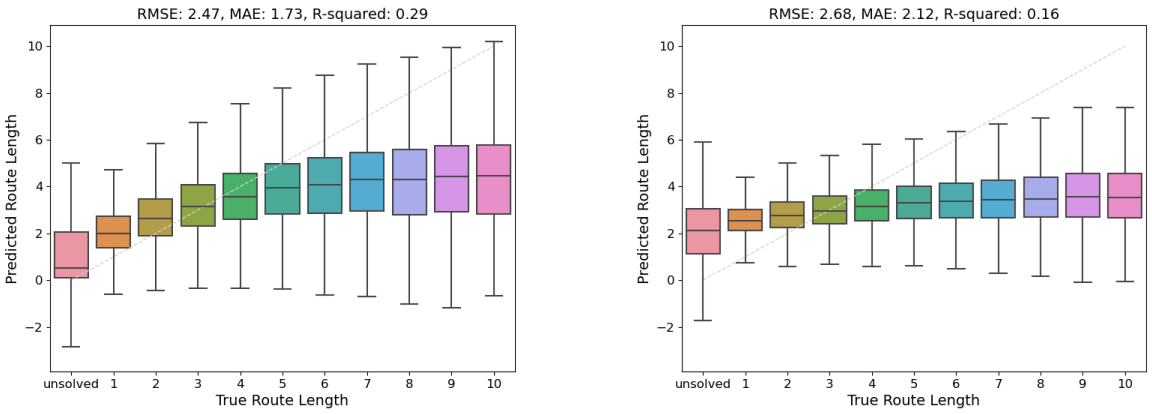

Figure 5.2.3.2: Box plots of *50k* regression models (trained from 40,000 molecules) from Chemprop GNN (left) and XGBoost (right) evaluated on **ChEMBL 200k** universal evaluation data set to verify the models' generalizability.

Metrics	GNN	XGBoost
RMSE	2.47	2.68
MAE	1.73	2.12
R-squared	0.29	0.16

Table 5.2.3.2: Table of regression evaluation metrics for *50k* models on ChEMBL 200k evaluation set

The **CASPYrus 50k** models demonstrate noticeably better generalizability on the **ChEMBL 200k** data set than the models trained on two other smaller partitions. From the confusion matrices, patterns are clearly visible on the diagonal on both models (see Figure 5.2.3.1). The GNN models continue to deliver better generalizability evaluation performance compared to the XGBoost models in both multiclass classification and regression tasks. Regarding regression tasks, in contrast to the previous two generalizability evaluations, the GNN model successfully aligned its prediction with the ideal fitting line in shorter route lengths $1-4$, along with the correctly identified group of unsolvable molecules. The XGBoost regression model, however, was not able to generalize from the **CASPYrus 50k** training set.

5.3 Hyperparameter Optimization

The figures below present the result overlay of 100 individual random hyperparameter optimization searches for both the *50k* GNN multiclass classification and regression models[3].

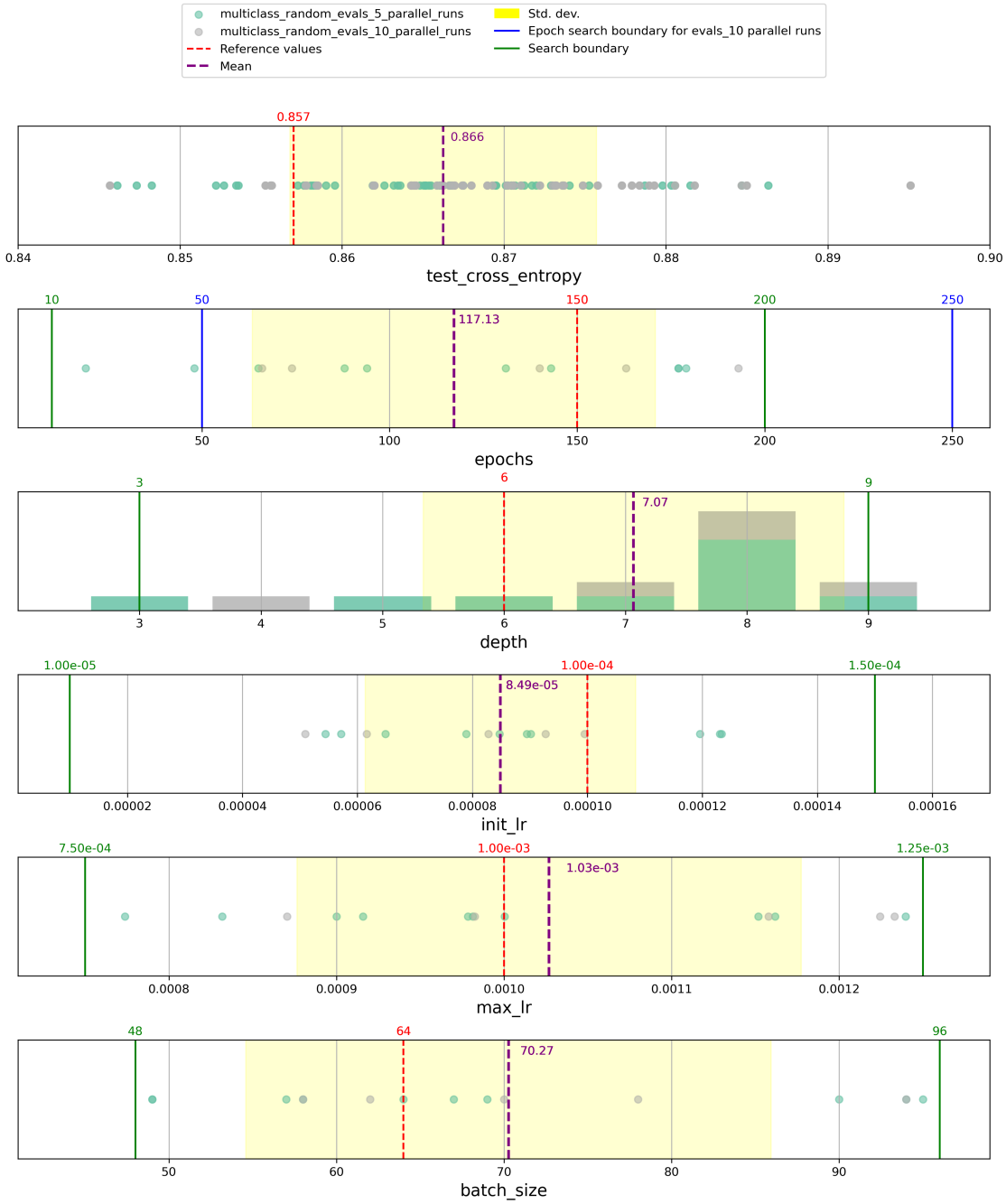

Figure 5.3.0.1: Distribution of hyperparameter optimization (random search) results for the GNN *50k* multiclass model. Note that the purple, dashed lines stand for the mean performance of each metric, and the light yellow-colored regions denote the distribution of 1 *Std.dev* of each metric from their mean values. Data points colored in green and grey indicate different runs the results were obtained from, as documented in Chapter 4.7. The epoch boundaries in blue serve only for the experiment carried out in 10 parallel runs, as detailed in the legend.

[3]The 100 random searches consist of 50 runs each from two different configurations of embarrassingly parallel experiments.

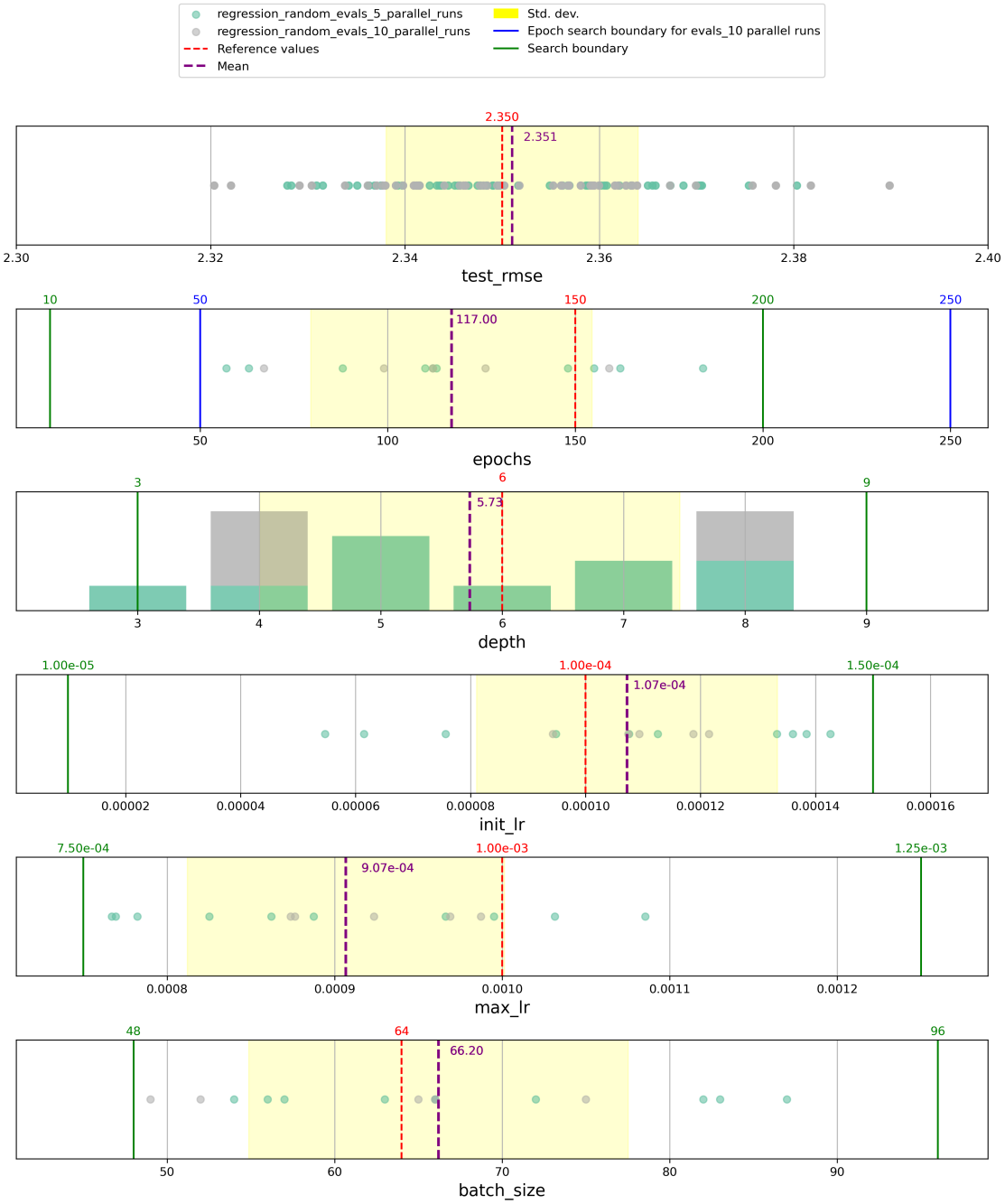

Figure 5.3.0.2: Distribution of hyperparameter optimization (random search) results for the GNN *50k* regression model. Interpretation of visual elements has been documented in the caption of Figure 5.3.0.1. Together with Figure 5.3.0.1, these figures demonstrate the distribution of our hyperparameters during the HPO experiments, which further reinforced the robustness of our initial and chosen values.

In Figures 5.3.0.1 and 5.3.0.2, the top rows reflect the distribution of test metrics (RMSE and cross-entropy) collected from all individual random search trainings, followed by distributions of the five hyperparameters selected to perform hyperparameter optimization: training epochs, network depth, initial learning rate, maximum learning rate, and batch size. The dashed red lines marked on each row denote the default or initially chosen values of all metrics and hyperparameters (for metrics, the default values refer to the results obtained during the final training run). The distributions of $Mean \pm 1 Std.dev$ are colored in light yellow.

Chapter 6

Discussion

In this chapter, various dimensions of arguments will be raised and discussed regarding the results obtained and their indications. The aspects of learning methods, machine learning models, training data, generalizability, training parameters, and limitations will be discussed in detail.

6.1 Learning methods

To begin with, we analyze the results based on the machine learning methods, namely, the comparison of multiclass classification and regression methods for their effectiveness in predicting synthetic route lengths given an input of molecular structure[1].

6.1.1 Multiclass classification

Overall, the classification tasks has proven their effectiveness at predicting the synthetic route length groups. From the training results of the **CASPYrus 50k** models in Chapter 5.1.3, the **CASPYrus 50k** multiclass classification models were able to achieve more than 0.500 classification accuracy and F1 scores. Noticeably, the *50k* GNN model has accomplished a classification accuracy of 0.687 and a F1 score of 0.685 from the 5,000 **CASPYrus** molecules training set. The evaluation results, as detailed in Chapter 5.2.3, further demonstrated the model's generalizability potentials. With a classification accuracy of 0.628 and an F1 score of 0.625 from inference on the **ChEMBL 200k** dataset, where the model was evaluated on 200,000 unseen molecules from the **ChEMBL** database (introduced in Chapter 4.3.2). Moreover, the confusion matrices indicate that the multiclass classification models are capable of identifying the underlying pattern that maps from the molecular structures to the synthetic route length classes.

Based on the results, one can argue that the multiclass classification tasks successfully demonstrated their potentials at mapping the underlying patterns from the molecular structures to synthetic route lengths. However, the learning curves of the multiclass classification tasks indicate that some models overfit. Starting with the validation cross-entropy of the *1k* GNN model as shown in Figure 5.1.1.2, the value of the learning objective reduced and converged to near 1.1 for the first 20 epochs but increased from there on, reaching near 3.50 at the end of the 150 epochs of training. Such a pattern on a greater scale could also be observed in the validation cross-entropy learning curve of the *10k* GNN model shown in Figure 5.1.2.2 with the learning

[1]For XGBoost models, the input data is in the format of the molecular fingerprint **ECFP**, as detailed in Chapter 2.6.4. The Chemprop GNN models extract features from **SMILES** automatically

objective. The *50k* reaching near 7.0 at the end. *50k* GNN model's validation cross-entropy experienced a less radical change in value than that from the *10k* model, as shown in Figure 5.1.3.2, but still reaches near 5.0 in the end. As this finding occurs primarily on the GNN models, it will be discussed more in detail in Chapter 6.2 where the models themselves will be analyzed.

6.1.2 Regression

Compared to the multiclass classification models, the regression models tend to generate less expressive predictions overall, except for the shorter routes on the **CASPYrus 50k** regression models. Box plots for the generalizability evaluations displayed in Chapter 5.2 demonstrated such observations. Only Figure 5.1.3.3 has shown the *50k* GNN model's potential to predict route lengths from "unsolvable" up to 4, which none of the other **CASPYrus** regression models have yet been able to demonstrate. The analysis for this observation is that the underlying relationship between the molecular structure denoted in **SMILES** and the discrete route lengths in numerical values may not be so expressive, which is especially the case for all **XGBoost** models, as the evaluation box plots barely show any patterns that could be found in the generalizability experiment (see Figures 5.2.1.2, 5.2.2.2, and 5.2.3.2). Training box plots, however, seem to indicate that models tend to overfit when using regression, as the "patterns" learned could not be reflected in the latter generalizability evaluations (see Figures 5.1.1.3, 5.1.2.3, and 5.1.3.3).

Despite the fact that the regression tasks did not excel in learning the underlying pattern between molecular structures and discrete route lengths in most cases, the **CASPYrus 50k** GNN regression model still demonstrated its capability to generalize its learned pattern to some extent. From route lengths 1 to 4, including unsolved molecules, the model's prediction roughly corresponds to the diagonal line in Figure 5.2.3.2, which indicates that the regression approach might be effective when, first, the model must be trained comprehensively with a large amount of data (*50k* in this case), and second, the predictive capabilities using regression have to be limited to the shorter route lengths mentioned previously. This is likely due to the abundance of the route lengths of 1 to 4 that exist in the training set, which led to a more comprehensive training coverage for the molecules with such synthetic route lengths. A better generalizability as a result of a larger training data set may also contribute to this finding, which will be discussed in detail in Chapter 6.4.

6.1.3 Summary of machine learning methods

In general, the regression method experimented with during this project did not prove its effectiveness at answering the research question but served as a valuable attempt to outline what may be needed to increase its predictive performance for future experiments. Multiclass classification models, in contrast, have demonstrated decent performance and generalizability in classifying synthetic route lengths based on molecular structure, thus being the recommended formulation for such machine learning tasks.

6.2 Machine learning models

Following the comparison of learning paradigms, this section continues to outline the comparison between the two machine learning models experimented with during this project. A brief comparison of the characteristics of the two types of machine learning models will be conducted, followed by a detailed evaluation of their performances for the machine learning tasks.

6.2.1 GNN models vs. XGBoost models

Chapter 2.3.1 and Chapter 2.3.2 introduced the two algorithms in detail. From the training and generalizability evaluation results demonstrated in Chapter 5, models trained on larger **CASPYrus** partitions via GNN exhibit in general better training and generalizability metrics. Such findings can be observed on Tables 5.2.3.1 and 5.1.3.1. One may wonder if the intrinsic difference between the two algorithms could lead to such performance differences. The table below incorporates the architectural characteristics of the two algorithms with respect to this project setting.

	Chemprop GNN models	XGBoost models
Preferred data type	Graph-like data such as molecular structures	Structured, independent data, such as tables
Noise Robustness	Sensitive to noisy edges/features	Handles noisy features via regularization
Multiclass model performance	Message passing captures topological patterns in graphs but requires hyperparameter tuning for optimal performance	Native Softmax objective support handles high-dimensional data efficiently but struggles with implicit relationships in graph formats
Regression model performance	Captures topological relationships through message passing. Effective for graph regression but requires edge noise handling for optimal performance	Built-in support for capturing non-linear patterns but struggles with graph-structured data dependencies
Feature engineering	Needs graph topology preprocessing	Automatic feature interaction detection
Regularization	Requires explicit techniques like dropout	Built-in (L1/L2, column subsampling)

Table 6.2.1.1: Comparison of GNN and XGBoost Models [50, 27, 9], detailing characteristics exhibited by machine learning models trained via these two algorithms in the scope of this project.

With that comparison, it is easily observed that models trained via GNN in this project have an advantage due to its affinity to graph-like data structures, such as molecular graphs. An additional support to this finding is the 2021 study of Jiang et al. that further demonstrated the performance advantage of GNN models and their variants in learning intricate relationships between structures and properties [13].

From the experimental results, such observations are consolidated by the fact that on all regression tasks, GNN outperforms XGBoost by exhibiting lower test and evaluation RMSE values. Figure 5.2.3.2 also demonstrates a better generalizability of the GNN *50k* regression model when evaluated on the **ChEMBL 200k** evaluation set, exhibiting an R-squared value that surpasses its XGBoost counterpart by 81.25%. Such performance advantages also hold true in the multiclass classification tasks with a margin of 15.02%.

Interestingly, if we focus on the *1k* results of GNN and XGBoost, it is noticed that the XGBoost model is overfitting with a small amount of training data on both multiclass classification and regression tasks (see Figures 5.1.1.1 and 5.1.1.3). This demonstrated that XGBoost is more sensitive to smaller training data set as compared to GNN. This may also be due to the fact that, with no explicit specification of data structures, the XGBoost model was initially treating the molecular fingerprints as structured, table-like inputs and learned incorrect, ungeneralizable features, as demonstrated by the **ChEMBL 200k** evaluation results. The bigger the training data becomes, the less unstable XGBoost models perform, though it rarely surpasses the performances that the GNN models offer, either during training or evaluation.

6.3 Training data

In this section, we aim to post-hoc uncover the adequate amount of data needed to carry out the machine learning task of estimating synthetic route lengths from molecular structures. Intuitively, the more training data is available, the better (and more generalizable) the models will be trained, assuming similar data distribution. In reality, such ideal scenarios of datasets with appropriate quality of data distribution are often major bottlenecks in the realm of machine learning, which must be taken into consideration. Thus, it necessitates the finding of a proper amount of training data that offers enough diversity and depth for the model to learn generalizable patterns.

As for this project, the amount of the training data has always been a major challenge. Despite that, the early experiments of *1k* and *10k* partitions have proven the feasibility of the research question: predicting the route lengths given the molecular structures. Starting with the *1k* models, even though the XGBoost models significantly overfit on the **CASPYrus** dataset, the learned '`unsolved`' class and the lengths '`3-5`' class in the multiclass classification task are still somewhat generalizable on the **ChEMBL 200k** data set, as shown in Figure 5.2.1.1. GNN offered similar performance to XGBoost in the *1k* partition as well. Thus, the *1k* partition is sufficient as a starting point, with 800 molecules being used to train the models and still being able to achieve a clear classification of the "unsolvable" class. As a result, the *1k* partition could potentially offer acceptable performance on binary synthesizability prediction similar to the *RAScore* by Thakker and Amol et al. [59], despite its rather poor performance on multiclass synthetic route length group predictions.

The *10k* partition, as introduced in Chapter 4.3.4, serves as an in-between argument from *1k* to *50k*. In the context of this section, the *10k* partition serves the purpose of demonstrating the effect of upscaling the magnitude of the amount of training data. The results align with our initial intuition, especially among multiclass classification models, as shown in Figure 5.2.2.1. Even though the regression results do not offer strong interpretability, comparing Figure 5.2.2.2 to Figure 5.2.1.2, the improvements in RMSE, MAE, and R-squared values and the outlook of the plot are visible when upscaling the amount of the training data.

The *50k* partition primarily serves the purpose of testing the best performance of all models when utilizing the **CASPYrus 50k** training data set to the maximum extent. The multiclass classification evaluation results obtained are significantly better than those for the *10k* partition, which could be interpreted by comparing Table 5.2.3.1 with Table 5.2.2.1. However, the *50k* regression tasks do not provide such insights, except for the smaller route length classes described in Chapter 6.1.2. Regardless, from the generalizability evaluation metrics, the results obtained from the *50k* partition surpass the ones from *10k* in **ChEMBL 200k** evaluations by considerable margins, as shown in Table 6.3.0.1.

Metric	Model	10k Value	50k Value	Percent Difference (%)
RMSE	GNN	2.64	2.47	-6.44
	XGBoost	2.83	2.68	-5.30
MAE	GNN	2.02	1.73	-14.36
	XGBoost	2.24	2.12	-5.36
R-squared	GNN	0.19	0.29	+52.63
	XGBoost	0.06	0.16	+166.67

Table 6.3.0.1: Percent difference of the generalizability evaluation metrics as a result of scaled-up training (*50k* vs. *10k* regression experiments). The effect of scaled-up training is visible from the increase in the values of evaluation metrics from a smaller portion of training data to a bigger one.

As we have concluded that the results from the regression tasks do not express enough predictive performance, it would make sense to use the results from the multiclass classification experiments as a deterministic factor to decide what can be a minimal adequate quantity of training data that is necessary for the machine learning task. If we look at the GNN models' evaluation accuracies, from 0.490 for *1k* partition to 0.571 for *10k*, to eventually 0.628 for *50k*, we can notice that the increase in accuracy is not linear with the growth in model size but rather somewhat logarithmic. That indicates a limit of such accuracy growth in between our best experimental value 0.628 and the theoretical perfect performance of 1.000. Detailed mathematical modeling of such a relationship is out of the scope of this project, but an educated speculation could be $100,000$ training molecules to reach a multiclass classification accuracy near 0.700[2].

Meanwhile, the classification accuracy 0.628 achieved by *50k* partition is still relatively acceptable, as the next steps of this project, if moving on towards production, would involve fine-tuning the models to provide insights to the lab chemists. A likelihood of whether or not a molecule could be synthesized within certain steps (or not possible at all) of chemical reactions with a prediction accuracy of around $60\% - 70\%$ can be seen as helpful. Aside from evaluation accuracy, training costs should also be taken into account. Therefore, a key takeaway on such an aspect is that *50k* molecules as a starting point is satisfying the needs for this project.

6.4 Generalizability

In this section, we aim to answer the question, "How well do scores generalize to a different data distribution?" As Hastie et al. explained in Chapter 7 of their 2009 book *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, the amount of training data plays a vital role in the model's generalizability [26]. Thus, we could look into our biggest and most performing models, the *50k multiclass*, and compare its performance metrics during training and evaluation. The table below gathered training and evaluation metrics for both GNN and XGBoost models for their generalizability analysis.

Metric	GNN		XGBoost	
	Training	Evaluation	Training	Evaluation
Accuracy	0.687	0.628	0.776	0.546
F1 Score	0.685	0.625	0.775	0.539
MCC	0.579	0.496	0.700	0.384
Precision	0.563	0.501	0.668	0.442
Recall	0.687	0.628	0.776	0.546

Table 6.4.0.1: Generalizability evaluation metrics of *50k* GNN vs. XGBoost multiclass models on the **ChEMBL200k** data set. Generally speaking, the less the drop in metric values from training to evaluation, the better the generalization capability the model has. Note that the `Accuracy` and `Recall` scores appear identical in this table. This is likely due to the default adoption of the *micro-averaged recall* in the Python package *Scikit-learn*, which leads to a mathematically identical calculation of `Recall` as to `accuracy` scores in multiclass classification [42, 55].

From Table 6.4.0.1, GNN exhibits a performance deviation between training and evaluation ranging from 8.59% to 14.34%, with the smallest drop in Accuracy and Recall (8.59%) and the largest in MCC (14.34%). In contrast, XGBoost shows a significantly larger deviation, ranging from 29.64% to 45.14%, with the smallest drop in Accuracy and Recall (29.64%) and the largest in MCC (45.14%). Both models exhibit the best metric preservation in precision where GNN experienced an 11.01% drop and XGBoost a 33.83%

[2]This prediction remains as an approximation to a CASP software's solution of synthetic route lengths to a target molecule.

drop. Arguably, our multiclass classification models generalize well, with GNN demonstrating more stable performance compared to XGBoost.

Metric	GNN		XGBoost	
	Training	Evaluation	Training	Evaluation
RMSE	2.01	2.47	2.08	2.68
MAE	1.44	1.73	1.66	2.12
R^2	0.49	0.29	0.46	0.16

Table 6.4.0.2: *50k* GNN vs. XGBoost regression generalizability comparison on the **ChEMBL200k** data set, similar to the comparison as detailed in Table 6.4.0.1.

From Table 6.4.0.2, GNN exhibits a relative performance deviation between training and evaluation with an RMSE increase of 22.89%, an MAE increase of 20.14%, and an R^2 drop of 40.82%. In contrast, XGBoost shows a larger deviation, with an RMSE increase of 28.85%, an MAE increase of 27.71%, and a significant R^2 drop of 65.22%. As the regression models were not highly performant during training, their rather low generalizability is expected, with GNN demonstrating better generalization compared to XGBoost.

Overall, with better generalizability evaluation metrics, we conclude that for the synthetic route length prediction tasks, the GNN models have an advantage over XGboost models on generalizability of learned mappings from molecular structures to synthetic route lengths.

6.5 Training parameters

In this section, we discuss whether or not the chosen and default hyperparameters make sense, or, in other words, are within a certain range that assures the model's optimal performance. Figures 5.3.0.1 and 5.3.0.2 presented the result overlay of 100 individual random hyperparameter optimization searches for both the 50k GNN multiclass classification and regression models[3]. On the graphs, for each parameter, the optimal parameters found along with their mean values and standard deviations are plotted. As explained in Chapter 4.7, the hyperparameter optimization experiments conducted in this project consist of multiple individual, independent runs, thus leading to multiple sets of optimal parameters. Therefore, to visualize all the results in a meaningful way, distributions of the parameters were made to better scrutinize the effectiveness of our chosen parameters that were used to train the models. Evidently, our chosen parameters (as stated in Table 4.5.1.1) are all within the range of $\pm 1 Std.dev$ from the mean optimal values, obtained from 100 hyperparameter optimization runs in total[4]. The following tables demonstrate the chosen values, the mean values, and the percent difference for each parameter:

[3]The 100 random searches consist of 50 runs each from two different *embarrassingly parallel* configurations. One of them has 5 parallel processes of 10 random searches, while the other consists of 10 parallel processes of 5 random searches, as detailed in Table 4.7.0.1

[4]Data obtained in the hyperparameter optimization experiments serve for the arguments in this project only, as generalizing one specific set of hyperparameters from the designated scenario to another rarely yields positive impacts on training results.

Hyperparameters	Chosen values	Mean optimal values	Relative difference (%)
Epochs	150	117	+28.21
Depth	6	7	-14.29
Initial learning rate	8.49e-05	1.00e-04	+17.79
Max learning rate	1.00e-03	1.03e-03	-2.91
Batch size	64	70	-8.57
Test cross-entropy	0.857	0.866	-1.04

Table 6.5.0.1: Comparison of chosen and mean optimal hyperparameter values for *50k* GNN multiclass from in total 100 hyperparameter optimization random searches.

Hyperparameters	Chosen values	Mean optimal values	Relative difference (%)
Epochs	150	117	+28.21
Depth	6	6	0.00
Initial learning rate	1.00e-04	1.07e-04	-6.54
Max learning rate	1.00e-03	9.07e-04	+10.25
Batch size	64	66	-3.03
Test RMSE	2.350	2.351	-0.04

Table 6.5.0.2: Comparison of chosen and mean optimal hyperparameter values for *50k* GNN regression from in total 100 hyperparameter optimization random searches.

From Tables 6.5.0.1 and 6.5.0.2, the chosen epochs, both multiclass classification and regression, are somwhat higher than the average value of 117 as a result of the hyperparameter optimization searches. The chosen initial learning rate in multiclass classification and the chosen max learning rate for regression are also slightly above average. The chosen depth in multiclass classification, however, was slightly less than that of the average, with the rest of the chosen or default hyperparameters not off by more than 10% from the mean optimal values from the random searches. Moreover, if we look at the mean value of the optimal test metrics (cross-entropy for multiclass and regression for RMSE), we notice that from the several runs, the impact of having a different combination of parameters did not exert a remarkable impact on the model's performance. Therefore, an argument can be made that the chosen (and default) hyperparameters that were used to train the machine learning models were within reasonable ranges of optimality.

However, certain limitations did come in our way while conducting hyperparameter optimization experiments. The first and most significant was timing, as running one hyperparameter optimization random search evaluation is essentially the same as re-training one model given a set of randomly sampled parameters. Throughout this project, it was noticed that training one *50k* model (with the chosen and default hyperparameters), regardless of the algorithms used, takes around 3–5 hours. As a convention, the hyperparameter optimization experiment was designed with 50 evaluations in total, thus completing one single run of the HPO could take from 150 hours to 250 hours, which usually led to timeout errors on the **ALICE** cluster. Therefore, to save time and to increase the chance of success to complete such evaluations, HPO tasks were broken down to multiple smaller runs with their results combined, as the random searches are supposed to be mututally independent of each other given the same search boundaries. That ultimately led to a range of "optimal parameters" instead of a single set as we presented, which makes it impossible to evaluate the potential performance increase of applying a optimal set of hyperparameters to the machine learning models after our HPO runs. Nevertheless, we managed to collect the test metrics at the end of each random search evaluation and plot all of them on Figures 5.3.0.1 and 5.3.0.2. Hence, the results demonstrate that the test metrics rarely fluctuated before and after hyperparameter optimization runs (see Tables 6.5.0.1 and 6.5.0.2).

6.6 Limitations

Finally, we would like to highlight some limitations and bottlenecks experienced during this project. To begin with, the research question and its novelty were entirely built on top of predictions from simulations of chemical processes. In other words, the potential error propogation began from the step where the **SMILES** strings were given to **AiZynthFinder** to solve for their routh lengths. As a chemical validation of the exact route length is extremely difficult and time-consuming, which is also out of the scope of this master's thesis, such limitations should be addressed.

Moreover, regarding the performance of the models, it was observed that **XGBoost** tends to overfit a lot and generalize poorly on partitions smaller than *50k*, which in the early stage of the experiment provided some false sense of confidence. The choice of this algorithm as a starting point was due to its straightforward interpretability and relatively easier hands-on features for predictive and modeling tasks, but as Table 6.2.1.1 indicates, XGBoost may not be a very good fit for the nature of this machine learning task due to its affinity to structured data instead of graph-like molecules. As a mitigation, we introduced the **Chemprop GNN** model, which was proven to be the right choice.

Depite that measure, overfitting was still a common unsolved issue between the two algorithms, likely due to the nature of the data source, since molecules still bear certain chemical structures and functional groups that appear more often than the others, which is an inevitable source of overfitting. Moreover, as introduced in Chapter 4.3.1, the original *Papyrus* dataset, that the **CASPYrus 50k** is derived from, is itself a curated dataset centralizing bioactive, drug-like molecules. That centralization essentially created an unbalanced data distribution that inevitably propagated to our model training, thus causing models to overfit. On top of that, the results from the HPO experiments also suggested that the chosen hyperparameters may cause overfitting due to higher chosen values of the training epochs.

Another bottleneck was the computation capacity of **ALICE**. As mentioned in Chapter 2.5.1, it is a shared supercomputing cluster of Leiden University, of which the availability of GPU nodes was not always guaranteed. There was also a hard stopping point of 7 days for long-running tasks, which our hyperparameter optimization experiments oftenly exceeded. Therefore, compromises were made to break down a single, long HPO run into multiple smaller runs, which led to a less interpretable result.

Chapter 7

Conclusions and Future Work

7.1 Overview

This project focuses on applying machine learning and predictive modeling to the traditional chemical simulation in the field of CASP on modern high-performance computing (**HPC**) clusters. Professional CASP software and databases **AIZynthFinder**, **CASPYrus**, and **ChEMBL** are extensively studied and utilized to generate training and evaluation datasets. Machine learning models based on two algorithms, **XGBoost** and **GNN**, were trained and evaluated to predict synthetic route lengths of organic molecules encoded in **SMILES**. Results show that the GNN *50k* multi-class classification model offers the best prediction and generalization capabilities, while the *50k* regression model was effective only at approximating shorter route lengths $(1-4)$ or recognizing unsolvable molecules. The **XGBoost** models, in contrast, suffered from overfitting issues likely due to their natural affinity to structured, tabular data types rather than chemical molecules. Hyperparameter optimization experiments were conducted. The results indicate that the default and chosen training hyperparameters are within reasonable ranges. In the end, certain limitations and possible causes of the overfitting are analyzed and discussed.

7.2 Answer to research question

To some extent, with comprehensive training and tuning, a machine learning model that is able to work with graph-like data structures is capable of predicting the lengths of the synthetic route lengths originated from the CASP software **AIZynthFinder**. Our most performing model, *50k* GNN multiclass classification, achieved a classification accuracy of 0.628 in generalizability evaluation (see Figure 5.2.3.1), which proved the feasibility of applying machine learning to direct synthetic route length estimation given only the chemical structures, in contrast to running resource-intensive CASP workflows.

7.3 Future work

As inspired by many related works and discoveries came across during this project, future works could be carried out to further push the boundary of human knowledge in this field. For instance, future studies may range from using synthesizability scores in a *de novo* drug design setting to generate easy-to-synthesise molecules that do not express a lot of side chains against targets of interest to uncertainty estimation of molecular synthetic route prediction. Also, the feasibility of predicting synthetic route lengths also sheds

light on the possibilities of predicting other molecular or reaction properties based on molecular structures, such as reaction cost [23], reaction greenness, toxicity estimation, and so on.

Chapter 8

Acknowledgement and reflection

I would like to express my most sincere appreciation to my supportive supervisors, MSc Alan Kai Hassen and Prof. Dr. Mike Preuss; my amazing girlfriend, Manon; my loving parents and grandparents; and my best friends, Janne, Elliot, Nick, and Ning, who offered very much needed technical and emotional support to complete this master's thesis. This is not easy work to begin with, and it has been further complicated by even more personal circumstances that, without their support, I could not possibly complete it.

Looking back, sometimes I wonder if this project should have taken so long. I think I can excuse myself with some personal circumstances, such as timing, financial stability, mental health, and level of knowledge, which all contributed to the personal factors of limitations in completing this project. Also, I got lucky (and unlucky at the same time) to have found a full-time job before I actually started working on the experiments. With essentially 2 full-time jobs (because this thesis counts as one), my time and energy were spread thin. As an international student, I have to pay expensive tuition fees. At the same time, as a working adult, I have the responsibility of balancing my work, my daily life, my health, and my relationships with everyone around me, with the thesis project added on top. Inevitably, it has caused me mental health problems that have slowed everything I had to/wanted to do. In essence, I think it is fair to say that I also had to thank myself for pushing it through during the most difficult times. That tiny bit of faith kept me going bit by bit, line by line, eventually leading me to finishing up one component of the code, to wrapping up the experiments, to finishing writing my first chapter, and eventually to the completion of this thesis. What happened in the middle will forever mean a lot to me. Not only did I learn technical skills but also a valuable ability to figure out the priorities of things and still be able to steer the most important when under pressure from all sides. And, more importantly than being able to handle multiple things at the same time, I realized that not everything has to happen at the same time. Sometimes unloading things off my shoulder for a brief moment can help to accelerate things in the long run. That being said, I would never say thanks to the pressure that pushed me to grow and mature, though, but to myself, who somehow learned and matured during the challenge and managed to survive.

In terms of the end result of the project, it is fair to say that there are regrets and areas for improvement. More polished and in-depth results and experiments could be made possible if such limiting factors were not the case. Or, if I had started reading literature and experimenting with code earlier, a larger scale of this project could be achieved, such as predicting more than just route lengths or even coming up with a new route score. The lessons learned from this project are that, to begin with, knowledge of the strengths and weaknesses of certain algorithms and models should be gathered before starting to experiment in order to find

a fitting model for a certain task. In addition, better project planning could be helpful for future efforts to reduce the trial-and-error part of the time spent and improve productivity. Now I have learned that knowing what to do, where to start, and whom to ask contributes to at least 50% of the eventual success of any project.

If I had to conclude my master's journey in one sentence, I would still say that I appreciate everything that has led me to it. Not everyone is lucky enough to face these challenges and overcome them, and I am one of those blessed few. When feeling down, I sometimes think of myself as if I am the main character of the opening music of *Cid Meier's Civilization 6* (a video game). The name of that soundtrack is *Sogno di Volare*, meaning *The Dream of Flight* in English. What goes up must come down, as flight as a process has to overcome gravity, which is doomed to never take place for free. But if there is a chance for me to jump off the ground and have a view from an angle that I would never be able to imagine from the surface to revisit myself, my past, my area of interest, and my future, that is the price I will not mind paying. I would like to thus conclude this project and my master's journey with my favorite quote from my favorite video game that always gives me some energy boost when reading:

"*From the first stirrings of life beneath water... to the great beasts of the Stone Age... to man taking his first upright steps, you have come far. Now begins your greatest quest: from this early cradle of civilization on towards the stars.*"

Bibliography

[1] Abeer Aljuaid and Mohd Anwar. Survey of supervised learning for medical image processing. *SN Computer Science*, 3(4):292, 2022. Published May 17, 2022.

[2] Anaconda, Inc. *Anaconda Software Distribution*, 2024. Accessed: 2024-10-26.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Cameron B. Browne et al. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] Olivier Béquignon, B. Bongers, Willem Jespers, A. IJzerman, B. Water, and Gerard Van Westen. Papyrus: a large-scale curated dataset aimed at bioactivity predictions. *Journal of Cheminformatics*, 15, 01 2023.

[7] Tianfeng Chai and Roland R. Draxler. Root mean square error (rmse) or mean absolute error (mae)? *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[8] Shuan Chen and Yousung Jung. Estimating the synthetic accessibility of molecules with building block and reaction-aware sascore. *Journal of Cheminformatics*, 16(1):83, July 2024. E-published: 2024 Jul 23; PMID: 39044299.

[9] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[10] E. J. Corey and W. Todd Wipke. Computer-assisted design of complex organic syntheses. *Science*, 166(3902):178–192, 1969.

[11] E.J. Corey and X.M Cheng. *The Logic of Chemical Synthesis*. Wiley, 1989.

[12] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[13] Jiang dejun, Zhenxing Wu, Kim Hsieh, Chen Guangyong, Ben Liao, Zhe Wang, Chao Shen, Dong-Sheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 09 2020.

[14] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1):8, June 2009. PMID: 20298526.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[16] Matthias Feurer and Frank Hutter. *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham, 2019.

[17] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

[18] Samuel Genheden, Amol Thakkar, Veronika Chadimová, Jean-Louis Reymond, Ola Engkvist, and Esben Bjerrum. AiZynthFinder, December 2020.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Thiago S. Guzella and Walmir M. Caminhas. Review: A review of machine learning approaches to spam filtering. *Expert Syst. Appl.*, 36(7):10206–10222, September 2009.

[21] Felix Haase and Matthias Neuenkirch. Predictability of bull and bear markets: A new look at forecasting stock market regimes (and returns) in the us. *International Journal of Forecasting*, 39(2):587–605, 2023.

[22] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Boca Raton, FL, 2010.

[23] Peng Han, Peilin Zhao, Chan Lu, Junzhou Huang, Jiaxiang Wu, Shuo Shang, Bin Yao, and Xiangliang Zhang. Gnn-retro: Retrosynthetic planning with graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36:4014–4021, 06 2022.

[24] Peng Han, Peilin Zhao, Chan Lu, Junzhou Huang, Jiaxiang Wu, Shuo Shang, Bin Yao, and Xiangliang Zhang. Gnn-retro: Retrosynthetic planning with graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):4014–4021, Jun. 2022.

[25] Alan Kai Hassen, Martin Šícho, Yorick J. van Aalst, Mirjam C. W. Huizenga, Darcy N. R. Reynolds, Sohvi Luukkonen, Andrius Bernatavicius, Djork-Arné Clevert, Antonius P. A. Janssen, Gerard J. P. van Westen, and Mike Preuss. Generate what you can make: achieving in-house synthesizability with readily available resources in de novo drug design. *Journal of Cheminformatics*, 17:Article number: 41, 2025.

[26] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2nd edition, 2009.

[27] Esther Heid, Kevin P. Greenman, Yunsie Chung, Shih-Cheng Li, David E. Graff, Florence H. Vermeire, Haoyang Wu, William H. Green, and Charles J. McGill. Chemprop: A machine learning package for chemical property prediction. *Journal of Chemical Information and Modeling*, 64(1):9–17, 2024. PMID: 38147829.

[28] Esther Heid, Kevin P. Greenman, Yunsie Chung, Shih-Cheng Li, David E. Graff, Florence H. Vermeire, Haoyang Wu, William H. Green, and Charles J. McGill. Chemprop: A machine learning package for chemical property prediction. *Journal of Chemical Information and Modeling*, 64(1):9–17, 2024. PMID: 38147829.

[29] Morris A. Jette and Tim Wickberg. Architecture of the slurm workload manager. In D. Klusáček, J. Corbalán, and G.P. Rodrigo, editors, *Job Scheduling Strategies for Parallel Processing*, volume 14283 of *Lecture Notes in Computer Science*. Springer, Cham, 2023. Accessed: 2024-10-26.

[30] Ian T Jolliffe. *Principal component analysis*. Springer, 2002.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, 2006.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[34] Leiden University. Alice high performance computing facility, 2024. Accessed: 2024-10-26.

[35] D. V. Lindley and A. F. M. Smith. Bayes estimates for the linear model. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(1):1–41, 1972.

[36] Cheng-Hao Liu, Maksym Korablyov, Stanisław Jastrzębski, Paweł Włodarczyk-Pruszyński, Yoshua Bengio, and Marwin Segler. Retrognn: Fast estimation of synthesizability for virtual screening and de novo design by learning from slow retrosynthesis software. *Journal of Chemical Information and Modeling*, 62(10):2293–2300, 2022. PMID: 35452226.

[37] Daniel Lowe. Chemical reactions from US patents (1976-Sep2016). 6 2017.

[38] J. MacQueen. Some methods for classification and analysis of multivariate observations. 1967.

[39] Brian W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975.

[40] Microsoft. *Visual Studio Code*, 2024. Accessed: 2024-10-26.

[41] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2024.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[43] Python Software Foundation. *Python Programming Language*, 2024. Accessed: 2024-10-26.

[44] RDKit Contributors. Maccs molecular fingerprints for casp. Implementation of MACCS (Molecular ACCess System) 166-bit structural keys for cheminformatics analysis in CASP workflows. Used for molecular similarity assessment and retrosynthetic planning in tools like ASKCOS.

[45] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. PMID: 20426451.

[46] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[47] Richard M. Russell. The cray-1 computer system. *Commun. ACM*, 21(1):63–72, January 1978.

[48] Lakshidaa Saigiridharan, Alan Kai Hassen, Helen Lai, Paula Torren-Peraire, Ola Engkvist, and Samuel Genheden. Aizynthfinder 4.0: developments based on learnings from 3 years of industrial application. *Journal of Cheminformatics*, 16(1):57, 2024.

[49] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[50] 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.

[51] Gisbert Schneider and Uli Fechner. Computer-based de novo design of drug-like molecules. *Nature Reviews Drug Discovery*, 4(8):649–663, August 2005.

[52] Marwin Segler, Mike Preuss, and Mark Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555:604–610, 03 2018.

[53] Martin Seifrid, Riley J. Hickman, Andrés Aguilar-Granda, Cyrille Lavigne, Jenya Vestfrid, Tony C. Wu, Théophile Gaudin, Emily J. Hopkins, and Alán Aspuru-Guzik. Routescore: Punching the ticket to more efficient materials development. *ACS Central Science*, 8(1):122–131, 2022.

[54] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[55] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45:427–437, 07 2009.

[56] Teague Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015. PMID: 26479676.

[57] Nicolas Suhadolnik, Jo Ueyama, and Sergio Da Silva. Machine learning for enhanced credit risk assessment: An empirical approach. *Journal of Risk and Financial Management*, 16(12), 2023.

[58] Igor V. Tetko, Pavel Karpov, Ruud van Deursen, and Guillaume Godin. Augmented transformer achieves 97% and 85% for top5 prediction of direct and classical retro-synthesis. *CoRR*, abs/2003.02804, 2020.

[59] Amol Thakkar, Veronika Chadimová, Esben Jannik Bjerrum, Ola Engkvist, and Jean-Louis Reymond. Retrosynthetic accessibility score (rascore) – rapid machine learned synthesizability classification from ai driven retrosynthetic planning. *Chem. Sci.*, 12:3339–3349, 2021.

[60] Cornelis J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979. Classic reference for F-measure derivation.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[62] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[63] Linli Xu, Martha White, and Dale Schuurmans. Optimal reverse prediction: a unified perspective on supervised, unsupervised and semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1137–1144, New York, NY, USA, 2009. Association for Computing Machinery.

[64] Barbara Zdrazil, Eloy Felix, Fiona Hunter, Emma J Manners, James Blackshaw, Sybilla Corbett, Marleen de Veij, Harris Ioannidis, David Mendez Lopez, Juan F Mosquera, Maria Paula Magarinos, Nicolas Bosc, Ricardo Arcila, Tevfik Kizilören, Anna Gaulton, A Patrícia Bento, Melissa F Adasme, Peter Monecke, Gregory A Landrum, and Andrew R Leach. The ChEMBL Database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic Acids Research*, 52(D1):D1180–D1192, 11 2023.

[65] Shuangjia Zheng, Jiahua Rao, Zhongyue Zhang, Jun Xu, and Yuedong Yang. Predicting retrosynthetic reactions using self-corrected transformer neural networks. *Journal of Chemical Information and Modeling*, 60(1):47–55, 2020. PMID: 31825611.