h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Darmstadt University of Applied Sciences

Faculty of Mathmatics and Natural Sciences &
Faculty of Computer Science

Optimizing Pre-Training Strategies for a Chemical Foundation Model Using VICReg

Peer Schliephacke

Matriculation number 1129023

First Supervisor: Prof. Dr. Jutta Groos

Second Supervisor: Prof. Dr. Elke Hergenröther

Submitted in partial fulfilment of the requirements for the degree of
Master of Science (M. Sc.) in *Data Science*

Issue Date: April 1, 2025
Submission Date: July 15, 2025

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Während der Vorbereitung dieser Arbeit habe ich ChatGPT verwendet, um vereinzelte Textpassagen in englischer Sprache stilistisch anzupassen. Nach der Nutzung dieses Tools/Dienstes habe ich den Inhalt nach Bedarf überprüft und bearbeitet und übernehme die volle Verantwortung für den Inhalt der Veröffentlichung.

Darmstadt, July 15, 2025

Peer Schliephacke

Abstract

Recent advances in self-supervised representation learning have introduced VICReg (Variance-Invariance-Covariance Regularization), a method designed to maximize agreement between embedding vectors produced by different encoders to generate meaningful representations. VICReg outperforms other self-supervised methods and does not rely on collapse-prevention techniques (e.g., stop-gradient operations, memory banks, or output quantization) addressing the common representation learning problem where models produce nearly identical embeddings for all inputs, yielding meaningless representations. Initially proposed for computer vision, VICReg uses differently augmented views of the same image, training their representations to align. In this thesis, VICReg is adapted to computational chemistry, where different molecular modalities serve as distinct views in a multi-modal training approach. Specifically: Graph Isomorphism Networks (GIN) capture graph-structured data, Equivariant Graph Neural Networks (EGNN) represent conformational data, and Long Short-Term Memory (LSTM) networks encode chemical language representations. The resulting modality-specific encoders are evaluated after VICReg pre-training through both linear and transfer learning evaluation protocols on an independent Absorption, Distribution, Metabolism, and Excretion (ADME) dataset. Additional experiments compare against model-agnostic supervised pre-training and modality-specific self-supervised pre-training using identical evaluation protocols. This work addresses several research questions: (i) How do supervised model-agnostic pre-training strategies compare to self-supervised, modality-specific approaches? (ii) Can VICReg be successfully adapted from computer vision to computational chemistry using multi-modal molecular representations without collapse? (iii) Which experimental configurations optimize downstream performance with VICReg? (iv) Does VICReg outperform both modality-specific and model-agnostic pre-training strategies in downstream tasks?

Zusammenfassung

VICReg (Variance-Invariance-Covariance Regularization) beschreibt eine Technik des selbstüberwachten Repräsentationslernens, um die Übereinstimmung zwischen von unterschiedlichen Encodern erzeugten Embedding-Vektoren zu maximieren. VICReg übertrifft andere selbstüberwachte Methoden und ist nicht auf Techniken angewiesen, um den sogenannten „Collapse"-Effekt zu vermeiden (z.B. Stop-Gradient-Operationen, Memory Banks oder Output-Quantisierung), ein Phänomen, bei dem das Modell nahezu identische Embeddings für alle Eingaben erzeugt. VICReg kommt ursprünglich aus der Computer Vision Domäne und nutzt unterschiedlich augmentierte Ansichten desselben Bildes, um die encodeten Repräsentationen unter einander anzugleichen. In dieser Arbeit wird VICReg auf den Bereich der Chemoinformatik übertragen, indem verschiedene Modalitäten eines Moleküls als unterschiedliche Ansichten in einem multimodalen Trainingsansatz dienen. Konkret erfassen Graph Isomorphism Networks (GIN) die Graph-Modalität, Equivariant Graph Neural Networks (EGNN) repräsentieren die Konformer-Modalität, und Long Short-Term Memory (LSTM)-Netzwerke kodieren die chemische Sprachmodalität. Die daraus resultierenden, modalitätsspezifischen Encoder werden nach dem VICReg Pre-training mithilfe einer linearen Evaluierung und einer Transfer-Learning-Evaluierung auf einem unabhängigen Datensatz für Absorption, Distribution, Metabolism und Excretion (ADME) getestet. Zudem wird in weiteren Experimenten jeder Encoder in einem modellunabhängigen, überwachten Pre-training und in einem modalitätsspezifischen, selbstüberwachten Pre-training trainiert und mit derselben Evaluierungsstrategie evaluiert. Abschließend erörtert diese Thesis verschiedene Forschungsfragen: (i) Wie vergleichbar sind überwachte, modellunabhängige Pre-training-Strategien gegenüber selbstüberwachten, modellspezifischen Pre-training-Strategien? (ii) Lässt sich VICReg erfolgreich von der Computer Vision auf die Chemoinformatik übertragen, indem Graph-, Konformer- und Textmodalitäten eines Moleküls genutzt werden? (iii) Welche VICReg Einstellungen führen zu besseren Downstream-Performances? (iv) Übertrifft VICReg sowohl modalitätsspezifisches Pre-training als auch modellunabhängige Pre-training-Strategien in Bezug auf die Downstream-Performance?

Contents

List of Figures vii

List of Tables x

List of Abbreviations xii

1 Introduction 1

1.1 Motivation and Research Questions 1

1.2 Structure 3

2 Theoretical Background 4

2.1 Computational Chemistry and Pharmacology 4

2.1.1 Pharmacokinetics 4

2.1.2 Simplified Molecular Input Line Entry System 5

2.1.3 Molecular Features 7

2.1.4 Compound Similarity 8

2.2 Modeling 10

2.2.1 eXtreme Gradient Boosting 10

2.2.2 Multi-Layer Perceptron 11

2.2.3 Language Modeling 12

2.2.4 Graph Neural Networks 17

2.2.5 Variance-Invariance-Covariance Regularization 22

2.3 Statistics 25

2.3.1 Metrics 25

2.3.2 Analysis of Variance 26

2.3.3 Tukey Honestly Significant Difference Post-Hoc Test 28

3 Methods **29**
3.1 Data . . . 29
3.1.1 Pre-training Dataset . . . 29
3.1.2 Downstream Dataset . . . 31
3.1.3 Preprocessing . . . 33
3.2 Architecture Details . . . 37
3.2.1 Hyperparameter Tuning . . . 37
3.2.2 VICReg for Computational Chemistry . . . 40
3.3 Performance Evaluation . . . 43
3.3.1 Evaluation Protocols . . . 43
3.3.2 Assess Performance Differences . . . 45
3.4 Experiments . . . 45
3.4.1 No Pre-training . . . 45
3.4.2 Model-agnostic Pre-training . . . 46
3.4.3 Model-specific Pre-training . . . 50

4 Results and Discussion **54**
4.1 Effect of Pre-training . . . 54
4.1.1 Linear Evaluation . . . 54
4.1.2 Transfer Learning . . . 56
4.1.3 Impact of Pre-training Strategies . . . 58
4.2 Different VICReg Strategies . . . 60
4.2.1 ADME vs Largemix . . . 60
4.2.2 Different Pre-training Strategies . . . 62
4.2.3 Impact of Differing VICReg Strategies . . . 65
4.3 Across Experiments . . . 67
4.3.1 Linear Evaluation . . . 67
4.3.2 Transfer Learning . . . 69
4.3.3 Comparing VICReg to other Pre-training Strategies . . . 70

5 Conclusions and Future Work **75**
5.1 Conclusions . . . 75
5.2 Future work . . . 77

Bibliography **79**

Appendices **94**

A Implementation Details **94**

A.1 Data . . . 94
A.1.1 ADME Data . . . 94
A.2 Hyperparameter Tuning . . . 95
A.2.1 GIN . . . 95
A.2.2 EGNN . . . 96
A.2.3 LSTM . . . 97
A.2.4 XGB . . . 98
A.3 Resulting Architecture . . . 100
A.3.1 GIN . . . 100
A.3.2 EGNN . . . 101
A.3.3 LSTM . . . 102
A.3.4 XGB . . . 102
A.4 Used Hardware and Software . . . 105
A.4.1 Hardware . . . 105
A.4.2 Software . . . 105

B Further Results **106**

B.1 Effect of Pre-training . . . 106
B.1.1 Aggregated . . . 106
B.1.2 GIN . . . 107
B.1.3 EGNN . . . 109
B.1.4 LSTM . . . 111
B.2 Different VICReg Strategies . . . 113
B.2.1 No Pre-training vs Pre-training . . . 113
B.2.2 Agnostic Pre-training vs Specific Pre-training . . . 120
B.3 Across Experiments . . . 127

List of Figures

2.1 Convert Molecules into SMILES 6
2.2 Featurizing Molecules with the Morgan Fingerprint 8
2.3 Schema of a LSTM Cell 12
2.4 Schema of Multihead Attention 15
2.5 Pre-training of Language Models with Token Masking 17
2.6 Message Passing of GNNs 18
2.7 Pre-training of GNNs with Node Masking 21
2.8 Structure of VICReg 22

3.1 Featurized Molecule 34
3.2 Tokenization of SMILES Strings 36
3.3 Schema of Hyperparameter tuning based on Compound Similarity . . . 38
3.4 VICReg Architecture for Computational Chemistry 41
3.5 Linear Evaluation Protocol 43
3.6 Transfer Learning Protocol 44
3.7 Supervised Pre-train Agnostic Architecture 49
3.8 Node Masking Pre-training with Node Attribute Masking 52

4.1 Linear Evaluation Results for different Pre-training Strategies (MAE) . 55
4.2 Transfer Learning Results for different Pre-training Strategies (MAE) . 57
4.3 Linear Evaluation Results for VICReg (MAE) 61
4.4 Transfer Learning Results for VICReg (MAE) 62
4.5 Linear Evaluation Results for VICReg with pre-trained Encoders (MAE) 63
4.6 Transfer Learning Results for VICReg with pre-trained Encoders (MAE) 65
4.7 Linear Evaluation Results of different Pre-training Strategies (MAE) . . 68
4.8 Transfer Learning Results of different Pre-training Strategies (MAE) . . 70
4.9 Correlation of Downstream ADME Endpoints 73

B.1 Linear Evaluation Results for different Pre-trainings (R2) 106
B.2 Transfer Learning Evaluation Results for different Pre-trainings (R2) . . 106
B.3 Linear Evaluation of Pre-training using GIN (MAE) 107
B.4 Linear Evaluation of Pre-training using GIN (R2) 107
B.5 Transfer Learning Evaluation of Pre-training using GIN (MAE) 108
B.6 Transfer Learning Evaluation of Pre-training using GIN (R2) 108
B.7 Linear Evaluation of Pre-training using EGNN (MAE) 109
B.8 Linear Evaluation of Pre-training using EGNN (R2) 109
B.9 Transfer Learning Evaluation of Pre-training using EGNN (MAE) . . . 110
B.10 Transfer Learning Evaluation of Pre-training using EGNN (R2) 110
B.11 Linear Evaluation of Pre-training using LSTM (MAE) 111
B.12 Linear Evaluation of Pre-training using LSTM (R2) 111
B.13 Transfer Learning Evaluation of Pre-training using LSTM (MAE)) . . . 112
B.14 Transfer Learning of Pre-training using LSTM (R2) 112
B.15 Linear Evaluation Results for VICReg (R2) 113
B.16 Transfer Learning Results for VICReg (R2) 113
B.17 Linear Evaluation of VICReg using GIN (MAE) 114
B.18 Linear Evaluation of VICReg using GIN (R2) 114
B.19 Transfer Learning Evaluation of VICReg using GIN (MAE) 115
B.20 Transfer Learning Evaluation of VICReg using GIN (R2) 115
B.21 Linear Evaluation of VICReg using EGNN (MAE) 116
B.22 Linear Evaluation of VICReg using EGNN (R2) 116
B.23 Transfer Learning Evaluation of VICReg using EGNN (MAE) 117
B.24 Transfer Learning Evaluation of VICReg using EGNN (R2) 117
B.25 Linear Evaluation of VICReg using LSTM (MAE) 118
B.26 Linear Evaluation of VICReg using LSTM (R2) 118
B.27 Transfer Learning Evaluation of VICReg using LSTM (MAE) 119
B.28 Transfer Learning Evaluation of VICReg using LSTM (R2) 119

B.29 Linear Evaluation Results for VICReg (R2) with Pre-trained Encoders . 120
B.30 Transfer Learning Results for VICReg (R2) with Pre-trained Encoders . 120
B.31 Linear Evaluation of pre-trained VICReg using GIN (MAE) 121
B.32 Linear Evaluation of pre-trained VICReg using GIN (R2) 121
B.33 Transfer Learning Evaluation of pre-trained VICReg using GIN (MAE) . 122
B.34 Transfer Learning Evaluation of pre-trained VICReg using GIN (R2) . . 122
B.35 Linear Evaluation of pre-trained VICReg using EGNN (MAE) 123
B.36 Linear Evaluation of pre-trained VICReg using EGNN (R2) 123
B.37 Transfer Learning Evaluation of pre-trained VICReg using EGNN (MAE) 124
B.38 Transfer Learning Evaluation of pre-trained VICReg using EGNN (R2) . 124
B.39 Linear Evaluation of pre-trained VICReg using LSTM (MAE) 125
B.40 Linear Evaluation of pre-trained VICReg using LSTM (R2) 125
B.41 Transfer Learning Evaluation of pre-trained VICReg using LSTM (MAE) 126
B.42 Transfer Learning Evaluation of pre-trained VICReg using LSTM (R2) . 126
B.43 Linear Evaluation Results for VICReg vs No VICReg (R2) 127
B.44 Transfer Learning Results for VICReg vs No VICReg (R2) 127
B.45 Linear Evaluation of VICReg vs No VICReg using GIN (MAE) 128
B.46 Linear Evaluation of VICReg vs No VICReg using GIN (R2) 128
B.47 Transfer Learning Evaluation of VICReg vs No VICReg using GIN (MAE) 129
B.48 Transfer Learning Evaluation of VICReg vs No VICReg using GIN (R2) 129
B.49 Linear Evaluation of VICReg vs No VICReg using EGNN (MAE) 130
B.50 Linear Evaluation of VICReg vs No VICReg using EGNN (R2) 130
B.51 Transfer Learning Evaluation of VICReg vs No VICReg using EGNN (MAE) 131
B.52 Transfer Learning Evaluation of VICReg vs No VICReg using EGNN (R2) 131
B.53 Linear Evaluation of VICReg vs No VICReg using LSTM (MAE) 132
B.54 Linear Evaluation of VICReg vs No VICReg using LSTM (R2) 132
B.55 Transfer Learning Evaluation of VICReg vs No VICReg using LSTM (MAE) 133
B.56 Transfer Learning Evaluation of VICReg vs No VICReg using LSTM (R2) 133

List of Tables

3.1 Pre-training Data 31
3.2 Raw ADME Data 32

A.1 ADME Training Data 95
A.2 ADME Test Data 95
A.3 GIN Architecture Hyperparameter Distribution 95
A.4 GIN Optimizer Hyperparameter Distribution 96
A.5 EGNN Architecture Hyperparameter Distribution 96
A.6 EGNN Optimizer Hyperparameter Distribution 97
A.7 LSTM Architecture Hyperparameter Distribution 97
A.8 LSTM Optimizer Hyperparameter Distribution 98
A.9 XGB RLM Hyperparameter Distribution 98
A.10 XGB HLM Hyperparameter Distribution 99
A.11 XGB MDR1-ER Hyperparameter Distribution 99
A.12 XGB Sol Hyperparameter Distribution 99
A.13 GIN Architecture 100
A.14 EGNN Architecture 101
A.15 LSTM Architecture 102
A.16 XGB RLM Architecture 102
A.17 XGB HLM Architecture 103
A.18 XGB MDR1-ER Architecture 103
A.19 XGB Sol Architecture 104

List of Abbreviations

ASCII American Standard Code for Information Interchange 5

ADME Absorption, Distribution, Metabolism, and Excretion 3

AI Artificial Intelligence 1

ANOVA Analysis of Variance 26

BCE Binary Cross-Entropy 47

BERT Bidirectional Encoder Representations from Transformers 16

BN Batch Normalization 43

CLS Classification 36

CV Computer Vision 1

DL Deep Learning 1

ECFP Extended-connectivity fingerprints 7

EGNN Equivariant Graph Neural Networks 2

ETKDG Experimental-Torsion Distance Geometry 34

GIN Graph Isomorphism Network 2

GNN Graph Neural Network 2

GPT Generative pre-trained transformer 16

HLM Human Liver Micrososomal Stability 31

hPPB Human Plasma Protein Binding 31

LSTM Long Short-Term Memory 2

MAE Mean Absolute Error . . . 25

MMFF Merck molecular force field . . . 35

MDR1-ER Multidrug Resistance Protein 1 Efflux Ratio . . . 31

ML Machine Learning . . . 1

MLM Masked Language Model . . . 16

MLP Multi-Layer Perceptron . . . 11

MSE Mean Squared Error . . . 38

NLP Natural Language Processing . . . 1

PK Pharmacokinetics . . . 3

QSAR Quantitative Structure-Activity Relationship . . . 2

ReLU Rectified Linear Unit . . . 46

RLM Rat Liver Micrososomal Stability . . . 31

RNN Recurrent Neural Network . . . 12

rPPB Rat Plasma Protein Binding . . . 31

SMILES Simplified Molecular Input Line Entry System . . . 2

Sol Solubility . . . 31

TPE Tree-structured Parzen Estimator . . . 37

Tukey test Tukey Honestly Significant Difference Post-Hoc test . . . 28

VICReg Variance-Invariance-Covariance Regularization . . . 2

XGB eXtreme Gradient Boosting . . . 10

Chapter 1

Introduction

1.1 Motivation and Research Questions

Advances in Artificial Intelligence (AI) have significantly impacted multiple domains including Natural Language Processing (NLP), Computer Vision (CV), and life sciences such as computational chemistry, enabling the development of sophisticated Deep Learning (DL) models that learn complex patterns from extensive datasets. These breakthroughs span from chatbots utilizing large language models [1, 2, 3] to CV systems achieving human-level performance in image classification and object detection [4, 5, 6] and life science innovations like AlphaFold [7], which revolutionized structural biology through precise 3D protein structure prediction. Contemporary developments feature foundation models which are large-scale, general-purpose DL systems trained on broad datasets to capture universal patterns [8, p.44-50]. These models provide transferable representations that reduce reliance on task-specific architectures [9]. The evolution of foundation models has been driven by the transformer architecture [10], whose self-attention mechanism captures contextual relationships. Furthermore, the integration of self-supervised learning [8, p.40-44], a DL paradigm where models derive representations from unlabeled data through intrinsic tasks, enhances the development of foundation models since no expensive data labels are needed. More recently, multimodal foundation models [11, 12, 13] process diverse data types (text, images, video) to learn cross-modal representations, often employing joint embedding architectures [14, 15] that align modalities in shared latent spaces enabling a model to learn even more semantic relationships between large datasets.

In the domain of computational chemistry, the application of foundation models offers potential for more accurate numerical representations of molecules [16] suitable for Machine Learning (ML) algorithms to enhance drug discovery [17] and molecular modeling [18]. This could enable few-shot learning approaches [19], which would

be particularly useful given that labeled molecular data is expensive to generate [20]. Traditional molecular representations use global molecular descriptors and sparse bit-maps indicating the presence of molecular substructures [21]. While such featurization strategies show success in cheminformatics tasks like Quantitative Structure-Activity Relationship (QSAR) prediction [22, 23, 24], DL strategies using graph-based representations through Graph Neural Network (GNN)s have become more prevalent in recent years [25, 26, 27, 28]. The success of graph-derived representations may overcome the task-specific nature of global molecular features and the sparsity of bit-map substructure representations, which are suboptimal for many ML algorithms [29, p.270-271]. However, there is no consensus that graph-based representations are universally better than conventional featurization methods [30], suggesting chemical foundation models could overcome these limitations [16]. Recent featurization approaches include conformer-based graph architectures [31] and text-based representations from the Simplified Molecular Input Line Entry System (SMILES) notation [32], which functions as a chemical language [33, 34]. These diverse molecular featurization strategies can serve as distinct modalities for training multi-modal foundation models. Experiments demonstrate multi-modal learning through combinations like structural formula images with graph/conformer representations using auto-encoders [35, 36], image-graph pairs with contrastive learning [37, 38], text-conformer alignment [39], and text-graph embedding concatenation [40]. However, many approaches rely on contrastive learning, which requires numerous negative pairs to prevent representation collapse and incurs substantial computational costs. Variance-Invariance-Covariance Regularization (VICReg) [41] provides an alternative by enforcing variance preservation across embedding dimensions, invariance between differently projected embeddings of identical samples, and decorrelation of embedding dimensions, eliminating both negative pair requirements and collapse risks. This approach offers flexibility without labeled data and avoids dependence on collapse-prevention techniques like large batch sizes [42], memory banks [43], momentum encoders [44], quantization methods [45], or stop-gradient operations [46].

This thesis explores the adaptation of VICReg to computational chemistry by leveraging molecular modalities to investigate whether VICReg can be effectively trained as a self-supervised molecular foundation model using text, graph, and conformer representations. The graph modality is processed by using a Graph Isomorphism Network (GIN) [47], conformer properties is processed by using an Equivariant Graph Neural Networks (EGNN) [31], and the chemical language features are utilized by using a bidirectional Long Short-Term Memory (LSTM) [48] with self-attention [10]. These encoding networks represent established DL approaches in computational chemistry [49, 50, 51], justifying their selection. Comparative experiments implement: model-agnostic super-

vised pre-training and modality-specific self-supervised pre-training to address: (i) How do supervised and self-supervised pre-training strategies compare in Absorption, Distribution, Metabolism, and Excretion (ADME) downstream performance? The core investigation implements VICReg for molecular representation learning, asking: (ii) Does VICReg pre-training work for molecules without causing representation collapse [52]? Multiple configurations are tested to examine: (iii) Do different VICReg settings significantly affect downstream performance? Finally, the performance comparisons address: (iv) Is VICReg superior to conventional pre-training strategies for chemical foundation models?

1.2 Structure

Chapter 2 contains fundamental theoretical background essential for understanding this work. Section 2.1 outlines key concepts in Pharmacokinetics (PK) and computational chemistry, while Section 2.2 focuses on ML and DL fundamentals for the implemented algorithms and associated terminology. Subsection 2.2.5 details the theoretical framework of the VICReg method. Section 2.3 describes evaluation metrics and corresponding equations for model performance assessment, along with statistical methods for multiple comparisons used in analysis.

Chapter 3 specifies implementation details: data sources and processing (Section 3.1), model architectures (Section 3.2), and evaluation protocols (Section 3.3). Section 3.4 comprehensively documents experimental designs. Chapter 4 presents and discusses findings: Section 4.1 analyzes non-VICReg pre-training comparisons, Section 4.2 examines VICReg configuration outcomes, and Section 4.3 compares VICReg performance against alternative pre-training approaches. Chapter 5 concludes with key insights (Section 5.1) and future research directions (Section 5.2).

Appendix A provides supplementary technical details including in vivo assay specifications, hyperparameter search spaces, architecture optimization results, and hardware/software configurations. Appendix B contains additional figures demonstrating statistically significant differences between experimental conditions.

Chapter 2

Theoretical Background

2.1 Computational Chemistry and Pharmacology

2.1.1 Pharmacokinetics

The field of PK studies how drugs behave in the human body after administration [53], emphasizing the mechanisms of compound transport rather than the identification of target interactions. PK comprises four key properties collectively termed ADME. Optimal ADME characteristics are essential for drug efficacy, as poor properties may negate therapeutic potential despite effective target engagement.

Absorption

Absorption quantifies a compound's entry into systemic circulation. The absorption efficiency depends on chemical properties, formulation, and administration route [54, 53]. Critical physicochemical factors include aqueous solubility, permeability, molecular weight, and ionization state. For example, high water solubility enhances gastrointestinal absorption compared to hydrophobic compounds.

Distribution

Distribution describes compound dispersion through tissues and fluids post-absorption [53]. Key properties include molecular size, polarity, lipophilicity, and plasma protein binding affinity. Suboptimal distribution patterns, such as excessive protein binding or poor tissue permeability, may lead to insufficient target site concentrations [54].

Metabolism

Metabolism characterizes enzymatic modification of compounds into metabolites [53, 54]. Hepatic enzymes are primarily responsible for converting parent compounds into metabolites, which are often inactive. The metabolic rates depend on structural features influencing enzyme-substrate interactions.

Excretion

Excretion encompasses compound and metabolite elimination [53, 54]. Renal clearance predominates as excretion mechanism, though biliary excretion contributes for some compounds. Incomplete excretion risks metabolite accumulation, potentially causing adverse effects through interference with physiological processes.

2.1.2 Simplified Molecular Input Line Entry System

Motivation

Chemical compounds can be represented through structural formulas to emphasize atomic connectivity and spatial relationships. Various formula types exist with distinct syntax elements offering different detail levels [55]. The SMILES notation system [32] encodes compounds as American Standard Code for Information Interchange (ASCII) strings, serving as a cornerstone for computational chemistry.

Specification of SMILES Strings

The SMILES system employs defined syntactic rules to represent molecular graphs $G = (V, E)$, where vertices V denote atoms and edges E represent bonds between atoms. Atom symbols constitute primary elements, with bond types specified using "=" (double), "#" (triple), and ":" (aromatic). Aromatic atoms can also be implicitly denoted through lowercase letters, while single bonds lack explicit symbols. Branching uses parentheses, while cyclic structures employ numerical indicators following atomic symbols for ring openings/closures.

Embedding of a Ciprofloxacin Molecule as SMILES String

Figure 2.1 [56] demonstrates the SMILES encoding for ciprofloxacin. Panel **A** displays the structural formula with key functional groups (hydroxyl, ketone). Panel **B** annotates cyclic structures (1-4), with yellow markers indicating ring openings/closures. Panel **C** color-codes molecular components: green (backbone), other colors (branches). Panel **D** shows the complete SMILES string:

- `N1CCN` initiates the first cyclic structure
- `C1` closes this ring within a branch (`(CC1)`)
- `(C(F)=C2)` encodes a fluorine containing a nested branch with a double bond
- Digits 2-4 manage remaining cyclic structures
- Subsequent branches follow analogous encoding rules

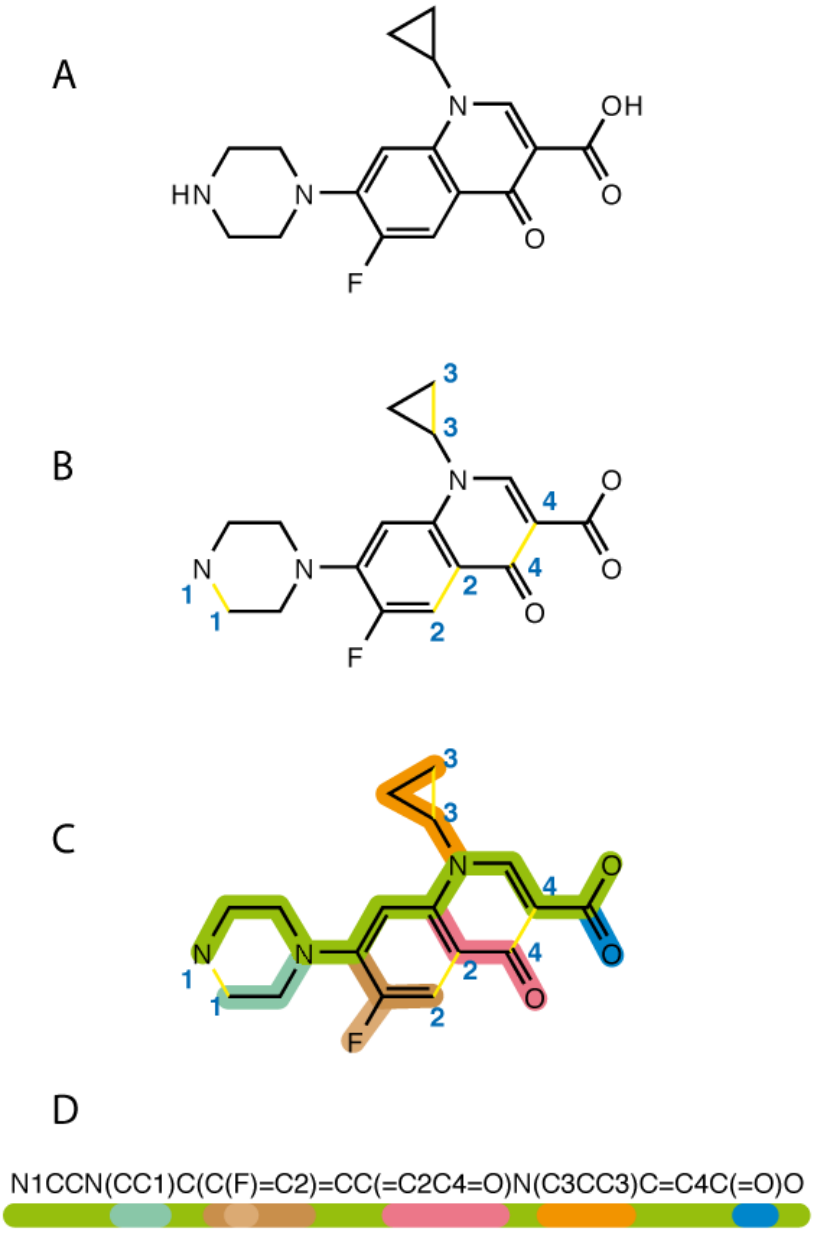

Figure 2.1: Two-dimensional structural formula of molecules can be converted into a SMILES string by using well defined syntactical elements.

2.1.3 Molecular Features

Molecular Descriptors

A suitable numerical representation is required to process molecular structures with ML algorithms. This is achieved through featurization methods that generate feature vectors by encoding chemical structures and properties. These features are often global molecular properties which are also known as molecular descriptors [57] describing numerical values that quantify molecular characteristics from simple properties to complex topological indices [58]. Molecular descriptors were originally developed for QSAR analysis [59], a method which correlates structural features with biological activity to enable predictive modeling of untested compounds. Examples include global molecular properties like molecular weight and hydrogen-bond donor counts [47].

Structural Fingerprints

Additional featurization methods use local molecular characteristics, complementing global molecular descriptors. A prominent approach generates bit-maps that systematically encode molecular substructures [21, 39]. These bit-maps (fingerprints) represent compounds through binary vectors where the positions correspond to specific substructures, with bit values indicating presence/absence of such a substructure. Those fingerprints are generated via substructure hashing into fixed-length vectors resulting in large sparse bit-representations due to numerous possible substructures. The Extended-connectivity fingerprints (ECFP) [60], based on the Morgan algorithm [61], encodes molecular structure through atom types and connectivity within radial neighborhoods.

Figure 2.2 [39] demonstrates how the ECFP operates with varying radii. The representation of an Aspirin molecule uses bit-map with a settable length where indices map to substructures defined by radius r. At r=0, substructures represent individual atoms, r=1 includes atoms with immediate neighbors and so on. Each unique radius dependent substructure hashes to specific bit-vector indices. Across compounds, overlapping substructures activate shared bits. Key limitations of molecular fingerprints include hash collisions from finite vector lengths, mitigated through $1024 - 2048$ bit configurations.

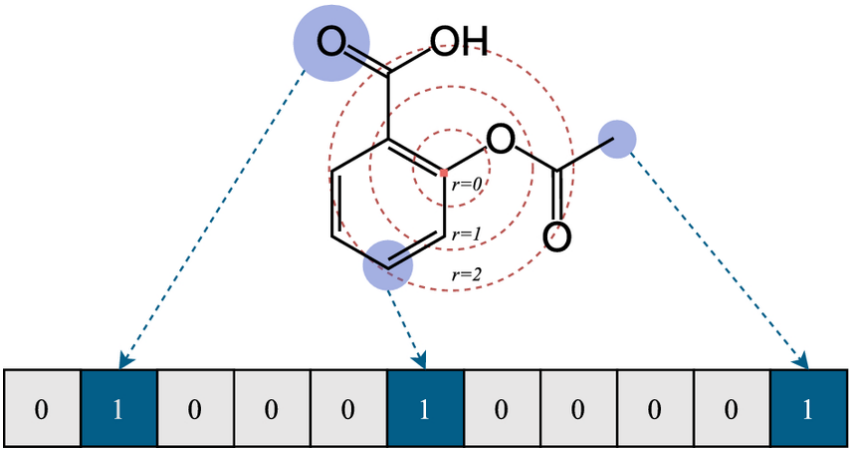

Figure 2.2: An Aspirin molecule is projected onto a bit-map by capturing the absence or presence of certain substructures. The definition of a substructure can be regularized by a radius parameter r.

2.1.4 Compound Similarity

Pairwise Fingerprint Similarity

The Structural similarity between two compounds can be quantified using fingerprint bit-strings generated by algorithms like ECFP. The Tanimoto coefficient T_{sim} [62] measures fingerprint similarity as defined in Equation 2.1 [63] where FP_1 and FP_2 are the molecular fingerprints of two molecules. The numerator counts shared active bits (substructures present in both compounds), while the denominator totals unique active bits across either fingerprint. Values approaching 1 indicate a higher substructural overlap.

$$T_{sim}(FP_1, FP_2) = \frac{|FP_1 \wedge FP_2|}{|FP_1 \vee FP_2|} = \frac{\text{Shared Bits}}{\text{Total Bits}} \tag{2.1}$$

- T_{sim} as similarity function
- FP_1, FP_2 as two bit-strings

Butina Clustering

Compounds can be clustered based on their structural similarity using the Butina algorithm [64]. The clustering algorithm is based on a fingerprint representation of a compound and uses a configurable distance threshold to define the clusters. The distance is defined as $1 - T_{sim}(FP_1, FP_2)$, where $T_{sim}(FP_1, FP_2)$ represents the Tanimoto similarity between the two fingerprints FP_1 and FP_2.

Algorithm 1 shows the pseudo-code of the Butina algorithm. A list of compounds M is taken as input along with a distance threshold d_{cutoff}. A pairwise distance matrix, defined as $D(FP_i, FP_j) = 1 - T_{sim}(FP_i, FP_j)$, is computed by calculating the pairwise Tanimoto distances for every pair of distinct compounds using their fingerprints. The distance threshold d_{cutoff} defines the maximum allowable dissimilarity for two compounds to be grouped into the same cluster. For example, $d_{\text{cutoff}} = 0.2$ corresponds to a minimum similarity of $T_{sim} = 0.8$. The algorithm first sorts molecules by their number of neighbors within d_{cutoff}, prioritizing those with the most neighbors as cluster seeds. Iteratively, each seed forms a cluster with its neighbors, which are then removed from the pool of unassigned molecules. This process continues until all compounds are clustered or remain as singletons.

Algorithm 1 Butina Clustering Algorithm

Input: List of molecules M, distance threshold d_{cutoff}
Output: List of clusters C

Compute pairwise distance matrix D for all molecules in M
For each molecule m_i, count neighbors N_i where $D_{ij} \leq d_{\text{cutoff}}$
Sort molecules in descending order of N_i
Initialize empty cluster list C
Initialize unassigned molecules $U \leftarrow M$
while $U \neq \emptyset$ **do**
    Select first molecule m_{seed} in sorted U
    Find all neighbors of m_{seed} in U: $S \leftarrow \{m_j \in U \mid D_{\text{seed},j} \leq d_{\text{cutoff}}\}$
    Add cluster $c \leftarrow \{m_{\text{seed}}\} \cup S$ to C
    Remove all molecules in c from U
end while
return C

2.2 Modeling

2.2.1 eXtreme Gradient Boosting

eXtreme Gradient Boosting (XGB) [65] is an ensemble ML algorithm that consists of gradient boosted trees [66] with enhancements in regularization and scalability. The model is trained sequentially to minimize the learning objective $\mathcal{L}_{XGB}^{(j)}$ as shown in Equation 2.3 [65] where j refers to the j-th iteration. The first term $\ell(y_i, \hat{y}_i^{(j-1)} + f_j(x_i))$ describes a loss function which calculates the difference between the ground truth y_i and the prediction $\hat{y}_i$. Here, $\hat{y}_i^{(j-1)}$ refers to the prediction from the previous iteration $j-1$ while $f_j(x_i)$ is an additive correction term computed by the tree of the current iteration j based on the input features x_i. Therefore, the ensemble is trained in such a way that each tree corrects the error from the previous tree. The second term $\Omega(f_j)$ serves as a combined regularization term of the tree f_j where L corresponds to the number of leaves and γ_{size} as a regularization for the tree complexity, penalizing deep trees. The term $\frac{1}{2}\lambda_{L2}\|w\|^2$ uses a hyperparameter λ_{L2} to apply L2 regularization to the tree's weights, helping to prevent overfitting.

$$\mathcal{L}_{XGB}^{(j)} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(j-1)} + f_j(x_i)) + \Omega(f_j) \tag{2.2}$$

$$\Omega(f) = \gamma_{size} L + \frac{1}{2}\lambda_{L2}\|w\|^2 \tag{2.3}$$

- ℓ as differentiable loss function at iteration j for n datapoints
- f_j as tree model with parameters w
- Ω as regularization function
- γ_{size} as complexity regularization term, L as amount of leaves
- λ_{L2} as L2 regularization term
- y as ground truth and $\hat{y}$ as model prediction, x as input features

2.2.2 Multi-Layer Perceptron

An Multi-Layer Perceptron (MLP) (also known as an Artificial Neural Network) [29, p.333-387] is a supervised DL algorithm that processes the input feature matrix x via a stack of consecutive linear layers l, learning more complex features in the process. The output is then compared with a target vector to calculate a loss value. Each layer l consists of multiple tunable weights $W^{(l)}$ and a learnable bias term $b^{(l)}$. Equation 2.4 [67, p.218] describes how the propagation of an input through layer l is performed via matrix multiplication. An input from the previous layer $h^{(l-1)}$ (with $h^{(0)} = x$) is multiplied by the weight matrix $W^{(l)}$, while the bias $b^{(l)}$ is added, resulting in a representation $repr^{(l)}$. This representation is transformed with a non-linear activation function f_{act}, allowing the network to capture complex non-linear relationships between the feature matrix x and the targets (see Equation 2.5 [67, p.218]). The result is a transformed representation $h^{(l)}$, which serves as the input for the next layer. The last layer can have one (single-task learning) or multiple (multi-task learning [68]) output neurons, making MLPs suitable for single- and multi-task learning objectives. The backpropagation algorithm [69] is used to compute gradients of the loss with respect to the weights, while the gradient descent method [29, p.163] updates the weights by adjusting them in the direction that minimizes the loss.

$$\text{Pre-activation:} \quad repr^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)} \tag{2.4}$$

$$\text{Activation:} \quad h^{(l)} = f_{act}\left(repr^{(l)}\right) \tag{2.5}$$

- W as weight matrix and b as bias term at layer l
- $repr$ as pre-activation logits and h as logits
- f_{act} as activation function

2.2.3 Language Modeling

Long Short-Term Memory

A LSTM [48] is a special type of Recurrent Neural Network (RNN) [70], invented to overcome the issues of RNNs in capturing long-term dependencies in sequential data, effectively addressing the problems of vanishing and exploding gradients that can occur during training [71, 72]. The LSTM consists of a cell that incorporates different gating mechanisms to regulate the flow of information, specifically the input gate, forget gate, and output gate, allowing the network to retain relevant information over long sequences while discarding unnecessary information. The LSTM cell features two types of inputs besides the sequential input features $x_{(t)}$: a hidden state $h_{(t-1)}$, which carries information from prior time steps, and the cell state $c_{(t-1)}$, which acts as long-term memory for a timestamp t.

Figure 2.3 [73] outlines the propagation through a LSTM cell. A combination of the previous hidden state $h_{(t-1)}$ and the current sequence input x_t is used as input for every gating function. The forget gate f_t decides which elements should be discarded, while the input gate i_t determines what new information to store. The control gate $\tilde{c}_t$ [74] is used to update the cell state, and the output gate o_t controls which parts of the cell state are passed on to the next hidden state. The output of the forget gate is combined with the previous cell state $c_{(t-1)}$ using the Hadamard product to update the current cell state. Moreover, the Hadamard product of the outputs from the input gate i_t and control gate $\tilde{c}_t$ is added to the current cell state, yielding the updated cell state c_t. The output gate's result is elementwise multiplied by the new cell state c_t, which is previously transformed by a tanh activation, yielding the new hidden state h_t, enhancing the new hiddent state with long-term information.

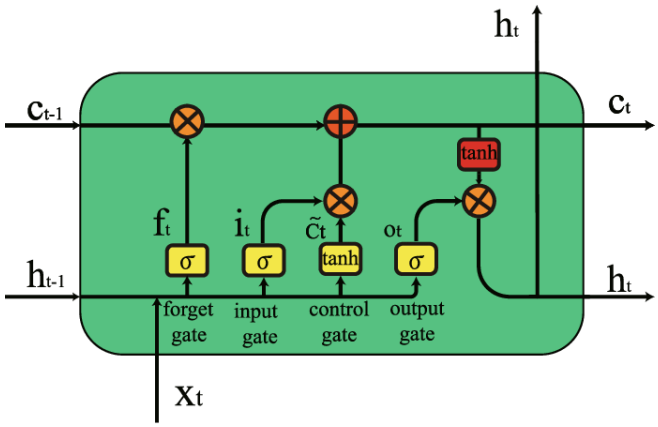

Figure 2.3: A LSTM cell takes a cell state c, a hidden state h, and a sequence input x as input and combines them involving different gating mechanisms to maintain a long-term and a short-term memory.

Equations 2.6-2.8 [29, p.611] refer to the forget gate f_t, input gate i_t and output gate o_t which follow the same form: A weight matrix W_h is multiplied by the hidden states $h_{(t-1)}$ and added to the product of the weight matrix W_x and the sequence input x_t along a specific additive bias term b. A sigmoid activation function σ transforms the result of the linear combination. The control gate $\tilde{c}_t$ has analogous weight matrices and bias terms compared to the gates: A weight matrix W_h, a weight matrix W_x, and a bias term b_c. The linear combination of the control gate is transformed with a tanh activation function instead of a sigmoid function σ. Equation 2.10 [29, p.611] refers to the calculation of c_t. The Hadamard product $\odot$ of the outputs of the forget gate f_t and the previous cell state $c_{(t-1)}$ is added to the Hadamard product of the results from the input gate i_t and control gate $\tilde{c}_t$. Lastly, Equation 2.11 [29, p.611] refers to the calculation of the hidden state h_t which is the Hadamard product of the output gate's o_t result and c_t transformed with a tanh activation function.

The LSTM maintains short-term memory through its hidden state $h_{(t)}$, which integrates the current sequence input $x_{(t)}$ and immediate context. Long-term memory is maintained via the cell state $c_{(t)}$, which propagates critical information across time steps and is regulated by the forget, input, control, and output gates. The forget gate selectively discards outdated information from $c_{(t-1)}$, the input gate and control gate add new relevant data, and the output gate modulates how c_t influences h_t [29, p.608-611].

$$f_t = \sigma(W_{hf} \cdot h_{(t-1)} + W_{xf} \cdot x_t + b_f) \quad \text{(Forget gate)} \tag{2.6}$$

$$i_t = \sigma(W_{hi} \cdot h_{(t-1)} + W_{xi} \cdot x_t + b_i) \quad \text{(Input gate)} \tag{2.7}$$

$$o_t = \sigma(W_{ho} \cdot h_{(t-1)} + W_{xo} \cdot x_t + b_o) \quad \text{(Output gate)} \tag{2.8}$$

$$\tilde{c}_t = \tanh(W_{h\tilde{c}} \cdot h_{(t-1)} + W_{x\tilde{c}} \cdot x_t + b_c) \quad \text{(Control gate)} \tag{2.9}$$

$$c_t = f_t \odot c_{(t-1)} + i_t \odot \tilde{c}_t \quad \text{(Updated cell state)} \tag{2.10}$$

$$h_t = o_t \odot \tanh(c_t) \quad \text{(Hidden state)} \tag{2.11}$$

- $\sigma(z) = \frac{1}{1+e^{-z}}$ [29, p.195] as sigmoid function
- $W_{\cdot}$ as weight matrices and $b_{\cdot}$ as bias terms
- h as logits
- t as timestamps
- x_t as sequence input of the current timestamp
- $\odot$ as Hadamard product

Self-Attention

Scaled Dot-Product Attention

An attention function specifies how much focus should be given to different parts of the input data when making predictions, allowing the model to dynamically weight the importance of various elements [29, p.620]. Self-Attention [10] refers to the case where every element of an input sequence attends to all other elements in the given sequence, thus learning contextualized relationships. Such a function can also be described as assigning a query vector and a set of key-value pairs of vectors to an output vector, resulting in a weighted sum determined by a compatibility function of the query. The query, key, and value vectors are projections of the input data, derived from learned linear transformations applied to the input embeddings.

Equation 2.12 [10] shows how self-attention can be defined as a scaled dot-product, where $Q, K \in \mathbb{R}^{d_k}$ and $V \in \mathbb{R}^{d_v}$ are the projected embeddings. The dot product of QK^T outputs the similarity between the query and all keys. The similarity is scaled by a factor of $\frac{1}{\sqrt{d_k}}$ to stabilize the gradients. A transformation with a softmax function produces the attention weights. Lastly, these attention weights are used to compute a weighted sum of the values.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.12}$$

- Q, K, V as query, key and value vector
- d_k as query and key dimension
- $\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ [67, p.204] as softmax function

Multihead Attention

Multihead Attention [10] refers to the application of the scaled dot-product attention multiple times. Therefore, the query, key, and value vectors are linearly projected h_{head} times with different learned projections.

Equation 2.14 [10] describes the multihead attention with h_{head} different heads. The matrices QW_i^Q, KW_i^K, VW_i^V are weight matrices which are specific to the attention head i. They project the original query, key, and value vectors into different subspaces, allowing each head to learn different aspects of the input data. Those heads are then concatenated and multiplied by the weight matrix W^O resulting in the final output.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_i, ..., \text{head}_{h_{head}})W^O \quad (2.13)$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.14)$$

- $W_.$ as weight matrices
- Q, K, V as query, key, and value vector
- h_{head} as amount of attention heads

Figure 2.4 [10] visualizes the schema of the multihead attention. The key, query, and value vectors are projected h_{head} times using different linear layers. Corresponding projections of the input key, query, and value vectors are then used for the scaled dot-product attention, resulting in h_{head} different weighted sums of the value vector, each capturing different representations of the input sequence. These representations are then concatenated and projected through a final linear layer.

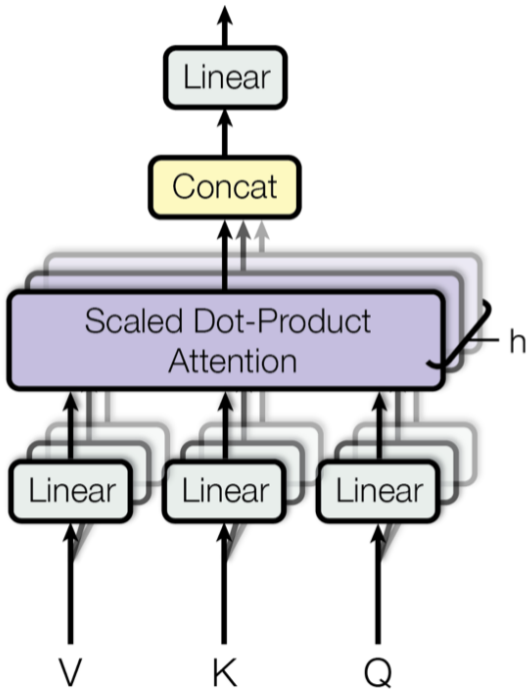

Figure 2.4: Multihead Attention consists of multiple running Scaled Dot-Product Attention layers.

Pre-training for Language Models

Pre-training [67, p.361-363] refers to a training method for ML and especially DL algorithms where the model is initially trained on different tasks before tackling the actual training objective. This can be done in a self-supervised or supervised manner, depending on the pre-training technique. For example, NLP models can be pre-trained with an autoregressive approach [75] by predicting the next element of a given sequence, which is suitable for generating decoder architectures like the Generative pre-trained transformer (GPT) architecture [1, 10]. Encoder-only architectures can be trained with a Masked Language Model (MLM), as seen in architectures such as Bidirectional Encoder Representations from Transformers (BERT) [3]. The MLM trains an encoder by randomly masking some of the input tokens with a special token $T_{[MASK]}$. These input tokens are predicted by the encoder, combined with a small MLP, based on the context, making pre-training with an MLM self-supervised.

In contrast to autoregressive or causal [76, p.85] approaches, the MLM-based pre-training strategy uses both left and right context, allowing the training of bidirectional models. To address the problem of pre-training and fine-tuning mismatch [3], where the $T_{[MASK]}$ token does not appear during fine-tuning or downstream task inference, the model occasionally replaces the masked token with a random token or leaves it unchanged during pre-training.

Figure 2.5 shows a schema of an MLM pre-training strategy, where T_i refers to the input token i, E_i refers to the embedding of token T_i, and R_i denotes the learned representation of T_i. Some tokens in the input sequence are randomly masked with a masking probability p, resulting in a partially masked sequence. In this example, token T_2 is masked and replaced with a special $T_{[MASK]}$ token, whose embedding is denoted as $E_{[MASK]}$. An encoder propagates the embeddings and outputs a contextualized representation R_i for each input embedding E_i. A classifier MLP is applied to the masked representation $R_{[MASK]}$, where the output size of the MLP matches the vocabulary size. The output logits are transformed with a softmax function to produce a probability distribution over all possible tokens. The model is trained to predict the original token T_2 at the masked position, where the ground truth corresponds to the token ID of the masked token. The MLM is optimized using the cross-entropy loss [77, p.206].

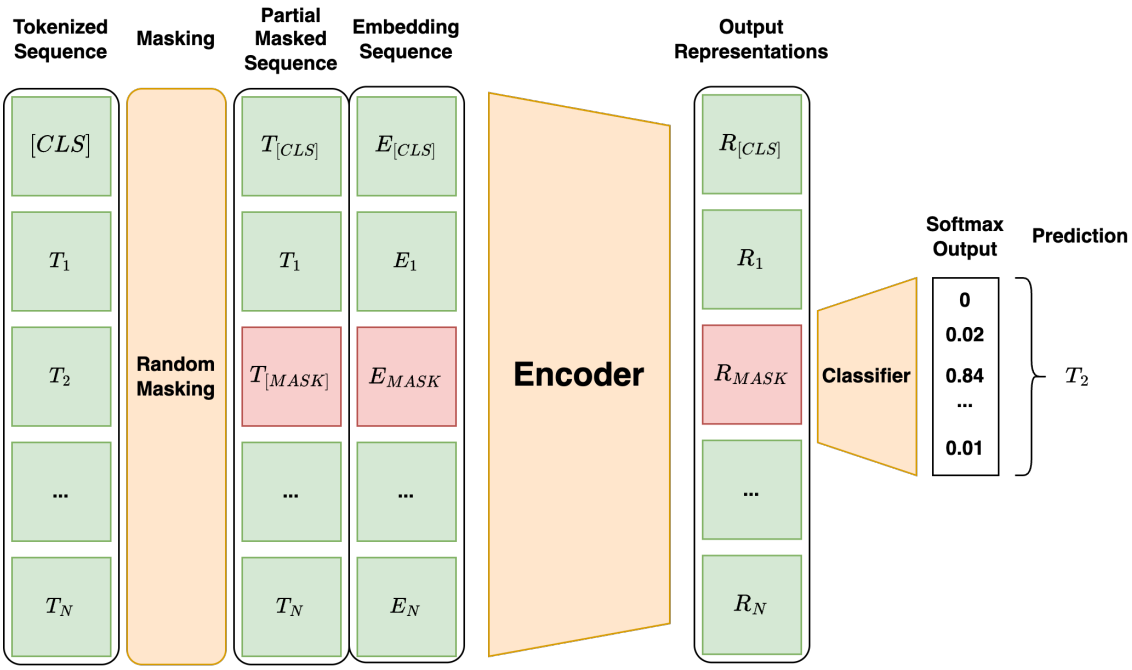

Figure 2.5: Schema of the MLM training process where a masked token is predicted based on contextualized representations (Self-made Figure).

2.2.4 Graph Neural Networks

GNNs [78] are neural networks specialized for graph-structured data and problems such as property prediction for an entire graph, specific nodes, or specific edges. GNNs have specialized layers designed to learn embeddings from nodes and edges of a graph, which are referred to as message-passing layers. Each node v consists of node features x_v, which serve as the starting point for the message-passing process. The features of each node v and its neighboring nodes $N(v)$ are embedded with an MLP and aggregated using an aggregation function (e.g., the mean value). Additionally, edge features can be incorporated into the message-passing process to enhance the representation of relationships between nodes. Afterward, the features of node v are updated based on the aggregated value, capturing local structures of the graph. This message-passing procedure can be repeated n times, aggregating information about the n_{th} closest neighbors of a node v. Many subtypes of GNNs exist, differing primarily in how they aggregate and update node representations. Finally, to obtain a graph-level embedding suitable for downstream tasks such as classification or regression, a readout function (e.g., sum, mean, or max pooling over node embeddings) is applied.

Equation 2.15 [47] outlines the aggregation procedure of a message-passing layer for a node v at layer l. An aggregation function is applied to capture the information from every embedding at layer $l-1$ of neighboring nodes $u \in N(v)$. Equation 2.16 [47] describes the combination of the aggregated information $a_v^{(l)}$ with the previous embedding $h_v^{(l-1)}$ of node v, allowing the model to integrate both the historical features of the node and the newly aggregated information to produce an updated embedding $h_v^{(l)}$. A combine function can be concatenation or any aggregation function.

$$a_v^{(l)} = \text{aggregate}^{(l)}(\{h_u^{(l-1)} : u \in N(v)\}) \tag{2.15}$$

$$h_v^{(l)} = \text{combine}^{(l)}(h_v^{(l-1)}, a_v^{(l)}) \tag{2.16}$$

- v, u as nodes, l as layer index
- a as aggregation
- h as logits
- $N(\cdot)$ as neighbour nodes of a node $\cdot$

Figure 2.6 [79] visualizes the concept of message-passing. An input graph (left) with six featurized nodes is used for the message-passing. The yellow node aggregates information from its neighbor nodes (red, green, blue) after the first message pass (1-hop neighbors). After the second message pass, the yellow node contains information from every node in the graph (2-hop neighbors). However, after the second message pass, the yellow node's updated features reflect the contributions from its closest neighbors and itself, allowing it to encapsulate both local and extended structural information about the graph.

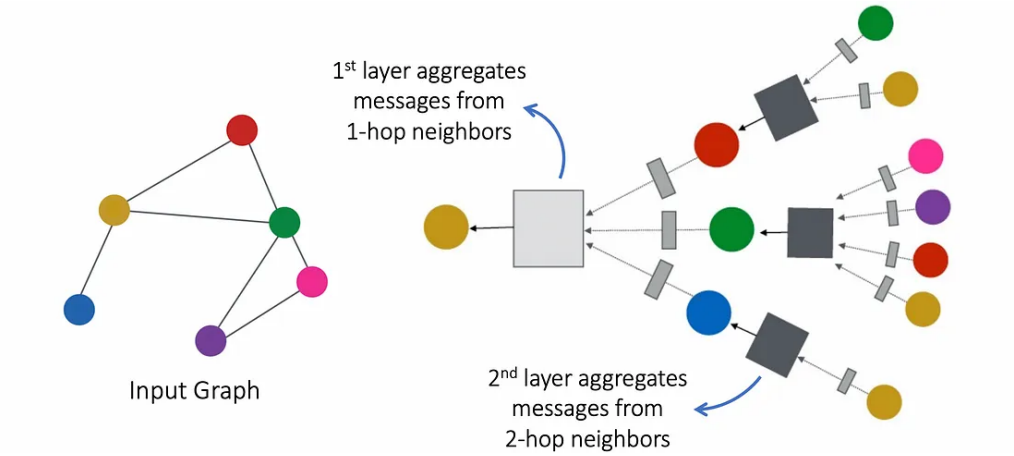

Figure 2.6: Schema of the message passing process of GNNs.

Graph Isomorphism Networks

A GIN [47] is a GNN [78] specialized in distinguishing between different graph structures. The aggregation function of a GIN is defined in Equation 2.17 [47]. The embedding $h_v^{(l)}$ for node v in layer l is calculated using an MLP, denoted as h_Θ. The input for this MLP consists of two terms: The first term $(1+\epsilon)\cdot h_v^{(l-1)}$ includes a hyperparameter ϵ, which regulates the importance of the embedding of node v from the previous layer $l-1$. The second term refers to the sum of the neighboring node embeddings $h_u^{(l-1)}$.

$$h_v^{(l)} = h_\Theta((1+\epsilon)\cdot h_v^{(l-1)} + \sum_{u\in N(v)} h_u^{(l-1)}) \tag{2.17}$$

- v, u as nodes, l as layer index
- h as logits
- $N(\cdot)$ as neighbor nodes of a node $\cdot$
- ϵ as importance regularization term
- MLP h_Θ

Equivariant Graph Neural Networks

An EGNN [31] is a GNN [78] that learns $E(n)$-equivariant graph representations. The term $E(n)$-equivariant refers to learned node coordinates that transform predictably under rotation, translation, and reflection (equivariance), while node features remain invariant to these operations. Therefore, an EGNN produces equivariant coordinates and $E(n)$-invariant features for a molecule, even when the input is transformed by rotation, translation, or reflection. The EGNN explicitly processes node coordinates (e.g., spatial positions) alongside node and edge features as input, updating them equivariantly across layers.

Equation 2.18 [31] describes the message-passing process of EGNNs. The message m_{vu} from node u to node v is produced by an MLP h_{θ_m}. This MLP takes as input the embedding $h_v^{(l)}$ from node v, the embedding $h_u^{(l)}$ from node u, the squared Euclidean distance between the coordinates of nodes u and v as $|c_v^{(l)} - c_u^{(l)}|^2$, and the edge attributes e_{vu}.

Equation 2.19 [31] refers to the calculation of the embedding h_v^{l+1} for a node v, which is computed by an MLP h_{θ_f}. This MLP takes the previous embedding $h_v^{(l)}$ and the aggregated messages from neighbors $\sum_{u \neq v} m_{vu}$ as input. The summation excludes $u = v$ to ensure updates depend only on neighboring nodes, not the node itself.

Equation 2.20 [31] shows how the coordinates $c_v^{(l+1)}$ are updated for node v. The update consists of the previous coordinates $c_v^{(l)}$ and a sum which is scaled by a factor of C_{scale}. The sum $\sum_{u \neq v}(c_v^{(l)} - c_u^{(l)})h_{\theta_c}(m_{vu})$ calculates the relative coordinate differences between a node v and its neighbors u weighted by a scalar calculated from an MLP h_{θ_c} by taking the message between nodes v and u as input. The coordinate update ensures $E(n)$-equivariance by combining relative displacements between nodes $(c_v^{(l)} - c_u^{(l)})$ which transform linearly under rotations, translations, or reflections with scalar weights yielded by h_{θ_c}. These scalars are derived from invariant inputs: the message m_{vu} (Equation 2.18) which depends on the squared Euclidean distance $\|c_v^{(l)} - c_u^{(l)}\|^2$ (invariant to $E(n)$) and node features $h_v^{(l)}$, ensuring h_{θ_c} is also invariant. Thus, the coordinate update $c_v^{(l+1)}$ transforms predictably under $E(n)$, while node features $h_v^{(l+1)}$ (Equation 2.19) remain invariant, as they aggregate messages computed from invariant quantities.

$$\text{Message:} \quad m_{vu}^{(l)} = h_{\theta_m}\left(h_v^{(l)}, h_u^{(l)}, \|c_v^{(l)} - c_u^{(l)}\|^2, e_{vu}\right) \tag{2.18}$$

$$\text{Feature Update:} \quad h_v^{(l+1)} = h_{\theta_f}\left(h_v^{(l)}, \sum_{u \neq v} m_{vu}\right) \tag{2.19}$$

$$\text{Coordinate Update:} \quad c_v^{(l+1)} = c_v^{(l)} + C_{scale} \sum_{u \neq v}(c_v^{(l)} - c_u^{(l)})h_{\theta_c}(m_{vu}) \tag{2.20}$$

- v, u as nodes, l as layer index
- h as logits
- $h_{\theta_\cdot}$ as MLPs
- $e_\cdot$ as edge attributes
- $c_\cdot$ as coordinates of node $\cdot$
- C_{scale} as scaling factor
- $m_{\cdot\cdot}$ as message between two nodes $\cdot\cdot$

Pre-training for Graph Neural Networks

GNNs can be pre-trained using various strategies that focus on different graph elements, such as nodes, edges, a combination of nodes and edges, or entire graph structures [80], combined with different training objectives like graph-property prediction, structural-similarity prediction, or node/edge attribute prediction [81]. Pre-training based on node/edge attribute prediction can be performed in a self-supervised manner, analogous to NLP models [3].

In this approach, node/edge attributes are masked with a special mask indicator, similar to the special mask token $T_{[MASK]}$ used in the NLP domain. The pre-trained GNN predicts the masked attributes based on neighboring structures, yielding the respective node/edge embedding. This embedding is then used in combination with an MLP to predict the node/edge attribute.

Figure 2.7 [81] illustrates a schematic self-supervised pre-training process for GNNs based on edge attribute prediction. The input graph consists of three nodes v_1, v_2, v_3, where all nodes are connected to each other. A masking model randomly masks edges between the nodes with a special mask indicator. Then, a trainable graph encoder processes the masked graph as input and outputs the corresponding embedding H. A respective decoder is trained to predict the missing edge attributes.

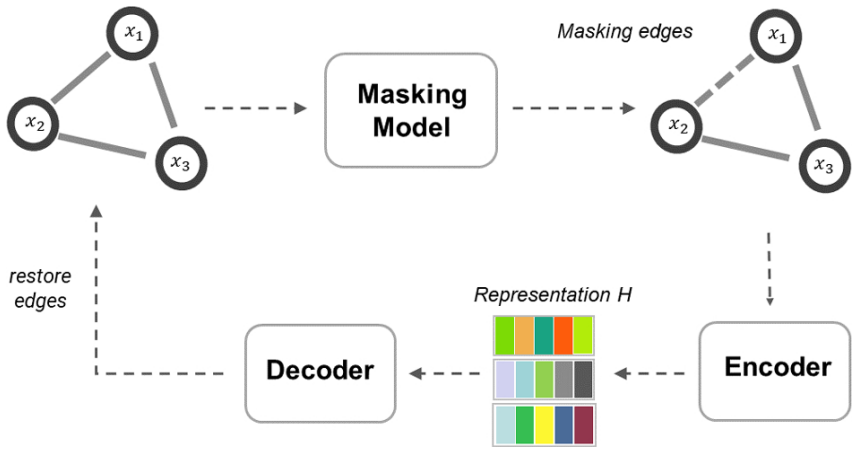

Figure 2.7: Schema of a self-supervised GNN pre-training based on masked edge attribute prediction.

2.2.5 Variance-Invariance-Covariance Regularization

VICReg [41] is a self-supervised representation learning method originating from the domain of CV. VICReg trains different encoders by maximizing the agreement between embedding vectors based on differently data-augmented [82] views of the same image, leading to information exchange across the encoders. Representation collapse is a known problem in representation learning, where an encoder produces a constant, non-informative representation that minimizes the loss but is impractical for downstream tasks [52]. VICReg prevents this collapse by using two regularization terms applied to the embeddings. One term maintains the variance across embedding dimensions to prevent total representation collapse, while the other decorrelates each pair of variables within an embedding to prevent dimensional collapse.

VICReg does not rely on established but expensive techniques to prevent the collapse problem, such as large batch sizes [42], memory banks [43], momentum encoders [44], quantization-based approaches [45], or a stop-gradient operation [46]. It even surpasses these techniques, as well as its supervised counterpart, in downstream performance tasks [41]. Furthermore, VICReg is highly flexible since the encoder and projector do not need to follow a siamese shared-weight architecture [83].

Figure 2.8 [41] illustrates how VICReg operates with a shared-weight encoder and projector for two inputs. A batch of images I is used as input data, where different data augmentation transformations $t_{aug}, t'_{aug} \in T_{aug}$ (in the Figure referred to as t and t') are applied to create different views X, X' of the same image. The encoders produce different representations Y, Y' for the same image based on the different views X, X'. Each representation is further processed with a projection MLP, resulting in the embeddings Z, Z'. A specific loss function is applied to Z, Z' to maintain the variance across a batch of embeddings, minimize the distance between the embeddings of the same image, and decorrelate variables within each embedding.

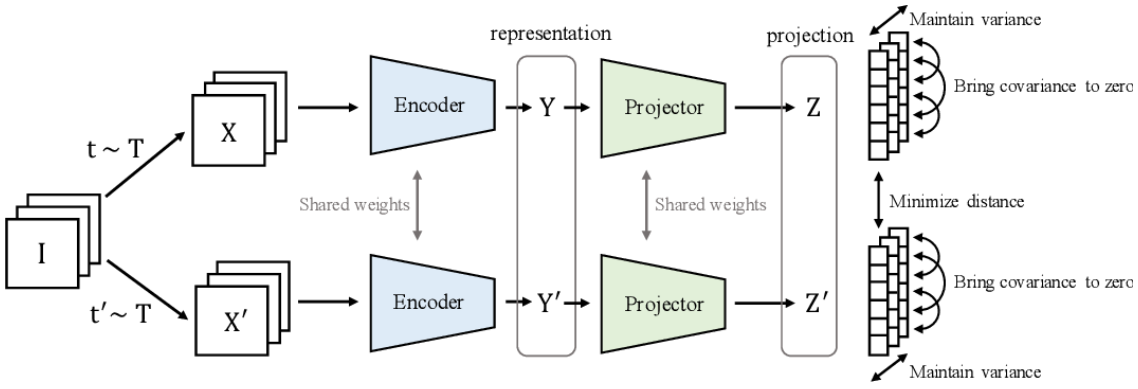

Figure 2.8: Example architecture for VICReg based on weight sharing.

Variance Loss

The main objective of the variance loss is to maintain the variance across a batch of embeddings. Equation 2.21 [41] defines the variance function $Std(x, \epsilon_{std})$ which computes the standard deviation of a vector x which is stabilized with a stability term ϵ_{std}. Equation 2.22 [41] refers to VICRegs variance loss function ℓ_v. A batch of embeddings Z serves as the input for ℓ_v. The embedding vector z^j corresponds to the j-th dimension from every sample within a batch. The loss function calculates the standard deviation $Std(z^j)$ for every embedding dimension d across all embeddings in Z. A hinge function ensures that only positive values are summed while a hyperparameter γ can be used to set a minimum value for the function S to prevent a representation collapse. Lastly, the loss is aggregated using the mean value.

$$Std(x, \epsilon_{std}) = \sqrt{Var(x) + \epsilon_{std}} \tag{2.21}$$

- x as vector
- ϵ_{std} as stability term

$$\ell_v(Z) = \frac{1}{d} \sum_{j=1}^{d} max(0, \gamma - Std(z^j, \epsilon_{std})) \tag{2.22}$$

- γ as threshold parameter
- z^j as embedding vector
- d as length of the embedding vectors
- Std as standard deviation with stability term ϵ_{std}

Covariance Loss

The covariance loss prevents redundant representations of the embedding dimensions by decorrelating the variables of the embeddings, thereby enforcing each embedding dimension d to capture different information. Equation 2.23 [41] shows the calculation of the covariance matrix Cov, where $\bar{z}$ represents the mean value of an embedding vector z. The covariance loss ℓ_c is defined in Equation 2.24 [41], which sums up the off-diagonal terms of the covariance matrix Cov, corresponding to the correlation between the embedding dimensions. The loss is scaled by the number of dimensions d. The decorrelation at the embedding level also decorrelates the representations at the encoder level [41].

$$Cov(Z) = \frac{1}{n-1}\sum_{i=1}^{n}(z_i - \bar{z})(z_i - \bar{z})^T \tag{2.23}$$

$$\ell_c(Z) = \frac{1}{d}\sum_{i \neq j}[Cov(Z)]_{i,j}^2 \tag{2.24}$$

- Z as a batch of embedding vectors $z \in \mathbb{R}^d$
- d as dimension
- $\bar{z}$ as mean value of an embedding vector z

Invariance Loss

An invariance loss ℓ_s is defined to ensure that the embedding batches Z and Z' of two differently augmented images are similar to each other in the latent space. Equation 2.25 [41] shows the invariance loss as the average mean-squared Euclidean distance between the different embedded views z_i and z_i', enforcing them to align.

$$\ell_s(Z, Z') = \frac{1}{n}\|z_i - z_i'\|_2^2 \tag{2.25}$$

- Z, Z' as embedding batches
- z_i, z_i' as embedding vectors

Loss Function

Equation 2.26 [41] defines the VICReg loss function $\mathcal{L}_{VICReg}$ for the embeddings Z, Z' consisting of equations 2.22, 2.24, and 2.25. Thus, the loss function combines the previously described losses and weights them according to their importance using the hyperparameters λ, μ and ν.

$$\mathcal{L}_{VICReg}(Z, Z') = \lambda\ell_s(Z, Z') + \mu[\ell_v(Z) + \ell_v(Z')] + \nu[\ell_c(Z) + \ell_c(Z')] \tag{2.26}$$

- λ as invariance regularization term
- μ as variance regularization term
- ν as covariance regularization term
- Z, Z' as embeddings of two differently augmented images

2.3 Statistics

2.3.1 Metrics

Mean Absolute Error

The Mean Absolute Error (MAE) refers to the mean average error based on the $p1$-norm [84]. Equation 2.27 [84] describes the error metric based on n data points. The error is calculated as the absolute value of the difference between the ground truth y_i and the prediction $\hat{y}_i$. Lower MAE scores correspond to a better-fitted model compared to a model that yields high MAE scores.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.27}$$

- n datapoints
- y as ground truth and $\hat{y}$ as prediciton

Coefficient of Determination

The Coefficient of Determination (also known as R2) quantifies the proportion of variance in the target variable that is explained by a regression model [85, p.556]. A higher R2 value indicates that the model captures a greater amount of the variability in the data. The R2 score is always a value between 0 and 1.

Equation 2.28 [85, p.556] defines the R2 score as the ratio of the sum of squares explained by the regression model to the total sum of squares. The numerator measures how much the model's predictions deviate from the mean, reflecting the explained variance, while the denominator measures the total variance in the observed data.

$$R^2 = \frac{\text{Regression sum of squares}}{\text{Total sum of squares}} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{2.28}$$

- n datapoints
- y as ground truth and $\hat{y}$ as prediciton
- $\bar{y}$ as mean ground truth value

2.3.2 Analysis of Variance

One-way ANOVA

The Analysis of Variance (ANOVA) is a statistical method used to test for significant differences among the means of different groups by partitioning the total variance into between-group and within-group components [86, p.161–218]. The F-statistic evaluates the ratio of the between-group variance to the within-group variance, where a higher value indicates greater evidence against the null hypothesis of equal group means.

Equation 2.33 [86, p.175] defines the F-statistic as the ratio of the mean square between groups variability (MS_b) to the mean square within groups variability (MS_w). The between-group sum of squares (SS_b) and within-group sum of squares (SS_w) are calculated deviations where SS_b captures the deviation between the group means, while SS_w measures the variation within each group. The MS_b and MS_w are derived by dividing SS_b and SS_w by their respective degrees of freedom.

$$SS_b = \sum_{g=1}^{G} n(\bar{X}_g - \bar{X})^2 \tag{2.29}$$

$$SS_w = \sum_{g=1}^{G} \sum_{i=1}^{n} (X_{ig} - \bar{X}_g)^2 \tag{2.30}$$

$$MS_b = SS_b/(G-1) \tag{2.31}$$

$$MS_w = SS_w/(G \cdot (n-1)) \tag{2.32}$$

$$F = \frac{\text{explained variance}}{\text{not explained variance}} = \frac{MS_b}{MS_w} \tag{2.33}$$

- G groups each containing n observations
- $\bar{X}_g$ as group-specific means of a random variables X_g
- $\bar{X}$ as grand mean value
- X_{gi} as the i-th realization of a random variable X_g
- MS_b and MS_w as the mean squares for the between and within group

Repeated Measures ANOVA

ANOVA for repeated measures extends the one-way design to handle within-subjects factors, where the same participants are measured under multiple conditions or time points [87, p.461–499]. This method partitions the total variance into between-subject variability, within-subject treatment effects, and residual error. The F-statistic evaluates whether systematic differences across conditions within subjects exceed the expected random variability.

Equation 2.40 [87, p.469] defines the F-statistic as the ratio of the treatment mean square (MS_{treat}) to the error mean square (MS_{error}). The treatment sum of squares (SS_{treat}), calculated via Equation 2.34 [87, p.467], quantifies condition-specific deviations from the grand mean. The total variability (SS_{total}) is computed as the sum of squared deviations from the grand mean (Equation 2.35 [87, p.467]), while between-subjects variability (SS_{sub}) isolates differences attributable to individual subjects (Equation 2.36 [87, p.467]). The error sum of squares (SS_{error}), derived in Equation 2.37 [87, p.467], represents residual variability after accounting for subject and treatment effects. Mean squares are obtained by dividing sums of squares by their respective degrees of freedom (Equations 2.38 [87, p.463] and 2.39 [87, p.463]).

$$SS_{\text{treat}} = n \sum_{g=1}^{G} (\bar{X}_{\cdot g} - \bar{X})^2 \tag{2.34}$$

$$SS_{\text{total}} = \sum_{i=1}^{n} \sum_{g=1}^{G} (X_{ig} - \bar{X})^2 \tag{2.35}$$

$$SS_{\text{sub}} = G \sum_{i=1}^{n} (\bar{X}_{i\cdot} - \bar{X})^2 \tag{2.36}$$

$$SS_{\text{error}} = SS_{\text{total}} - SS_{\text{sub}} - SS_{\text{treat}} \tag{2.37}$$

$$MS_{\text{treat}} = SS_{\text{treat}}/(G-1) \tag{2.38}$$

$$MS_{\text{error}} = SS_{\text{error}}/((n-1)(G-1)) \tag{2.39}$$

$$F = \frac{\text{treatment variance}}{\text{error variance}} = \frac{MS_{\text{treat}}}{MS_{\text{error}}} \tag{2.40}$$

- n subjects measured at G groups (conditions)
- X as random variable
- $\bar{X}$ as grand mean, $\bar{X}_{i\cdot}$ as subject mean, $\bar{X}_{\cdot g}$ as group mean
- X_{ig} as a realization of the random variable $X_{\cdot g}$ for the i-th subject and group g

2.3.3 Tukey Honestly Significant Difference Post-Hoc Test

The Tukey Honestly Significant Difference Post-Hoc test (Tukey test) is a post-hoc procedure used after a significant ANOVA to identify which specific group means differ across multiple comparisons [87, p.391–393]. It employs the studentized range distribution [87, p.389–391] to compute critical values for pairwise mean differences, ensuring robustness against Type I error inflation.

Equation 2.41 [87, p.392] defines the minimum significant difference as a function of the within-group mean square (MS_w), the number of groups (G), and the sample size per group (n). Pairwise differences exceeding this threshold are considered statistically significant.

$$q_{\alpha,tr,df}\sqrt{\frac{MS_w}{n}} \tag{2.41}$$

- G groups with equal sample size n per group
- MS_w as within group mean from one-way ANOVA (Equation 2.30)
- $df = G(n-1)$ as degrees of freedom for the error term
- $q_{\alpha,tr,df}$ as critical value from the studentized range distribution at significance level α for tr treatments

Chapter 3

Methods

3.1 Data

3.1.1 Pre-training Dataset

Beaini et al. [88] gathered and combined existing public molecular datasets to develop new datasets that are appropriate for training large-scale molecular foundation models. Additionally, more high quality experimental labels were added to provide relevant compound properties for supervised pre-training approaches. Three different datasets are introduced which consists of smaller datasets from different molecular related domains. For the model pre-training in this experiment the largemix dataset is used due to its size and mixture of quantum and biochemical properties. The largemix dataset involves four different datasets: PCQM4M_G25_N4, PCBA_1328, L1000_VCAP, and L1000_MCF7.

PCQM4M_G25_N4

This dataset originates from the PubChemQC project [89], where quantum properties of 3.8M (million) small molecules were computed. The molecules come from PubChem [90], a database that contains chemical structures and bioactivity data. *Beaini et al.* use given quantum properties like the HOMO-LUMO gap [91] and compute additional 3D descriptors based on the precomputed quantum properties, resulting in 25 regression task labels. The labels of the regression tasks are standardized using z-scores to improve training stability.

PCBA_1328

The PCBA_1328 dataset comprises 1.56M molecules annotated with 1328 distinct binary classification tasks derived from PubChem bio-assays [90]. For that, PubChem's bio-assay repository was systematically parsed by selecting assays that met stringent criteria: each must include over 6000 molecules labeled as either *Active* or *Inactive*, with at least 10 examples in both categories. The found molecular identifiers were converted to canonical SMILES to unify the assays into a single dataset. Bio-assays are experimental protocols that measure a molecule's biological activity (e.g., binding affinity, inhibition) against specific targets, such as proteins or cellular pathways. In this context, *Active* denotes molecules exhibiting the desired biological effect (e.g., inhibiting a disease-related enzyme), while *Inactive* refers to those lacking such activity [90].

L1000_VCAP/MCF7

The data in the L1000 datasets are from the LINCS L1000 database [92], containing high-throughput transcriptomics data. VCAP and MCF7 refer to prostate and human breast cancer cell lines, respectively, where 978 landmark genes [93] were screened with around 30.000 perturbations. These perturbations originated from small drug-like molecules that were added to the cell lines in order to investigate changes in gene expression. Therefore, the 978 landmark genes describe 978 ranked classification tasks per dataset. Both L1000 datasets are excluded from the largemix dataset since the proposed datasets can degrade performance for certain downstream tasks [94].

Table 3.1 summarizes the construction of the used pre-training data, highlighting their diversity in data types, tasks, and scale. The PCQM4M_G25_N4 dataset contains 3.8 million (M) molecules annotated with 25 quantum mechanical properties for regression (R) tasks. PCBA_1328 contains 1.6M molecules across 1328 bio-assay based classification (C) tasks, capturing binary activity labels (active/inactive) against diverse biological targets. The transcriptomics datasets L1000_VCAP and L1000_MCF7 are smaller in scale (15.000 and 12.000 molecules, respectively) but include 978 multi-task ranked classification (RC) labels each, derived from gene expression profiles in cancer cell lines. The L1000 data are not considered due to their noisy behavior regarding downstream tasks [94]. Therefore, the pre-training dataset consists of 5.4M molecules with 25 regression tasks and 1328 binary classification tasks, capturing quantum and biochemical properties of compounds. Intersecting SMILES between the pre-training data and the downstream task data are removed in favor of the downstream dataset to prevent bias due to previously seen molecules.

Dataset	Type	Task	# mol.	Labels	Used
PCQM4M_G25_N4	Quantum	R	3.8M	25	✓
PCBA_1328	Bio-assays	C	1.6M	1328	✓
L1000_VCAP	Transcriptomics	RC	15k	978	✗
L1000_MCF7	Transcriptomics	RC	12k	978	✗

Table 3.1: Properties of the pre-training data showing the used datasets, corresponding dataset domain, task type, amount of molecules and amount of labels.

3.1.2 Downstream Dataset

The public ADME dataset from *Fang et al.* [95] serves as a downstream dataset to compare the different experiments, as the named dataset contains publicly available high-quality ADME data [96]. Originally, *Fang et al.* compiled a comprehensive ADME dataset by curating and standardizing experimental measurements over a span of two years. They published a comparable ADME dataset consisting of 3516 unique compounds, which are similar to the collected dataset based on pairwise compound similarity, spanning an equivalent chemical space. The dataset contains molecules from commercial vendor libraries and pre-clinical compounds, with six different endpoints: Human Plasma Protein Binding (hPPB), Rat Plasma Protein Binding (rPPB), Solubility (Sol), Rat Liver Micrososomal Stability (RLM), Human Liver Micrososomal Stability (HLM), and Multidrug Resistance Protein 1 Efflux Ratio (MDR1-ER). The curated and preprocessed version from Polaris [97] is used in this experiment.

Endpoints

All endpoints are regression tasks for different ADME properties, showing how a drug behaves in the human body. For example, hPPB and rPPB both describe the plasma protein binding [98, p.187-195] property of a drug, indicating how strongly a molecule binds to proteins in blood. A low binding affinity can limit the drug's ability to interact with target proteins, a critical factor in the distribution phase of ADME. Plasma protein binding is measured as a percentage of $\frac{\text{Bound Drug}}{\text{Total Drug}} \times 100$, where the numerator represents the amount of drugs bound to proteins, while the denominator includes the amount of bound and unbound drug. The endpoints RLM and HLM [98, p.145-170] measure how quickly a drug is metabolized by liver enzymes (metabolism in ADME). This is quantified using the half-life $t_{1/2}$, which describes the time required for 50% of the drug to be metabolized. A short half-life indicates rapid metabolism, which reduces the drug's duration of action, leading to a higher dosing frequency.

The Sol of a drug quantifies how much a drug dissolves in water, describing the absorption properties of compounds. Sol [98, p.56-86] can be expressed as a fraction of dissolved mass to volume: $\frac{\text{Mass Dissolved}(\mu g)}{\text{Volume}(mL)}$, where low solubility values lead to poor absorption. MDR1-ER [98, p.111-121] evaluates whether a drug is actively expelled from cells by a specific protein, the MDR1 transporter, which influences a compound's absorption and distribution properties. MDR1-ER measures how strongly a drug is pushed out of cells versus how well it enters a cell. It's calculated by comparing the drug's exit speed to its entry speed, which is called the efflux ratio (ER). A higher value means that a drug is less effective due to the cell's active rejection system. The particular equations of the in vitro assays used by *Fang et al.* [95] for the endpoints hPPB, rPPB, RLM, HLM, and MDR1-ER can be found in Appendix A.1.1 as Equations A.1-A.4.

Dataset Properties

The ADME endpoint values are transformed using the decadic logarithm operation for stability purposes. Table 3.2 shows the number of compounds per endpoint, as well as the mean, standard deviation, minimum, quartiles, and maximum value of each endpoint. The count column shows that both plasma protein binding endpoints (hPPB and rPPB) have far fewer data points compared to the four other endpoints. Labels for the endpoints HLM and RLM are provided for around 87% of all given ADME compounds, while 75% of all compounds have a label for the endpoint MDR1-ER and 62% of all molecules provide a Sol label. Labels for the endpoints rPPB and hPPB are provided for only 6% of all molecules.

Task	count	mean	std	min	25%	50%	75%	max
LOG HLM CLint	3082	1.32	0.62	0.68	0.68	1.20	1.80	3.37
LOG RLM CLint	3049	2.26	0.75	1.03	1.69	2.31	2.84	3.97
LOG MDR1-MDCK ER	2640	0.40	0.69	-1.16	-0.16	0.15	0.90	2.73
LOG HPPB	194	0.77	0.85	-1.59	0.17	0.87	1.50	2.00
LOG RPPB	168	0.76	0.80	-1.64	0.23	0.78	1.38	2.00
LOG SOLUBILITY	2173	1.26	0.68	-1.00	1.15	1.54	1.69	2.18

Table 3.2: Descriptive statistics for the unsplitted downstream dataset containing six ADME endpoints (tasks).

The endpoints hPPB and rPPB are removed due to limited data availability. The remaining data are split into a train and test set based on compound similarity clusters. The Butina [64] algorithm is used to create clusters based on compound similarities measured by the Tanimoto distance, with a distance cutoff threshold of $d_{\text{cutoff}} = 0.65$ (equivalent to a Tanimoto similarity of 0.35) [99]. The corresponding molecular fingerprints are calculated using the Morgan fingerprint [61] with 1024 bits and a radius of 2. After cluster calculation, the data is sorted based on cluster frequency, with compounds assigned to the most frequent clusters listed first. The dataset is split so that the 80% most frequent clusters are assigned to the training set, while compounds from the 20% least frequent clusters are assigned to the test set. This ensures that the test dataset contains the most structurally dissimilar compounds, challenging the model's generalization ability, unlike popular alternatives such as random splits or scaffold-based approaches [100, 101]. Furthermore, the test split contains all singletons that were not assigned to any similarity clusters. The distribution of the train and test splits can be found in Appendix A.1.1 as Table A.1 and Table A.2.

The endpoint data is combined as a multi-task ADME dataset, since multi-tasking is a known and efficient way to boost ADME prediction performance for neural network-based models [102, 103]. Consequently, compounds can contain null values for tasks where no label was provided. In addition, the datasets are also modeled as single-task datasets using XGB to provide a performance baseline. Each single-task dataset consists of one target column per endpoint, while both the single-task and multi-task models involve the same compounds in their train and test splits, with the number of tasks being the only difference.

3.1.3 Preprocessing

Tabular Representation

The tabular featurization strategy from *Fang et al.* [95] is implemented to represent molecules as a combination of molecular descriptors and fingerprint bits. Accordingly, a Morgan fingerprint [61] with 1024 bits and a radius of 2 is used to assess local features, and 316 *RDKit* [104] descriptors are used as global features, resulting in a feature vector of length 1340 per molecule.

Graph Representation

The graph representation is calculated using node and bond features per molecule with Chemprop [26] graph featurizers. The following atom features are used for every atom in a molecule: atom symbol, node degree, formal charge, chiral type, number of hydrogen atoms, atom mass, aromaticity information, and hybridization type, resulting in a feature vector of length 72 consisting of 71 one-hot encodings of categorical features and 1 continuous feature (atom mass). Atom bonds are featurized using a one-hot encoded edge feature vector of length 14, involving bond information such as bond type, conjugation, whether a bond is part of a ring structure, and information about the stereochemistry.

Figure 3.1 [105] shows a schema of a featurized molecule which is constructed as a graph $\mathcal{G}$ with edges $\mathcal{E}$ and vertices $\mathcal{V}$ where $|\mathcal{V}|$ corresponds to the amount of atoms and $|\mathcal{E}|$ describes the amount of bonds betweem two atoms. Each vertex consists of its own features as a feature vector of length d_v, describing each atom i as a vector $v_i \in \mathbb{R}^{d_v}$. Each edge between two vertices i and j can also be featurized. Therefore, an atom bond can be expressed as a vector $e_{i,j} \in \mathbb{R}^{d_e}$ of length d_e.

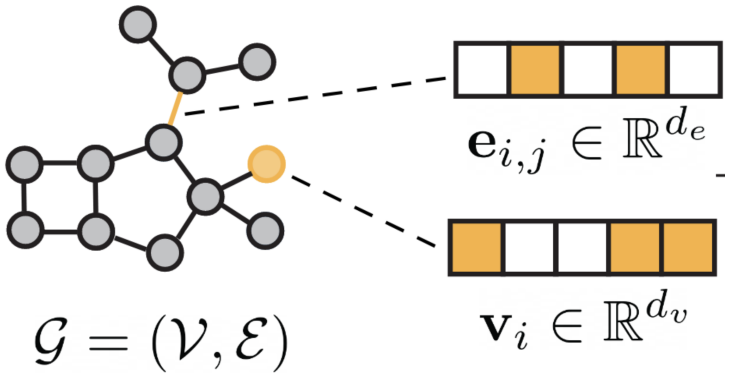

Figure 3.1: Schema of a molecular graph where each atom v_i and bond $e_{i,j}$ consist of it's own features.

Conformer Representation

A conformer is a distinct 3D spatial arrangement of a molecule's atoms, arising from rotations around single bonds and representing thermally accessible geometries, often separated by low rotational barriers [106, p.22, p.41-44]. Hence, the conformer representation uses the same node and edge features as the graph representation but augments them with 3D coordinates. The 3D coordinates are calculated using a distance geometry-based method [107], which incorporates an experimental torsion angles method named Experimental-Torsion Distance Geometry (ETKDG) [108]. A multi-step

approach is used to calculate the 3D coordinates, since these coordinates are not obtainable for every molecule due to structural ambiguities and algorithmic limitations [109].

Algorithm 2 shows this multi-step approach as pseudocode. The procedure to generate the coordinates takes a SMILES string and an integer (max_attempts) as input. First, a SMILES string is converted into a molecule object, which is a structured representation of a chemical compound as a graph. Then, hydrogen atoms are explicitly added to the molecule because their positions affect bond angles, molecular volume, and stability [109] when generating 3D coordinates. Afterwards, the ETKDG algorithm [108] is applied to generate the 3D coordinates. If the first attempt fails, it retries with randomized starting positions. The number of repetitions is parameterized by the argument max_attempts which was set to 100 in this work. Upon successful completion, the 3D positions are refined using a method called Merck molecular force field (MMFF) [110] to adjust the structure to a more realistic configuration with a low-energy state. If the generation of the 3D coordinates fails (e.g., the molecule is too large or complex [111]), a flat 2D representation is calculated instead. Lastly, the coordinates are returned and added as a trainable graph feature.

Algorithm 2 Conformer Generation

```
1:  procedure GenerateConformers(SMILES, max_attempts)
2:      mol ← Convert SMILES to molecule                          ▷ Initial parsing
3:      Add hydrogens to mol                                      ▷ Explicit hydrogen modeling
4:      3D_success ← False
5:      for attempt ← 1 to max_attempts do
6:          3D_success ← Attempt 3D coordinate generation (EmbedMolecule)
7:          if 3D_success then
8:              break
9:          else
10:             Randomize starting positions                      ▷ Retry with new initialization
11:         end if
12:     end for
13:     if 3D_success then
14:         Energy minimization (MMFF)                            ▷ Refine geometry
15:         coords ← mol.Get3DCoordinates()                       ▷ Retrieve optimized 3D coordinates
16:     else
17:         Generate2DCoordinates(mol)                            ▷ Fallback to 2D representation
18:         coords ← mol.Get2DCoordinates()                       ▷ Retrieve 2D coordinates
19:     end if
20:     return coords                                             ▷ Return 3D or 2D coordinates based on success
21: end procedure
```

Text Representation

The text representation is created by implementing a SMILES tokenizer, which tokenizes [76, p.35] the SMILES strings at the character level. Thus, each syntactical element of the SMILES string is mapped to a unique integer representing a token to build up a vocabulary of the SMILES language. Furthermore, special tokens for sequence padding, unknown characters, and a Classification (CLS) token are added. A CLS token is a special token typically used to aggregate information from the entire sequence to solve a certain downstream task. Lastly, each token corresponds to an embedding vector $E \in \mathbb{R}^d$ with an embedding dimension d of 128.

Figure 3.2 [112] outlines the featurizing process of a molecule suitable for training language models. A molecule is expressed as a SMILES string and tokenized into unique elements, which are mapped to unique integers. These integers are the token IDs, which serve as indices for an embedding layer table containing trainable dense representations of a token [29, p.506-507].

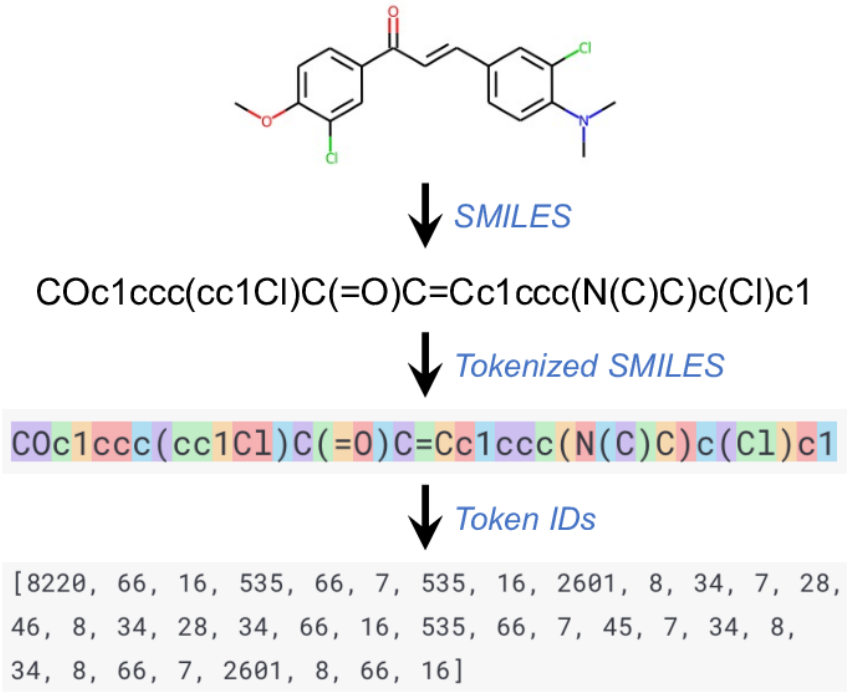

Figure 3.2: Schema of SMILES preprocessing for language models.

3.2 Architecture Details

3.2.1 Hyperparameter Tuning

Optuna [113] is used as a hyperparameter optimization framework to find an optimized model architecture for the XGB, GIN, EGNN, and LSTM models based on the training split of the ADME dataset. For this, a Bayesian [114] sampling algorithm is used, the Tree-structured Parzen Estimator (TPE) [115]. The TPE algorithm iteratively models the hyperparameter space by distinguishing high-performing configurations from poor ones, prioritizing regions where optimal parameters are statistically likely. This enables efficient exploration of the search space for all four architectures, balancing exploration of new configurations and exploitation of known high-performing trials. The performance is measured by the mean validation MAE based on a similarity 5-fold cross-validation.

Figure 3.3 outlines the hyperparameter tuning schedule. The training split of the downstream dataset is used for tuning. The validation split is generated using the Butina clustering algorithm [64] to validate a hyperparameter configuration on a 5-fold cross-validation based on molecular similarity. This approach ensures that structurally dissimilar compounds are partitioned into the test fold, minimizing bias from analogous chemical structures during evaluation. Each model has its own pre-defined hyperparameter space from which the TPE algorithm samples the model configuration by choosing random parameters at first. For each hyperparameter tuning trial, 5 model instances are created with the same sampled hyperparameters. Each model instance is trained on a different fold out of the 5-fold cross-validation. The MAE on the test fold is aggregated using the mean value. The mean validation MAE is assigned to the current hyperparameter set. After n initial trials, the TPE algorithm leverages this data to construct a probabilistic model of high-performing configurations, prioritizing hyperparameters that statistically minimize the validation MAE. The hyperparameter set that minimizes the validation MAE the most is used as the model configuration for all further experiments.

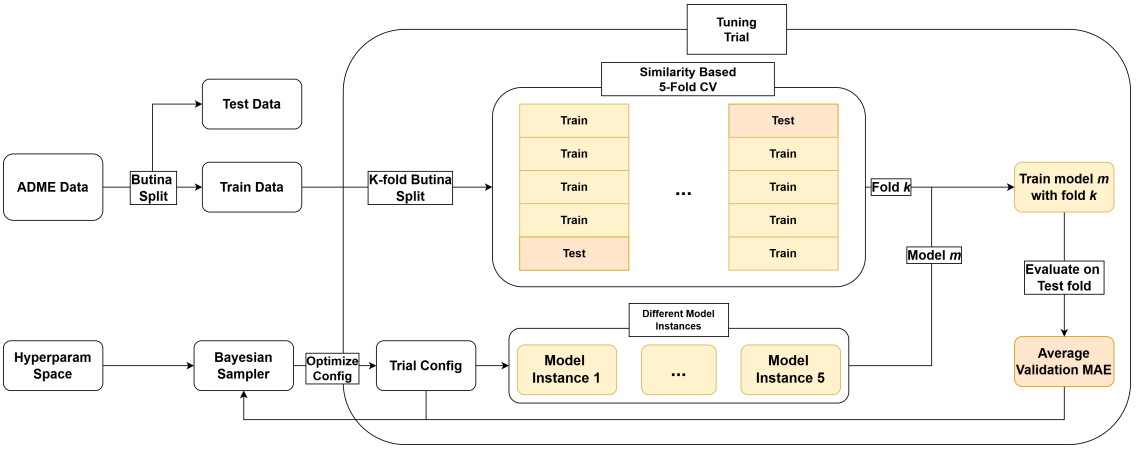

Figure 3.3: Schema of Bayesian hyperparameter tuning based on compound similarity in combination with a 5-fold cross-validation (Self-made Figure).

Neural Networks

Masked Multi-task Loss

A masked multi-task loss function is implemented to handle missing target values for multi-task regression problems. This is crucial since not every compound in the multi-task evaluation set contains a value for every endpoint (see Table 3.2). Thus, an indicator function $m_t^{(i)}$ (Equation 3.1) is defined, which yields 1 if a target value $y_{i,t}$ is not NaN and 0 if it is. The index i refers to the sample index, while the index t refers to the corresponding task (endpoints in the case of the ADME data). Additionally, the function N_t (Equation 3.2) uses $m_t^{(i)}$ to identify the number of non-NaN values per task in a batch B. Consequently, the function T'_{task} (Equation 3.3) is defined as the indicator function $1(N_t > 0)$, which counts the number of tasks containing at least one non-NaN value per batch.

Equation 3.4 shows the masked multi-task loss function for regression tasks. Each task $t \in T_{task}$ (four ADME endpoints) is viewed separately within the batch B to calculate the task-specific loss. More specifically, for each task-specific target value $y_{i,t}$, the Mean Squared Error (MSE) is calculated as $m_t^{(i)} \cdot (\hat{y}_{i,t} - y_{i,t})^2$, where the term $m_t^{(i)}$ ensures that the MSE is calculated only for target values that are not missing. The scaling term $\frac{1}{\max(N_t,1)}$ uses a max function to prevent a zero division error in case of a batch where a task contains only NaN values by dividing by 1. Otherwise, the scaling term uses the number of non-NaN values per task within a batch, denoted as N_t. This results in a MSE calculation where the mean is based only on non-NaN values. Lastly, another scaling term $\frac{1}{T'_{task}}$ and the indicator function $1(N_t > 0)$ ensure that the task-specific loss values are aggregated by the mean operation, involving only tasks with non-NaN values.

$$m_t^{(i)} = \begin{cases} 1 & \text{if } y_{i,t} \text{ is not NaN} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

$$N_t = \sum_{i=1}^{B} m_t^{(i)} \tag{3.2}$$

$$T'_{task} = \sum_{t=1}^{T_{task}} 1(N_t > 0) \tag{3.3}$$

$$\mathcal{L}_{\text{MSE}} = \frac{1}{T'_{task}} \sum_{t=1}^{T_{task}} \left[\frac{1}{\max(N_t, 1)} \sum_{i=1}^{B} m_t^{(i)} \cdot (\hat{y}_{i,t} - y_{i,t})^2 \right] \cdot 1(N_t > 0) \tag{3.4}$$

- B as bach size
- T_{task} as tasks
- $\hat{y}_{i,t}$ as prediction and $y_{i,t}$ as target for sample i, task t

Tuning Details

For each neural network model (GIN, EGNN, and LSTM), $t_{hp} = 100$ tuning trials are used. Inspired by the Google tuning playbook [116], the architecture hyperparameters are tuned first, and the optimizer parameters are tuned afterwards based on the previously found architecture. While tuning the architecture, the AdamW [117] optimizer is used with its standard parameters. The architecture tuning runs for $t_{hp}^{(a)} = \lfloor t \cdot s \rfloor = \lfloor 100 \cdot 0.66 \rfloor = 66$ trials and uses $n = t_{hp}^{(a)}/2 = 33$ starting trials, where $s \in [0, 1]$ is the fraction of trials used for architecture tuning. The architecture parameters involve predictor parameters, which describe parameters of a regular MLP: hidden dimension, dropout probability [118], and depth. Every encoder tunes the hidden dimensionality of the encoder, the encoder depth, and the output dimensionality. The GIN and the LSTM also tune the dropout probability of the encoder, which is left out for the EGNN since its original implementation does not feature dropout layers [31]. The LSTM also tunes the number of attention heads.

The optimizer tuning uses the previously best neural network architecture, identified during the architecture tuning, to tune the parameters of an AdamW optimizer. Therefore, $t_{hp}^{(o)} = \lfloor t \cdot (1 - s) \rfloor = \lfloor 100 \cdot 0.34 \rfloor = 34$ describes the number of trials for the optimizer tuning, with $n = t_{hp}^{(o)}/2 = 17$ starting trials. The tuned optimizer parameters are the coefficients for computing the running averages of the gradient and its square

(β_1, β_2), a numerical stability term ϵ, the initial learning rate η_{init}, and the weight decay coefficient λ_{wd}. Each network is trained for 1000 epochs with early stopping [119] based on the validation loss, parameterized by a patience value of 20 for each fold. Every network uses 5 warmup epochs [120] to gradually increase the learning rate to stabilize training. A cosine-annealing scheduler with warm restarts [121] is used to adapt the learning rate during training, with number of iterations $T_0 = 20$, increasing factor $T_M = 1$, and a minimal learning rate $\eta_{min} = 1e^{-6}$. Furthermore, Monte Carlo dropout [122] layers are used with 150 iterations for validating the test folds. Additionally, the LSTMs are trained with 7 SMILES-augmentations [34] per SMILES string. SMILES-augmentation refers to a method that creates different SMILES representations for the same compound by using varying starting atoms and paths along the molecular graph, resulting in different string representations. A fixed batch size of 128 is used for every network architecture.

XGB

The XGB architecture is tuned for $t_{hp} = 100$ trials on every single-task (RLM, HLM, Sol, MDR1-ER) ADME dataset. For that, $n = t_{hp}/2 = 50$ starting trials are used for the hyperparameter tuning of XGB. The XGB hyperparameter space consists of the fraction of column samples per tree, the algorithm's learning rate, tree depth, minimum child weight, number of trees, λ_{L2} (L2) and α_{L1} (L1) regularization term, and the fraction of a subsamples per tree. The hyperparameter spaces and their descriptive statistics are described in Table A.3-A.12 in Appendix A.2. The best hyperparameter set for each model can be found in Table A.13-A.19 in Appendix A.3.

3.2.2 VICReg for Computational Chemistry

In the original VICReg [41] paper, a siamese neural network [83] is trained on different augmented views X, X' of the same image $I \in \mathcal{I}$. This can be done in a self-supervised manner since labels are not needed to align the output representations of the encoders with the VICReg loss function (see Equation 2.26). The authors also state that VICReg can be applied to multiple encoders without weight sharing. Different views X, X' of the same image I can be seen as different representations through transformations $t_{aug}, t'_{aug} \in T_{aug}$ [82]. Applied to the domain of computational chemistry, different representations S' of the same SMILES string S can be derived by using different modalities through various SMILES featurizers $\phi^{\cdot}$. Thus, various representations of the same compound are created by utilizing the graph, conformer, and text modalities, with each modality being processed by its respective encoder (GIN, EGNN, and LSTM).

Figure 3.4 shows how VICReg is adapted to the domain of computational chemistry. A batch of compounds S serves as unlabeled training data, while different featurizers ϕ', ϕ'', ϕ''' transform the compounds S into their corresponding modality representations S', S'', S'''. Specifically, featurizer ϕ' tokenizes and embeds the compounds S based on the SMILES language, yielding a batch of d-dimensional embeddings S' to represent the text modality. Featurizer ϕ'' utilizes the graph modality of the compounds S by featurizing every atom and bond between atoms of the compound with predefined node and edge features, resulting in a batch of graph representations S''. Lastly, the featurizer ϕ''' uses the conformer modality by leveraging the 3D positions of compounds S to create the conformer representations S'''. The different views of compounds S propagate through their respective encoders to output batches of encoder representations Y', Y'', and Y'''. Each encoder representation is further processed by a distinct projector P', P'', and P''' to output batches of embeddings Z', Z'', and Z'''. Finally, the projections are used with the VICReg loss function to minimize the invariance between the embeddings Z', Z'', Z''' and to maintain the variance within a batch $Z^{\cdot}$. Furthermore, the loss function decorrelates the embedding dimensions d of every batch of embeddings $Z^{\cdot}$. Afterward, the projectors are discarded and each encoder can be saved for further downstream task evaluation.

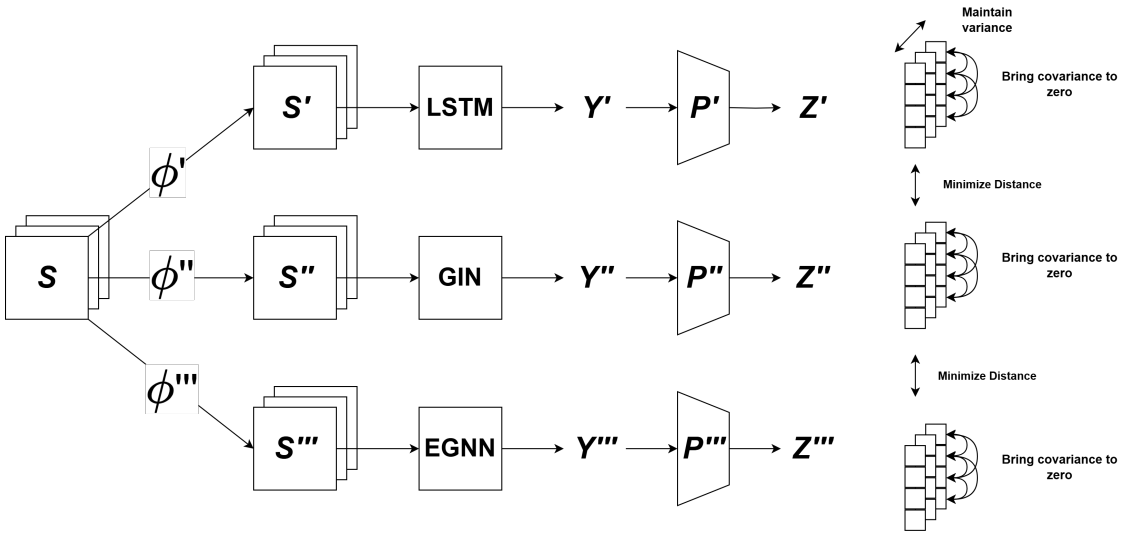

Figure 3.4: Transfering the VICReg architecture to computational chemistry where S is a batch of compounds, $\phi^{\cdot}$ is a modality specific featurizer and S', S'', S''' are batches of different compound views. Each encoder yields a representation $Y^{\cdot}$ which is expanded through a projector $P^{\cdot}$ to output an embedding $Z^{\cdot}$. The VICReg loss function operates on the embeddings Z (Self-made Figure).

The invariance loss function (see Equation 3.7) is slightly adapted to provide training stability when using more than two encoders. The invariance loss ℓ_s calculates the pairwise mean squared Euclidean distance (see Equation 2.25) between two embedding batches Z_i and Z_j. Consequently, the invariance loss is calculated as the sum of every unique pairwise combination of two embedding batches Z_i and Z_j from a list of embedding batches $Z_{List} = [Z_1, ..., Z_n]$. Thus, Z_{List} contains the embeddings from each modality-specific encoder, resulting in a list of 3 batches (graph, conformer, and text modality). For stability purposes, the invariance loss is scaled with a factor of $\frac{\lambda}{C_{comb}}$ to ensure training stability where C_{comb} is the result of the binomial coefficient $\binom{|Z_{List}|}{2}$ resulting in $\binom{3}{2} = 3$ pairwise invariances. A function $\mathcal{P}(Z_{List})$ (Equation 3.5) is defined for a list of embedding batches Z to yield every unique pair of two distinct embedding batches $Z_i, Z_j \in Z_{List} \mid i \neq j$. Accordingly, the variable C_{comb} (see Equation 3.6) corresponds to the cardinality of the resulting set $\mathcal{P}(Z_{List})$ which is equivalent to the binomial coefficient $\binom{|Z_{List}|}{2}$.

$$\mathcal{P}(Z_{List}) = \{\{Z_i, Z_j\} \mid 0 \leq i < j < |Z_{List}|\} \tag{3.5}$$

$$C_{comb} = \binom{|Z_{List}|}{2} = |\mathcal{P}(Z_{List})| \tag{3.6}$$

Equation 3.7 shows the VICReg loss function $\mathcal{L}_{VICReg}$ adapted to n encoders. The encoder outputs result in a list of embedding batches Z_{List} containing batches $Z_1, ..., Z_n$. The regularized variance loss $\mu \cdot \ell_v(Z_i)$ and the regularized covariance loss $\nu \cdot \ell_c(Z_i)$ are calculated independently for every batch $Z_i \in Z_{List}$. The invariance loss is computed as the mean of the pairwise invariance losses over all unique unordered pairs of elements in Z_{List}, and is scaled by the regularization term λ.

$$\mathcal{L}_{VICReg}(Z_{List}) = \frac{\lambda}{C_{comb}} \sum_{i,j}^{C_{comb}} \ell_s(Z_i, Z_j) + \mu \sum_i \ell_v(Z_i) + \nu \sum_i \ell_c(Z_i) \tag{3.7}$$

- Z_{List} as a list of embedding batches $Z_1, ..., Z_n$ yielded by n different encoders
- C_{comb} as amount of pairwise embedding batch combinations (Equation 3.6)
- $\mathcal{P}(Z)$ (Equation 3.5) as function to provide embedding pairs
- λ, μ, ν as VICReg regularization terms
- ℓ_s as invariance loss (Equation 2.25), ℓ_v as variance loss (Equation 2.22) and ℓ_c covariance loss (Equation 2.24)

3.3 Performance Evaluation

3.3.1 Evaluation Protocols

Linear Downstream Evaluation

A linear evaluation protocol is implemented similarly to the works of self-supervised representation learning models from the CV domain [41, 43, 42, 46]. For this, an evaluation schedule is used in which a linear multi-task regressor is trained on the frozen representations of a pre-trained encoder and evaluated on the test split. Each regressor is trained for 200 epochs with the masked multi-task loss (Equation 3.4). A linear learning rate warmup with 10 epochs, a batch size of 256, an AdamW optimizer with default parameters, and a cosine-annealing scheduler with maximum number of iterations $T_{max} = 190$, minimal learning rate $\eta_{min} = 1e^{-6}$, initial learning rate (after warumup) $\eta_{init} = 1e^{-3}$ is used for the training process. For stability purposes, Batch Normalization (BN) layers [123] are added to normalize the frozen representations. The evaluation process is repeated with 30 different random seeds resulting in a distribution of test metrics for every encoder [124, 125, 41, 97].

Figure 3.5 visualizes the linear evaluation protocol. A batch of compounds S is featurized with a modality-specific featurizer $\phi^{\cdot}$, yielding a different representation of compounds $S^{\cdot}$. The featurized compounds $S^{\cdot}$ are propagated through a frozen encoder, which outputs the corresponding representations. These representations are used to train a BN layer paired with a linear layer that outputs the downstream prediction. After training, the linear regressor is evaluated on the unseen test split.

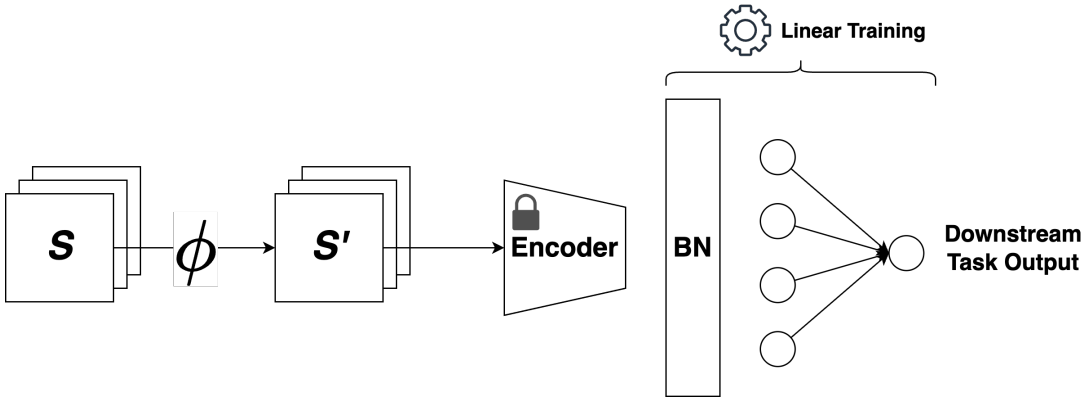

Figure 3.5: Schema of the linear evaluation protocol. A linear layer paired with a BN layer is trained on the frozen encoder representation to test an encoder's representation capability (Self-made Figure).

Transfer Learning Downstream Evaluation

A more flexible evaluation protocol is implemented using transfer learning [67, p.602-607] to evaluate the effect of the different pre-training strategies. For this, only the encoders' BN layers are kept frozen, as this approach helps maintain the stability of feature representations during training [126]. Instead of a linear layer, an MLP with 4 layers, 512 hidden dimensions, BN layers, dropout layers with dropout probability $p_{drop} = 0.2$, and the Mish [127] activation function is used for the downstream task. The AdamW optimizer is used with default parameters, except the initial learning rate after the warmup epochs, which is set to $1e^{-3}$ for the predictor and $1e^{-4}$ for the encoder to ensure that the learned weights are not changed too drastically. Again, the downstream task is described by the ADME dataset, which is why the masked multi-task loss function is used. The transfer learning evaluation protocol uses the same training schedule configuration as the linear evaluation protocol. Furthermore, each transfer learning evaluation run is also performed 30 times with different random initializations to capture uncertainties in the test metric distributions.

Figure 3.6 visualizes the transfer learning protocol. A batch of compounds S is featurized by a featurizer ϕ to provide a batch of suitable compound modalities S' for the corresponding encoder. The encoder and the predictor are trained while the encoder BN layers remain frozen.

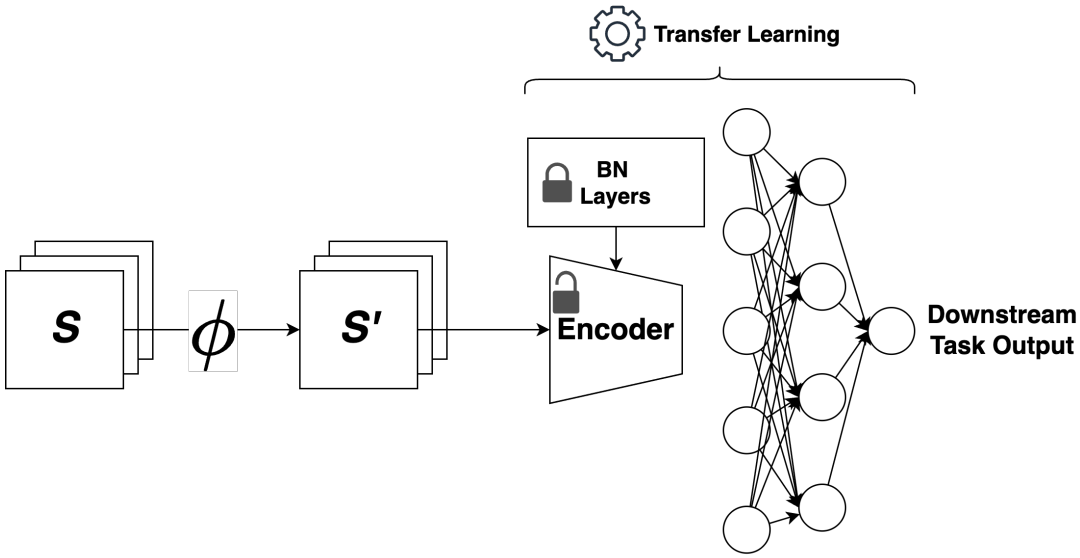

Figure 3.6: Just the parameters of the encoders BN are kept frozen for the transfer learning evaluation protocol. An MLP is trained on the downstream task instead of just a linear layer (Self-made Figure).

3.3.2 Assess Performance Differences

The resulting test metric distributions from the evaluation runs are used for statistical tests with multiple comparisons. Therefore, each model evaluation run results in 30 R2 and MAE samples, which capture the performance and uncertainty of the corresponding model. These metrics are compared to assess whether significant differences exist between different experiments by using a repeated measures ANOVA. If the p-value is $\ll 0.05$, a Tukey test serves as a post-hoc procedure to show which experiment differs significantly from the rest. The test metric distributions are always compared within the same endpoint.

3.4 Experiments

Several experiments are conducted to investigate how different training strategies influence the encoders' performance on the downstream task with the ADME dataset. The experiments can be grouped by similar types of training strategies, while sub-experiments differ in chosen aspects.

3.4.1 No Pre-training

The "No Pre-training" experiment involves the training and evaluation without using the largemix pre-training dataset. Therefore, only the training split of the ADME is used for model training.

XGB and Neural Networks

Every model is trained on the training split and evaluated on the test split of the ADME data, while a 10% fraction of the training split serves as a validation split for techniques such as early stopping. The validation split is based on structural compound similarity using Butina clusters as the splitting criterion. The XGB models are trained with the tabular compound representation and serve as the baseline models for each respective endpoint. Hence, each encoder paired with an MLP is trained and evaluated on the same ADME data based on the previously determined hyperparameter configurations. The neural networks are trained with the masked multi-task loss, while the XGB uses the MSE for single-task problems.

The XGB, GIN, EGNN, and LSTM models are trained 30 times with different random weight initializations to generate test metric distributions, which are further utilized for statistical testing. The neural networks use the previously determined architecture and optimizer hyperparameters instead of using the default AdamW parameters or a predefined MLP configuration like in the transfer learning evaluation.

VICReg

The VICReg model is trained with a batch size of 512 for 300 epochs. The encoders are configured with the best architecture hyperparameters found during hyperparameter optimization, except for the encoder output size, which is set to 512. Each encoder uses a distinct projector with 3 layers, where each layer has 512 hidden units and the output layer has 1024 hidden units. The projectors have BN layers and a Rectified Linear Unit (ReLU) activation function. An AdamW optimizer is used with a weight decay coefficient of $1e^{-3}$ and an initial learning rate after the warmup epochs of $\eta_{init} = 1e^{-2}$. A cosine annealing scheduler is used to adapt the learning rate with the parameters $T_{max} = 295, \eta_{min} = 1e^{-5}$, while 5 warmup epochs are used to provide a stable start to the training process. The VICReg regularization parameters, which resulted from the ablation study of the original work, where $\lambda = \mu = 25$ and $\nu = 1$, are used since they performed the best [41]. Each VICReg encoder is saved and evaluated with both evaluation protocols, yielding a test metric distribution for each endpoint. This experiment should provide initial insights into VICReg's test performance, making comparisons to other experimental settings possible.

3.4.2 Model-agnostic Pre-training

A supervised, model-agnostic pre-training strategy is implemented, which uses the labeled largemix dataset. The dataset contains 3.8M compounds labeled with 25 quantum-based regression tasks (PCQM4M_G25_N4) and 1.6M compounds labeled with 1328 binary bio-assay classification tasks (PCBA_1328). The dataset is sparse and does not contain every task label for every compound, which is why the masked multi-task loss from Equation 3.4 is used again. The same issue is present for the binary classification tasks. Consequently, a masked multi-task binary classification loss function is defined, analogous to the masked regression loss function.

Masked Multi-task Classification Loss

The masked classification loss is created by using the Binary Cross-Entropy (BCE) [77, p.206], which is shown in Equation 3.8. The BCE quantifies the dissimilarity between the probability distribution of the ground truth label y and the predicted probabilities derived from logits $\hat{y}$. The term $\sigma(\hat{y})$ denotes the sigmoid function applied to the logits to obtain probabilities. For each sample, the BCE combines two terms to compute the loss value.

The first term $-y \log(\sigma(\hat{y}))$ calculates the loss for predicting the positive class ($y = 1$). The term yields a small loss if the ground truth is similar to the transformed logits $\sigma(\hat{y})$, since the *log* transformation outputs smaller numbers for values that converge to 1. The sign ensures that the loss is a positive value because the *log* transformation of numbers between 0 and 1 is always negative. In contrast, the loss magnitude is higher when the transformed logit value converges to 0, since the logarithm of values approaching zero tends to negative infinity, resulting in a larger penalty.

The second term, $-(1-y) \log(1-\sigma(\hat{y}))$, describes the loss calculation for predicting 0 labeled targets ($y = 0$). This term yields a small value if the transformed logit refers to a lower probability towards the positive class, since $-\log(1-\sigma(\hat{y}))$ becomes smaller due to the *log* transformation of a number converging to 1. In the opposite case, the loss increases.

Equation 3.9 displays the masked multi-task classification loss function. For each task $t \in T_{task}$, the BCE is calculated for each sample i within a batch B. If sample i has no label for task t, the function $m_t^{(i)}$ (Equation 3.1) yields 0, while in any other case it is weighted wit 1. Accordingly, the sum $\sum_{i=1}^{B} m_t^{(i)} \cdot \ell_{\text{BCE}}(\hat{y}_{i,t}, y_{i,t})$ sums up the BCE loss for every case where a valid label $y_{i,t}$ is provided for observation i and task t. A scaling factor $\frac{1}{\max(N_t,1)}$ provides the scaling factor for an effective mean aggregation operation, depending on the amount of missing values within a batch of a multi-task dataset, where N_t (Equation 3.2) refers to the number of observed tasks t in batch B. If the batch contains only NaN values, a value of 1 is used in the denominator to prevent a zero division error. The function T'_{task} (see Equation 3.3) serves as a sum of indicator functions N_t (see Equation 3.2) to count the number of tasks that have at least one non-NaN value per batch, to aggregate the task-wise losses as the mean value through a scaling factor $\frac{1}{T'_{task}}$.

$$\ell_{\text{BCE}}(\hat{y}, y) = -y \log(\sigma(\hat{y})) - (1 - y) \log(1 - \sigma(\hat{y})) \tag{3.8}$$

$$\mathcal{L}_{\text{BCE}} = \frac{1}{T'_{task}} \sum_{t=1}^{T_{task}} \left[\frac{1}{\max(N_t, 1)} \sum_{i=1}^{B} m_t^{(i)} \cdot \ell_{\text{BCE}}(\hat{y}_{i,t}, y_{i,t}) \right] \cdot 1(N_t > 0) \tag{3.9}$$

- $\sigma(z) = \frac{1}{1+e^{-z}}$ [29, p.195] as sigmoid function
- B as batch size
- T_{task} as tasks
- $\hat{y}_{i,t}$ as prediction logits and target $y_{i,t}$ for sample i, task t
- ℓ_{BCE} as BCE loss function

Hybrid Supervised model-agnostic Pre-training Architecture

A multiheaded MLP architecture is implemented, which uses the masked multi-task regression (Equation 3.4) and classification (Equation 3.9) losses to train any encoder architecture on the labeled largemix dataset in a supervised manner.

Figure 3.7 visualizes the pre-training process when using the hybrid supervised, model-agnostic pre-training strategy. A batch of compounds S is featurized by a featurizer $\phi^{\cdot}$ into a batch of featurized compounds $S^{\cdot}$ suitable for a modality-specific encoder. The supervised pre-training model consists of the encoder and a multiheaded MLP, which is composed of a projector P and two loss-specific multi-task heads H_1 and H_2. The encoder yields a batch of representations Y for every compound in $S^{\cdot}$. These representations are expanded by the projector into a batch of embeddings. The embeddings are used to train a head H_1 for regression tasks and a head H_2 for classification tasks. The heads H_1 and H_2 are smaller, task-specific, multi-task MLPs.

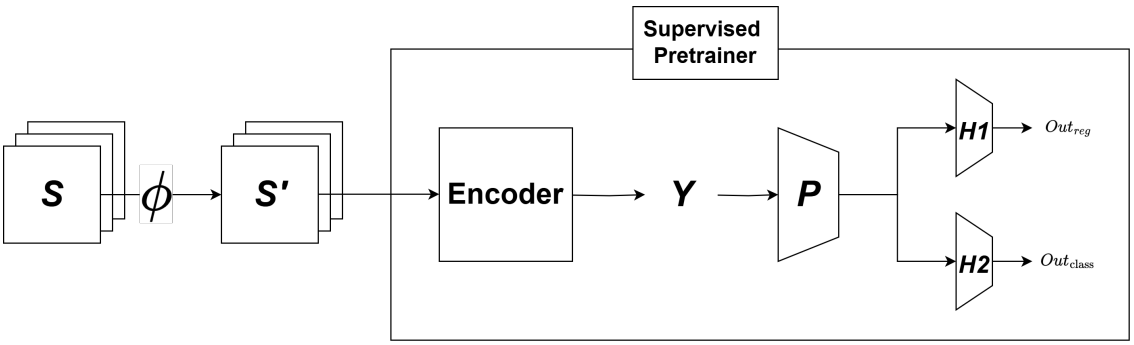

Figure 3.7: Architecture of the model-agnostic pre-training network. An encoder is trained with the largemix data in a supervised manner where a projector P has a head for regression and for classification tasks. The loss is calculated independently and aggregated to update the weights of the model (Self-made Figure).

Agnostic Pre-training Experiments

Encoder-wise

A GIN, EGNN, and LSTM encoder are pre-trained with the hybrid supervised, model-agnostic pre-training architecture on the largemix dataset. Each agnostic pre-training run is conducted over 1000 epochs using early stopping based on the validation loss configured with a patience of 30. The validation split is generated using a random split with a fraction of 10%. The batch size is set to 4096 and the encoder output size to 1024, while the remaining encoder architecture parameters are set to those found by hyperparameter tuning. Each training run uses 5 warmup epochs, an AdamW optimizer with default parameters, and a cosine annealing scheduler with warm restarts, where $T_0 = 20$, and $\eta_{min} = 1e^{-5}$. SMILES strings with a length longer than the 0.99 quantile regarding sequence length are removed, since long sequences can cause GPU bottlenecks for large batch sizes. As a consequence, around 5k compounds are dropped from the pre-training dataset. Additionally, the LSTM uses two A100 GPUs instead of one, with a data parallel strategy that allows for efficient training by splitting the input data across the GPUs. The LSTM is also trained with 5 SMILES augmentations per SMILES string. After the model-agnostic pre-training terminates, the encoder is saved while the multiheaded MLP is discarded. Each saved encoder is evaluated using both the linear and the transfer learning evaluation protocols. The LSTM evaluation uses the tokenizer that was built on the training split of the pre-training set.

VICReg

The VICReg model is trained with the same constraints as the LSTM on the largemix pre-training dataset. Thus, SMILES strings longer than the 0.99 length quantile are removed, two A100 GPUs are used instead of one, and the training tokenizer is used for the LSTM evaluation. The VICReg model is trained for 30 epochs with a batch size of 4096. The encoder dimension is set to 2048 and the projector dimension to 4096. The remaining parameters are set analogously to those of the No Pre-training VICReg experiment. In contrast to the VICReg parameters of the experiment without the largemix pre-training, gradient clipping [67, p.322] is used and the initial learning rate after the warmup epochs η_{init} is set to $1e^{-3}$ preventing unstable gradients.

VICReg on Pre-trained Encoders

A VICReg model is trained using the pre-trained encoders from the model-agnostic pre-training to investigate whether the use of already pre-trained encoders makes a difference for pre-training with VICReg. For this purpose, the same setting from the VICReg training on largemix is used, with the only difference being that the encoders are already pre-trained instead of randomly initialized.

3.4.3 Model-specific Pre-training

The third experiment focuses on modality-specific self-supervised pre-training strategies. Therefore, the largemix dataset is still used as the pre-training dataset, but the labels are not part of the training process. An MLM-based approach is used for the LSTM, while both GNN architectures are trained with a node masking strategy.

Specific Pre-training Architectures

LSTM

An MLM is implemented to pre-train the LSTM encoder for the specific pre-training experiments. In contrast to BERT [3], only the token masking strategy is deployed and no sentence prediction task is used. Therefore, a batch of token IDs is randomly masked with a masking probability of 15%, and the LSTM tries to predict the masked tokens with a small classifier. The cross-entropy [77, p.206] loss is used as the loss function.

GIN and EGNN

The GIN and EGNN encoders are trained with the same specific self-supervised pre-training strategy, since both modalities operate on graph-structured data. A masked node attribute pre-training strategy [80] is implemented with a mask probability $p =$

15% to decide which nodes are masked out in the current batch, using the zero vector as an attribute mask for each chosen node. Thus, the node feature matrix contains the targets for the self-supervised training in combination with a small predictor for each node feature. Different loss functions are used since the node attributes involve different feature types. Additionally, only the predictions for the masked-out nodes are taken into account for the loss calculation to prevent overfitting on the given node attributes. The loss value for the multilabel classification tasks is calculated with cross-entropy, the loss for the binary feature is calculated using BCE, and the loss for the regression task is calculated with the MSE. The different loss values are aggregated as a mean value.

Figure 3.8 visualizes the GNN-based pre-training using node attribute masking as a self-supervised method. A graph consisting of 3 nodes serves as an input example. The colors yellow, orange, and red indicate the types of different features: yellow-colored features are categorical features with more than 2 categories, orange-colored features refer to binary classification features, and red-colored features are continuous regression features.

The first step (**"1. Featurizing"**) of the pre-training schedule is the featurizing step. The result of featurizing a graph is an adjacency matrix representing the connectivity between atoms and a node feature matrix. The node feature matrix contains the features of every node, where the rows correspond to the nodes and the columns to the features. Different features are concatenated, resulting in a long feature vector with either bits or continuous numbers to represent a featurized atom.

The second step (**"2. Feature Masking"**) shows how node features are masked. A node mask is calculated based on the masking probability p to choose which nodes are masked out. In this example, node 3 is masked out with the corresponding mask $[0, 0, 1]$. The masked-out node is highlighted with a red border in the node feature matrix, consisting of only 0 values for each node attribute.

Finally, step three (**"3. Self-Supervised Training"**) visualizes how the GNNs are trained in a self-supervised manner. The node feature matrix containing the masked-out nodes propagates through the GNN-based encoder, yielding logits for each node. The true labels are inferred from the unmasked node feature matrix. The encoder logits of each feature are used to train a linear head, where each feature has its own classification or regression head. The output is used with the ground truth realization to calculate a loss based on the feature type. The losses are aggregated as a mean value.

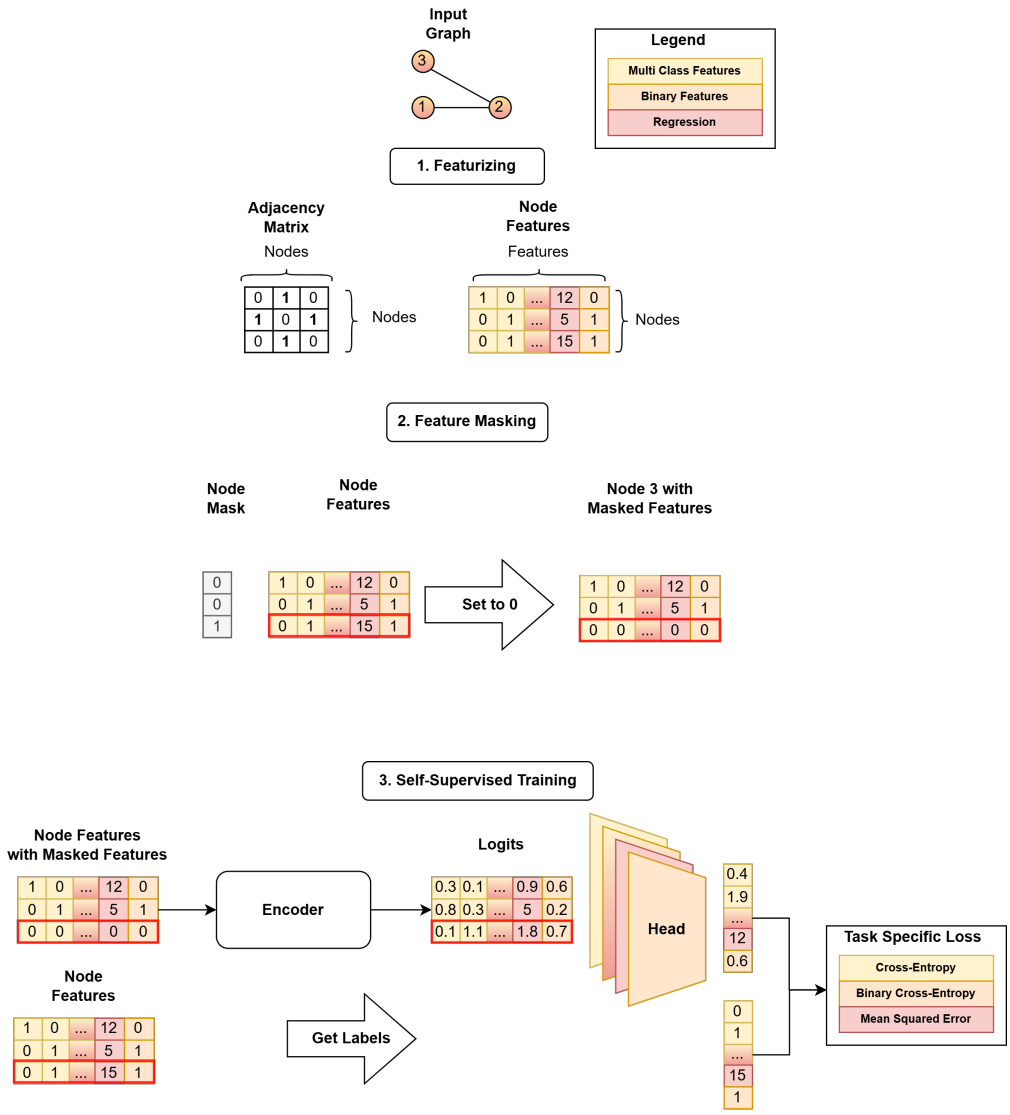

Figure 3.8: Example of a GNN specific pre-training run with a molecule consisting of three nodes. The properties of the third node are masked out using a zero vector, enabling a self-supervised training for the GNN by predicting the masked features (Self-made Figure).

Specific Pre-training Experiments

Encoder-wise

The three modality-specific encoders are pre-trained on the largemix dataset with the respective modality-specific pre-training strategy. Each pre-training uses an AdamW optimizer with default parameters, a batch size of 4096, and a cosine-annealing scheduler with $T_{max} =$ epochs and $\eta_{min} = 1^{-6}$. Due to a fast training loss convergence the GNNs are pre-trained for 7 epochs, with 2 warmup epochs and a masking probability $p = 15\%$, while the LSTM is pre-trained for 30 epochs with 5 warmup epochs, a masking probability $p = 15\%$, and 5 SMILES augmentations per SMILES string. Again, the LSTM

runs on two A100 GPUs, and compounds with a compound length ≥ 0.99 compound length quantile are removed from the pre-training data.

VICReg on Pre-trained Encoders

To investigate whether it makes a difference if a VICReg model is pre-trained with agnostic pre-trained encoders, modality-specific self-supervised pre-trained encoders, or no pre-trained encoders, a VICReg model is trained on the largemix dataset with the modality-specialized pre-trained encoders. The hyperparameters are chosen analogously to the VICReg training parameters from the largemix data.

Chapter 4

Results and Discussion

4.1 Effect of Pre-training

In the following, the effect of model-agnostic supervised and model-specific self-supervised pre-training is investigated by comparing the results of the respective pre-training strategies with those of the baseline model (XGB). The results are also compared with those of neural networks that used optimized hyperparameters for the optimizer and predictor without pre-training, to determine whether different pre-training strategies provide any benefit for ADME prediction as downstream tasks. Each section follows the same schema, where the linear and transfer learning evaluation results are described first and then discussed. Moreover, after each results subsection, a brief summary of the supplementary material is provided to support the observed results with additional information.

4.1.1 Linear Evaluation

Figure 4.1 shows the different test MAE distributions depending on encoder type, experiment, random state, and downstream task for the linear evaluation protocol. The y-axis shows the test MAE, while each column represents a different model type. The x-axis refers to the different experiments, which describe the two different pre-training strategies in this case, where "Agnostic Pretrain" refers to the model-agnostic supervised pre-training and "Special Pretrain" refers to the modality-specialized self-supervised pre-training strategies depending on the encoder modality. The color corresponds to the four different ADME downstream tasks: blue-colored values refer to the RLM test scores, orange-colored values refer to the HLM test scores, green-colored values refer to the Sol test scores, and red-colored values refer to the MDR1-ER test scores. The dotted lines provide information regarding the single-task XGB models as baseline performance. Each endpoint contains 30 different value points per experiment, referring

to 30 re-trained model instances that were initialized with different random seeds to capture the uncertainty of each experiment.

The results show that the agnostic pre-training strategy yields superior results compared to the modality-specific pre-training for GNN-based models (GIN and EGNN), as the results of the agnostic pre-training have lower test MAE values for each endpoint. The LSTM results differ from this observation, as the modality-specific pre-training yields better results for the Sol endpoint. However, the agnostic pre-training is still superior for the remaining three endpoints. Additionally, the GNN models even surpass the MDR1-ER scores of the XGB with the linear evaluation protocol. The modality-specific pre-trained LSTM results are on par with the XGB's Sol results, while the XGB outperforms every other linear evaluated model for every other endpoint.

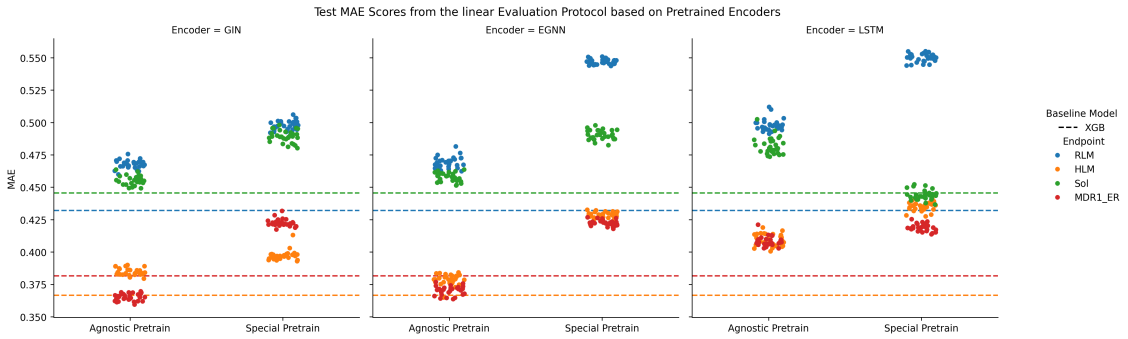

Figure 4.1: Test MAE results regarding the linear evaluation protocol for the ADME downstream tasks. Generally, the agnostic pre-training shows superior results which can even surpass the XGB results for MDR1-ER.

Supplement Figure B.1 shows the test R2 scores for the same evaluation procedure, capturing the same observation since the agnostic pre-training achieves superior results compared to the modality-specific pre-training results. The agnostic pre-trained GNN models even surpass the test R2 scores of the XGB for the endpoints Sol and MDR1-ER based on the linear evaluation.

The model-agnostic supervised pre-trained GIN model is significantly better than the self-supervised pre-trained GIN for both test metrics, except for the R2 results on endpoint HLM (see Supplement Figures B.3 and B.4). The EGNN results from the agnostic pre-training are significantly superior to those from the modality-specific pre-training when using the linear evaluation protocol (see Supplement Figures B.7 and B.8). The LSTM results show a similar trend except for endpoint Sol, where the MLM-based pre-training shows superior performance on both test metrics for that particular endpoint (see Supplement Figures B.11 and B.12).

4.1.2 Transfer Learning

Figure 4.2 follows the same construction schema as Figure 4.1. Therefore, the figure visualizes the test MAE performance of different encoders coming from the transfer learning evaluation protocol. Additionally, the results from the models without pre-training which contain optimized predictor and optimizer hyperparameters are also included and referred to as "No Pretrain." Again, the GNN-based models (GIN and EGNN) show similar results compared to the linear evaluation, as the results of the agnostic supervised pre-training strategy exceed those of the self-supervised pre-trained models and the models without pre-training. This is particularly visible for the endpoint MDR1-ER when comparing the agnostic pre-trained model results to the model results without pre-training, and for the endpoints MDR1-ER, RLM, and HLM regarding the modality-specific pre-trained GNN model results.

Thus, the GNN models without pre-training show similar results to the agnostic pre-trained GNNs, yielding better results than the modality-specific pre-trained GNNs. Nevertheless, the supervised pre-trained GNN and the GNN models without pre-training outperform the baseline XGB on every endpoint. The supervised pre-trained GNN models also show superior performance compared to the GNN models without pre-training. Both GNN models with the self-supervised, modality-specific pre-training perform at least as well as the baseline XGB, depending on the random seed since the RLM performance seems worse than the XGB performance for most of the random states.

The LSTM results differ from the GNN results because the results from the modality-specific pre-trained model surpass those from the LSTM without pre-training. Analogous to the GNNs, the supervised agnostic pre-training yields better results than the model without pre-training. However, the LSTM results based on the modality-specific pre-training outperform the supervised pre-training results for the endpoints HLM, Sol, and RLM. The LSTM with supervised pre-training yields the best test MAE scores for the endpoint MDR1-ER. Lastly, both LSTM pre-training strategies outperform the baseline XGB results on all four endpoints. The LSTM without pre-training shows inferior test MAE scores for all four endpoints.

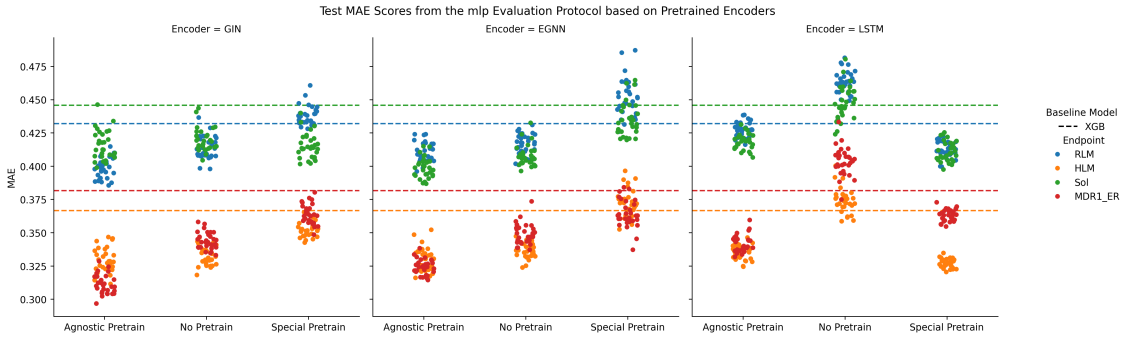

Figure 4.2: Summary of the transfer learning evaluation protocol for different pre-training strategies regarding the test MAE scores. The agnostic pre-training seems superior for graph based models, while the MLM based approach performs the best for the LSTM model.

Supplement Figure B.2 displays the transfer learning results regarding the test R2 score. The GNN-based models show a similar trend compared to the MAE results, as the agnostic pre-training yields superior results for all endpoints. The modality-specific pre-trained GNN results are inferior to the models using the optimized configuration without pre-training and the baseline XGB. The GNN models without pre-training are superior to the baseline XGB but inferior to the model-agnostic pre-trained GNNs. The LSTM without pre-training shows inferior R2 results compared to the baseline model for every endpoint. However, both LSTM pre-training strategies lead to test R2 distributions that are at least as good as the XGB results for every endpoint.

The test MAE results for the agnostic pre-trained GIN model from the transfer learning evaluation protocol are significantly superior for endpoints RLM, HLM, and MDR1-ER compared to the other experiments, while the GIN model without pre-training achieves similar results for HLM (see Supplement Figure B.5). The Sol results of all three GIN experiments show no significant differences, resulting in similar test MAE scores that are significantly better than the XGB's MAE scores. For the test R2 scores, the agnostic pre-trained GIN model significantly outperforms every other model for every endpoint (see Supplement Figure B.6), with the exception that the GIN model without pre-training shows similarly high R2 scores for endpoint HLM.

The agnostic pre-trained EGNN model yields significantly better test MAE and R2 scores for every endpoint compared to the other EGNN models in this experiment, while the EGNN model without pre-training yields similar results for RLM and HLM (see Supplement Figures B.9 and B.10). The transfer learning results of the LSTM encoder show that the specialized pre-training strategy provides significantly better MAE scores compared to the other LSTM models for endpoints RLM, HLM, and Sol (see Supplement Figure B.13). The agnostic pre-trained LSTM achieves the best MAE scores for MDR1-ER.

The test R2 results are endpoint-dependent: the XGB baseline and the specialized pre-trained LSTM achieve the best results for endpoint Sol, the agnostic pre-trained LSTM achieves the best results for MDR1-ER, the self-supervised pre-trained LSTM achieves the best results for endpoint RLM, and both the agnostic pre-trained LSTM and the specialized pre-trained LSTM achieve the best results for endpoint HLM (see Supplement Figures B.14).

4.1.3 Impact of Pre-training Strategies

The results of the linear evaluation protocol provide a first impression of the representation capability of the different encoders, since only the parameters of a BN and a linear layer are trained, resulting in only a few adjustable parameters for the ADME downstream tasks. The transfer learning evaluation leverages a slower encoder update to retain the information gained from the pre-training step through the use of a lower encoder-specific learning rate. Generally, the pre-trained models show superior and significantly better results for every modality-specific encoder, which proves that the pre-training of different neural network architectures can have beneficial advantages regarding downstream task performance [128, 129, 3], making the domain of computational chemistry no exception [130, 131, 80].

The supervised model-agnostic pre-training strategy yields the best results for the GNN-based models regarding the ADME downstream task performance scores (see Supplement Figures B.3-B.10). Hence, the agnostic pre-trained GNN models provide better representation abilities compared to the baseline XGB and the other pre-training strategies. This observation aligns with findings from GNN-specific pre-training experiments, which show that supervised pre-training can be overall superior to self-supervised pre-training when using node attribute masking as a self-supervised pre-training strategy [80]. Furthermore, the GNN results from the node attribute masking pre-training strategy are inferior to those of the GNNs without pre-training, mostly because the GNNs without pre-training have optimized predictor and optimizer hyperparameters tailored to the ADME data space. Therefore, those models might contain hyperparameters already adjusted to the properties of this particular ADME dataset, while the predictor and optimizer parameters of the pre-trained models are fixed to default parameters, making the superiority of pre-trained models in this comparison even more meaningful.

However, the LSTM benefits from both pre-training strategies, as both pre-trained LSTM models outperform the LSTM without pre-training and the XGB for most downstream tasks. One reason is certainly the poor performance of the LSTM without pre-training, which yields the worst results for every endpoint (see Supplement Figures B.13 and B.14). This result may originate from the small size of the training split

from the ADME dataset (see Supplement Table A.1), since language models require large data sources to work properly [132, 133]. Therefore, pre-training the LSTMs with SMILES augmentations [34] is a crucial step, as the augmentation of SMILES strings expands the size of the pre-training dataset even further. This is particularly important for self-supervised pre-training using MLM, as it enables the LSTM to leverage vast amounts of diverse unlabeled data, allowing it to learn different contextual relationships and patterns of the chemical language expressed through the SMILES notation system. Consequently, both supervised and self-supervised pre-training are valid strategies, yielding competitive test metrics on the downstream tasks and confirming that language models combined with self-attention are highly capable of learning complex patterns from large data sources [3, 134].

The superiority of the supervised agnostic pre-training strategy could also originate from a closer relation between the supervised tasks and the downstream task [135] since the largemix dataset contains bio-assay expression data. Thus, the captured information about biochemical activities may give the agnostic pre-training strategy an advantage towards the ADME downstream task [136] making it suitable for supervised model-agnostic pre-training experiments for bio-chemical related tasks.

An additional factor for the "poor" performance of the self-supervised GNN pre-training approach might lay in the diversity of different node attributes using chemical featurizing approaches because these attributes may not capture the full complexity of the chemical structures represented in the SMILES strings. This limitation could be coming from the overrepresentation of carbon atoms in organic chemistry [106, p.1] datasets, which dominate most molecular structures and may make the atom attribute prediction easier compared to other self-supervised or supervised pre-training tasks.

Exploring alternative more sophisticated self-supervised GNN pre-training strategies such as contrastive learning [137, 138], Graph level similarity prediction [139, 140], a combination of node and edge masking [81], or contextualized node predictions [80] might better account for the nuanced roles of carbon and other heteroatoms.

In summary, pre-training strategies can significantly enhance model performance in computational chemistry for ADME prediction as downstream tasks. A solely self-supervised pre-training approach seems to be inferior compared to supervised pre-training strategies for GNNs confirming findings from other experiments [80], while language models benefit from self-supervised learning via MLMs. The supervised model-agnostic pre-training leads to outperforming results for every modality specific encoder.

4.2 Different VICReg Strategies

This section focuses on experiments involving different pre-training strategies with VICReg to investigate which settings result in the most beneficial outcomes for the ADME downstream tasks.

4.2.1 ADME vs Largemix

One experiment is conducted where a VICReg model is trained with the largemix data and another where a VICReg model is trained with the training split of the evaluation data, to investigate how different pre-training dataset sizes affect VICReg's downstream performance. Thus, the experiment with the VICReg model trained on the ADME data is labeled "ADME," while the VICReg model trained with the largemix dataset is labeled "Largemix." The resulting encoders from both experiments are evaluated using the linear and transfer learning evaluation protocols on the ADME data.

Linear Evaluation

Figure 4.3 summarizes the results of the linear evaluation protocol for the test MAE metric, using the same visualization as in the pre-training comparison. The MAE is generally lower for the encoders where the VICReg model was trained with the largemix dataset. This is particularly evident for the endpoints MDR1-ER, RLM, and HLM, as the performance gap between the experiments is rather large. However, the performance benefit for Sol is also present, though smaller. This is especially notable for the LSTM encoder, since the performance gaps are larger between the different LSTM results. The baseline XGB shows superior test MAE scores on every endpoint compared to every encoder, indicating that the yielded representations are not sufficient to train a linear layer.

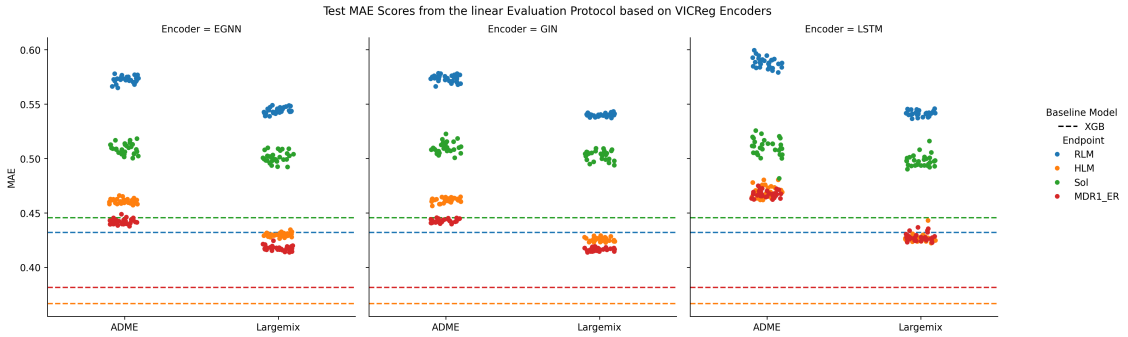

Figure 4.3: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the linear evaluation protocol. ADME refers to the pre-training on the ADME training split, while largemix refers to the pre-training on the largemix dataset. The y-scale refers to the test MAE coming from an evaluation of the ADME dataset.

Supplement Figure B.15 shows the linear VICReg evaluation for the test R2 metric. The results are analogous to the MAE results, where the VICReg trained on the largemix data consistently outperforms the VICReg results trained on the ADME data for the linear evaluation protocol. The results from the linear-evaluated VICReg encoders trained with the largemix dataset are significantly superior to the results from the ADME pre-training for every encoder and every test metric (see Supplement Figures B.17 and B.18 for GIN, Figures B.21 and B.22 for EGNN, and Figures B.25 and B.26 for the LSTM).

Transfer Learning

Figure 4.4 shows the results of the transfer learning evaluation protocol for different dataset sizes. The VICReg model trained on the largemix pre-training dataset achieves overall better results, similar to the linear evaluation protocol. Each encoder achieves better test MAE scores on each endpoint, with the LSTM showing the largest improvements. Comparing the results to the baseline XGB scores, it is notable that the downstream task performance of VICReg trained on the ADME dataset is inferior for every endpoint. On the other hand, the VICReg performance resulting from pre-training on the largemix dataset shows that every encoder achieves at least similar performance compared to the XGB, depending on the random state. The EGNN shows superior MDR1-ER results and similar Sol results, while the RLM and HLM results are inferior compared to the XGB. The GIN encoder shows similar results, with the difference being the outperforming HLM performance of the GIN model, while the LSTM seems to outperform the XGB on every endpoint.

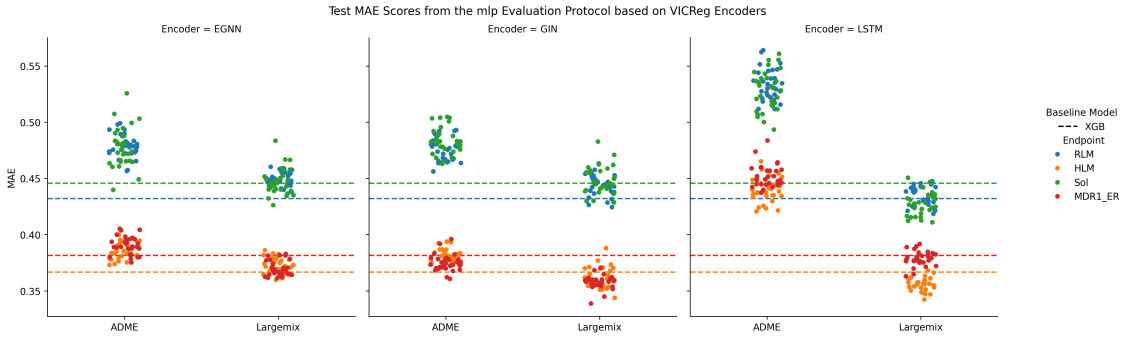

Figure 4.4: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the transfer learning evaluation protocol. ADME refers to the pre-training on the ADME training split, while largemix refers to the pre-training on the largemix datasetThe y-scale refers to the test MAE coming from an evaluation of the ADME dataset.

Supplement Figure B.16 displays the transfer learning evaluation for the test R2 metric. The GIN encoder yields analogous results compared to the MAE results for every endpoint, where the results from the largemix pre-training appear to be slightly better. The EGNN shows overall better test R2 scores with lower variance when pre-trained on the largemix data, which is also true to a larger extent for the LSTM encoder. Analogous to the linear evaluation differences, every transfer learning difference is significantly better when using the larger pre-training dataset for VICReg training on every metric-endpoint combination (see Supplement Figures B.19 and B.20 for GIN, Figures B.23 and B.24 for EGNN, and Figures B.27 and B.28 for LSTM).

4.2.2 Different Pre-training Strategies

Furthermore, the effect of using different pre-trained encoders for the VICReg training is investigated. For that, the results of a VICReg training with randomly initialized encoder weights, a VICReg model that uses the encoder weights of the model-agnostic pre-trained experiment, and a VICReg model that uses the encoder weights of the modality-specific pre-trained experiment are compared to show if a VICReg training with pre-trained encoders yields a benefit over a VICReg training using randomly initialized encoder weights.

Linear Evaluation

Figure 4.5 shows the results of the linear evaluation protocol for VICReg trained with either randomly initialized encoders (referred to as "Largemix"), encoders from the model-agnostic supervised pre-training (referred to as "Pre (Agnostic)"), and encoders from the modality-specific pre-training (referred to as "Pre (Specific)"). Starting with the GIN results, it is notable that the VICReg results based on the agnostic pre-trained encoders show lower test MAE scores for the endpoints RLM and HLM, while the results for the endpoints Sol and MDR1-ER seem worse compared to the other two weight initializations. The EGNN shows analogous results. However, the test MAE results of the randomly initialized VICReg appear to be worse than the pre-trained encoder variants for the LSTM. Thus, for the LSTM results, both VICReg runs with differently initialized encoder weights based on different pre-training strategies yield lower MAE scores for every endpoint, where the training with the agnostic pre-trained weights shows lower error scores for the endpoints RLM and HLM, while the training with the specialized pre-trained weights shows lower error scores for every endpoint. The comparison to the XGB is analogous to the previous comparison between the linear evaluated VICReg and XGB, since the baseline model outperforms every encoder on every endpoint.

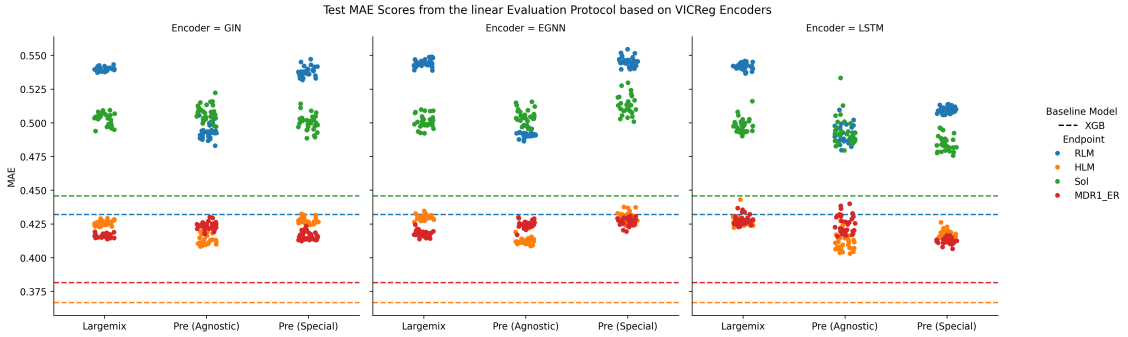

Figure 4.5: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the linear evaluation protocol. Largemix refers to a VICReg training with random initialization, Pre (Agnostic) refers to a VICReg training with agnostic pre-trained encoders, while Pre (Special) refers to a VICReg training with modality specific pre-trained encoders.

Supplement Figure B.29 shows the test R2 scores for this experiment. The GNN-based models show similar trends, where the training with the agnostic pre-trained encoders is superior to the other two alternatives. The specialized pre-trained encoder weights show no clear benefit, yielding similar test R2 scores compared to the random initialization. Analogous to the MAE results (Figure 4.5), the VICReg model with the agnostic pre-trained weights shows benefits for the endpoints RLM and HLM, while no

clear benefits are observable for the results of the endpoints Sol and MDR1-ER. The R2 results for the LSTM encoder show that the agnostic pre-trained weights provide better test R2 scores for every endpoint compared to the random initialization. Additionally, the modality-specific pre-trained weights yield better test results for every endpoint except RLM compared to the randomly initialized weight experiment.

Supplement Figures B.17, B.18, B.21, and B.22 show that the use of agnostic pre-trained encoder weights results in significantly better outcomes for the endpoints RLM and HLM when using GNN-based architectures. However, the results for the endpoints Sol and MDR1-ER are still superior when using the randomly initialized weights. Figures B.25 and B.26 show that either one of the pre-training weights demonstrates significantly better results depending on the endpoint. The use of the agnostic pre-trained encoder weights yields the best results for the endpoints RLM and HLM, while the specialized pre-trained weights result in the best outcomes for the endpoints Sol and MDR1-ER.

Transfer Learning

Figure 4.6 shows the transfer learning results for VICReg models trained with either randomly initialized or pre-trained encoder weights. The downstream task performance of the GIN encoder varies by the used encoder weights because the VICReg model using randomly initialized weights shows worse performance for the endpoints Sol and RLM compared to the test scores from the VICReg model with the pre-trained encoders. Furthermore, the MDR1-ER and HLM performance seem to be better when using the model-agnostic pre-trained encoder weights since the uncertainty of the VICReg trained with the randomly initialized encoders seems higher indicated by the larger variance of the test metric distributions. The EGNN results from the randomly initialized VICReg model and the model instance where the specialized pre-training weights were used are rather similar regarding the Sol, RLM, and HLM MAE performance. The randomly initialized model shows better MDR1-ER results compared to the VICReg experiment with the specialized pre-trained weights. Nevertheless, the EGNN results from the agnostic pre-trained experiment are generally better for any given endpoint when compared to the other two EGNN results. The LSTM also shows that the results from a VICReg training based on agnostic pre-trained encoder weights surpass the results from the random initialization and specialized pre-trained encoders. The randomly initialized LSTM results seem better than the results from the VICReg model using the specialized pre-trained encoders for every endpoint. Comparing the VICReg model with randomly initialized weights and the model using the supervised pre-trained encoders to the XGB baseline scores shows that the use of the pre-trained encoders results in a larger gap between the downstream task performance and the baseline model for every endpoint and every

encoder. The gap is usually larger for the endpoints MDR1-ER and HLM.

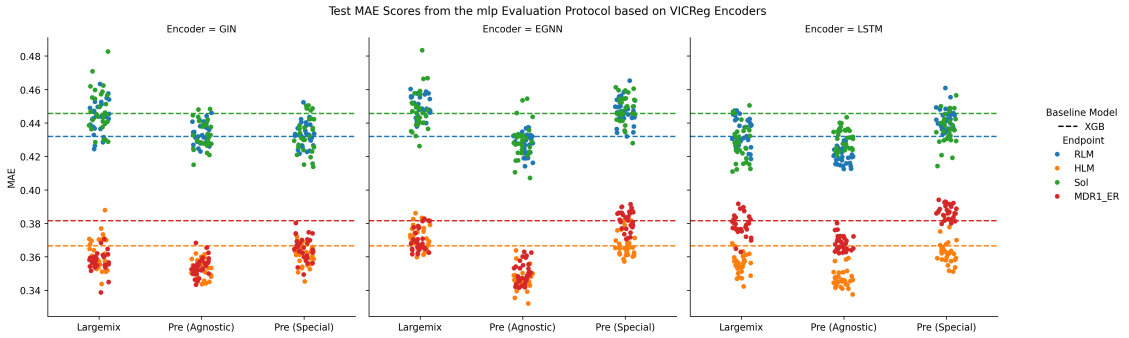

Figure 4.6: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the transfer learning evaluation protocol. Largemix refers to a VICReg training with random initialization, Pre (Agnostic) refers to a VICReg training with agnostic pre-trained encoders, while Pre (Special) refers to a VICReg training with modality specific pre-trained encoders. The y-scale refers to the test MAE coming from an evaluation of the ADME dataset.

Supplement Figure B.30 shows the respective results for the R2 test scores. All encoders show the same trend, where the results from the agnostic pre-trained encoders yield the highest test R2 scores for any tested endpoint except for the LSTM since the Sol performance is marginally worse for some random states compared to the other two evaluation strategies.

Supplement Figures B.33, B.34, B.37, and B.38 show that using the weights from agnostic pre-trained GNN models yields overall significantly better results for most of the endpoints. Only the endpoints RLM and Sol show that the modality-specific pre-trained GIN results are on par with the GIN results coming from the model-agnostic pre-trained GIN encoder, while the results from the randomly initialized VICReg model are always inferior. For every endpoint except Sol, the LSTM yields significantly better results when using the agnostic pre-trained encoder weights. The test R2 scores for Sol show no significant differences across all three experiments, while the test MAE results indicate that the random initialization yields similar results compared to the use of model-agnostic pre-trained encoders.

4.2.3 Impact of Differing VICReg Strategies

Both the VICReg pre-training on the ADME training split and the pre-training on the largemix data yield test performances showing that VICReg is capable of learning meaningful representations of molecules. However, the pre-training on the largemix results in superior test scores, underscoring the importance of dataset scale for self-supervised pre-

training [141], providing supporting evidence that the domain of computational chemistry is no exception [142]. The results also highlight that the regularization setting from the original VICReg [41] work is most likely to prevent the collapsing problem with $\lambda = \mu = 25, \nu = 1$, since not a single VICReg experiment suffered from representation collapse.

Furthermore, it appears that the weight initialization of the VICReg encoders significantly affects the performance of downstream tasks, outlining that better VICReg representations can be obtained by using pre-trained encoders for the VICReg training. For that, it appears that VICReg models using the agnostic pre-trained encoders yield the best test results. This observation aligns with the results from the pre-training experiments (Section 4.1), showing that the learned representations from the previously pre-trained encoders are further utilized, which affects the downstream task performance significantly. On the other hand, the modality-specific pre-trained encoders yield only significant benefits for some of the four downstream tasks. In some cases, the randomly initialized VICReg model shows superior performance compared to the use of the modality-specific pre-training weights (e.g., Endpoint HLM in Supplement Figure B.32 or endpoint MDR1-ER in Supplement Figures B.33, B.34, B.35, and B.36). This is more often the case for the EGNN encoder than for the other encoder types, outlining that the use of pre-trained encoders may not affect every encoder equally.

Ultimately, the use of different encoder weights reflects a similar trend to the pre-training evaluation without VICReg. This seems plausible since the exact encoder weights resulting from the pre-training experiments were used as initialization for the VICReg training, which heavily influences the training process. Consequently, the bioassay information regarding the downstream tasks, as well as the inferior performance coming from the self-supervised GNN pre-training, is reflected in the VICReg results. The superiority of the MLM-based pre-training for the LSTM encoder is not apparent in the VICReg results because the agnostic pre-trained encoder initialization dominates the LSTM results (See Supplement Figure B.39-B.42), indicating that the gained information from the self-supervised pre-trained LSTM is overshadowed by the information from the self-supervised pre-trained GNNs. Hence, the use of pre-trained encoders can significantly affect the VICReg training process for downstream performance and the representation capability of VICReg.

4.3 Across Experiments

Lastly, the results from the VICReg experiments are compared to the results from the pre-training experiments without VICReg to investigate if VICReg provides any benefits for training chemical foundation models. Therefore, the results from the VICReg models (either randomly initialized, initialized with agnostic pre-trained encoders, or initialized with specific pre-trained encoders) are compared with the results from the agnostic/specific pre-trained encoders without VICReg. The XGB results and the results of the tuned task-specific model architectures without pre-training are shown as references.

4.3.1 Linear Evaluation

Figure 4.7 displays the results for the linear evaluation protocol, reflecting the previously described results. For the GIN encoder, the supervised model-agnostic pre-training yields the best test MAE scores for any endpoint, visible through a rather large gap compared to the other experiments. The second-best results come from the node masking self-supervised pre-training, which yields the second-best results for any endpoint except MDR1-ER and RLM because the VICReg model with the random and the self-supervised pre-training initialization shows better MDR1-ER results and the VICReg model using the agnostic pre-trained encoders shows better RLM results. The remaining GIN results reflect the outcome of the previously discussed experiment, where the VICReg with the agnostic pre-trained weight initialization has the best performance except for the endpoint MDR1-ER. The EGNN shows similar results, where the supervised model-agnostic pre-training scores the best test MAE results with a large gap compared to the other experiments. The second-best results are endpoint-dependent, since the second-best MDR1-ER results come frome the VICReg strategy using the randomly initialized encoders, the second-best HLM and RLM results come from the VICReg model with the supervised agnostic pre-trained weight initialization, and the second-best Sol results come from the self-supervised pre-trained EGNN model. The LSTM results yield no model that is superior to the others since the ranking is endpoint-dependent. Therefore, the VICReg model with the agnostic pre-trained weight initialization yields the best RLM results, the self-supervised pre-trained encoder using a MLM provides the best Sol results, and the agnostic pre-trained encoder achieves the best MDR1-ER and HLM results. Using the XGB as a baseline model, it appears that a simple linear layer can achieve even higher results by using frozen representations in some cases. This observation is visible for the endpoints MDR1-ER when using the agnostic pre-training strategy for GNN-based models and Sol when using the self-supervised MLM-based pre-training.

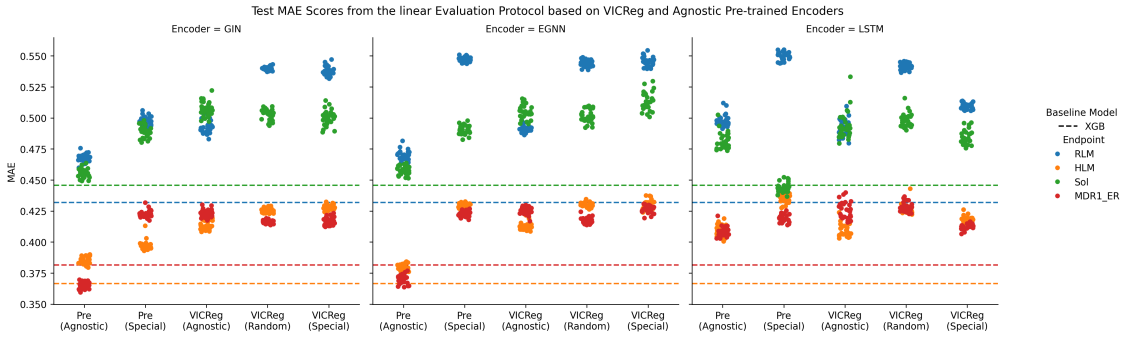

Figure 4.7: Aggregated VICReg and pre-trained results without VICReg for each encoder on each endpoint (color). Each point corresponds to a different random seed used in the linear evaluation protocol. Pre (Agnostic) refers to supervised agnostic pre-training, while Pre (Special) refers to modality-specific pre-trained encoders. VICReg (Agnostic) denotes the use of an agnostic pre-trained encoder for VICReg training, VICReg (Random) indicates VICReg pre-training with randomly initialized encoders, and VICReg (Special) shows results from VICReg with modality-specific pre-trained encoders. The y-axis represents the test MAE score obtained from evaluation on the ADME dataset.

Supplement Figure B.43 shows the test R2 scores for the same models. Again, the GNN-based models show similar observations, with the difference that the supervised agnostic pre-training achieves the best R2 scores for every endpoint. For the LSTM, the results show no model that achieves the best results for every endpoint at the same time. The best test R2 scores for MDR1-ER come from the VICReg model with the self-supervised initialized weights and the model-agnostic pre-training. The best RLM and HLM R2 scores are the outcome of the VICReg with the supervised agnostic pre-training initialization, while the best Sol scores originate from the self-supervised MLM approach. The GNN models pre-trained with the supervised schedule can still compete with the XGB test results regarding the endpoints MDR1-ER and Sol.

Supplement Figures B.45, B.46, B.49, and B.50 show that the supervised pre-trained GNN-based models achieve significant superiority regarding the linear evaluation protocol (excluding XGB) for both test metrics. Like before, there is no LSTM training strategy that yields the best results for every endpoint. Supplement Figures B.53 and B.54 show that the self-supervised MLM-based pre-training provides the best Sol scores, the supervised agnostic pre-training provides the best MDR1-ER scores, the VICReg model with the agnostic pre-trained weights provides the best RLM results, while the best HLM results are achieved by the agnostic pre-trained LSTM and the LSTM coming from the VICReg experiment with the model-agnostic pre-trained initialization.

4.3.2 Transfer Learning

Figure 4.8 shows the transfer learning results for the test MAE scores. This comparison also includes the results from the models involving optimized predictor and optimizer hyperparameters, which is referred to as "No Pre" (No Pre-training). The GIN results show that the supervised agnostic pre-trained GIN model achieves the best test MAE scores for every endpoint, which is closely followed by the results of the task-specific GIN architecture without pre-training. The EGNN results follow a similar trend, since the supervised agnostic pre-trained EGNN yields the best results, followed by the EGNN without pre-training. The best LSTM results are shared by the supervised agnostic and the self-supervised pre-training, since the MLM-based approach shows the best HLM, RLM, and Sol results, while the supervised pre-training provides the best MDR1-ER results. It is notable that every pre-training strategy achieves better results than the LSTM without pre-training. Comparing the pre-training results to the XGB baseline, it appears that almost every experiment reaches at least the same test MAE scores for the endpoints HLM and Sol, depending on the random state except for the modality specific pre-trained EGNN and the LSTM without pre-training. The same observation is visible for MDR1-ER, except for the LSTM model without pre-training and the one pre-trained with the VICReg model involving the specialized pre-trained encoders. The XGB's RLM performance is superior in most cases since only the optimized models without pre-training (except LSTM) or models using the agnostic pre-training outperform the baseline model on this endpoint.

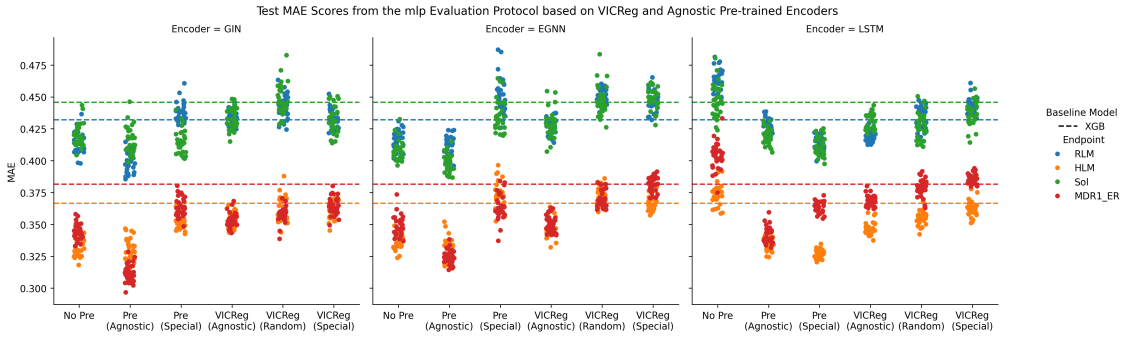

Figure 4.8: Aggregated VICReg and pre-trained results without VICReg for each encoder on each endpoint (color). Each point corresponds to a different random seed used in the linear evaluation protocol. The x-label "No Pre" shows the results without pre-training but with optimized predictor and optimizer parameters. Pre (Agnostic) refers to supervised agnostic pre-training, while Pre (Special) refers to modality-specific pre-trained encoders. VICReg (Agnostic) denotes the use of an agnostic pre-trained encoder for VICReg training, VICReg (Random) indicates VICReg pre-training with randomly initialized encoders, and VICReg (Special) shows results from VICReg with modality-specific pre-trained encoders. The y-axis represents the test MAE score obtained from evaluation on the ADME dataset.

Supplement Figure B.44 shows the test R2 scores for the same models. Again, the GNN-based models show similar observations to the MAE results, with the difference that the supervised agnostic pre-training achieves the best R2 scores for every endpoint, closely followed by the results from the model without pre-training. The highest test R2 scores from the LSTM are the outcome of the supervised and self-supervised pre-training closely followed by the VICReg, yielding similar results. Notably, the VICReg results with the supervised initialization are on par with the best results.

Supplement Figures B.47, B.48, B.51, and B.52 show that the supervised agnostic pre-training dominates the transfer learning evaluation protocol significantly for GNN-based models, followed by the VICReg with the agnostic pre-trained weights. Supplement Figures B.55 and B.56 show that either the self-supervised or the supervised pre-training leads to the best LSTM results with a significant difference.

4.3.3 Comparing VICReg to other Pre-training Strategies

The results show that the best downstream task evaluation performance scores originate from the supervised model-agnostic pre-training, highlighting the success of pre-training as a prevalent technique to leverage model performance [128, 80, 15, 142, 139]. However, the GNN-based models without pre-training yield the second-best results (See Figure 4.8), underlining that the models without pre-training exhibit two advantages

described by optimized predictor and optimizer hyperparameters. This finding indicates that the separate tuning schedule regarding the architecture and optimizer tuning strategy from the Google tuning guide [116] yields effective results. Additionally, those hyperparameters were tuned on a strict cross-validation evaluation schedule through the use of a similarity cross-validation based on the Butina [64] clustering algorithm, enhancing the generalization ability of those models.

The success of the supervised model-agnostic pre-training may originate from the fact that the supervised pre-training tasks contain bio-assays, describing a related domain to the ADME downstream tasks, since ADME describes biochemical properties of a drug. A beneficial effect using the PCBA bioassay data as supervised pre-training for biochemical-related tasks was already shown in an experiment by *Laufkötter et al.* [136], where pre-training on the PCBA dataset improved binding prediction tasks. Therefore, the learned representations from the model-agnostic pre-training and the models without pre-training yield good results for the ADME downstream task, showing that the labeled largemix data is appropriate for pre-training for biochemical-related downstream tasks and that task-specific models trained with rigid splitting techniques are still relevant. Nevertheless, an evaluation on a completely different downstream task would provide evidence of whether the supervised agnostic pre-training remains superior compared to the other pre-training methods. As previously discussed, the masking of node attributes alone may be to simply due to the high amount of carbon atoms as nodes, explaining the poor performance of the conducted self-supervised GNN pre-training experiments.

However, the evaluation scores of VICReg are often similar to the test scores of the self-supervised GNN pre-training methods, indicating that deploying VICReg to the domain of chemoinformatics doesn't provide any benefit based on the results of this experiment, since the supervised agnostic pre-training and the models without pre-training are outperforming VICReg on the ADME downstream task. Even so, this doesn't mean that VICReg isn't suitable as a method to create chemical foundation models, because the conducted experiments are all based on one VICReg setting derived from the original VICReg implementation [41]. An extensive ablation study is needed to fully reveal the potential of VICReg, similar to the ablation study of the original work. This could involve longer training, since the VICReg model is trained for 30 epochs due to training convergence, experiments with different batch sizes, experiments with larger encoder settings since the encoder settings are tied to the hyperparameter optimization on the small ADME training split, experiments using even more data since labels are not needed, experiments using shared weights architectures [83] like in the original VICReg paper, experiments using different projector sizes, or experiments that test different regularization settings for the VICReg hyperparameters λ, μ, and ν.

Nonetheless, the adaptation of VICReg to the domain of computational chemistry is possible without encountering a representation collapse and still leverages the LSTM downstream performance compared to the LSTM without pre-training, indicating that language models with self-attention benefit from pre-training on large datasets [3, 15, 142]. Regardless, this was not the case for the GNN-based models, since the task-specific results are outperforming the self-supervised pre-training experiments. Additionally, the MLM-based self-supervised pre-training shows results that are at least on par with the supervised agnostic pre-training results, demonstrating that techniques from the field of NLP can be successfully adapted to the domain of chemoinformatics. This highlights the use of SMILES augmentations [34] due to the ease of generating larger datasets from scratch, which is particularly useful for self-supervised approaches. Also, most of the pre-training procedures (including the VICReg results) yield results that are at least as good as the XGB baseline results, which are powerful on their own, because the XGB models are tuned on the single-task endpoints using the same rigid cross-validation tactic.

Consequently, VICReg is capable of yielding meaningful representations in a self-supervised manner, meaning that labels are not needed for model training, making it a promising technique for building chemical foundation models that scale with the amount of available unlabeled training data. Further experiments need to be conducted to investigate which VICReg settings are preferable in the domain of computational chemistry, since the performance of GNNs and MLM-pre-trained LSTMs are powerful alternatives that outperformed the VICReg results in the described experimental settings.

Another notable observation is the performance trend regarding the test metrics of the different ADME downstream tasks, namely the endpoints RLM, HLM, Sol, and MDR1-ER. In most of the results, the endpoint MDR1-ER is coupled with the best performance metrics, followed by HLM. The other two endpoints (RLM and Sol) are assigned lower performance scores overall, indicating that the downstream tasks are correlated with each other. Figure 4.9 shows the pairwise correlation matrix between the targets of the downstream ADME tasks. The pairwise correlations are only calculated if two entries for the same molecule exist. The matrix visualizes the Pearson correlation $\rho \in [-1, 1]$, where a value near 0 indicates no correlation. It appears that the endpoints RLM and HLM are highly correlated, with a correlation coefficient $\rho = 0.76$. This is plausible since RLM and HLM are described by the same in vitro assay [95], with the only difference being that the liver microsomes come from rats (RLM) [143] instead of humans (HLM), capturing similar ADME properties. Another positive correlation is observable between Sol and MDR1-ER, with a $\rho = 0.18$, describing a relationship between the solubility and permeability properties of a molecule [144]. In contrast to

the named correlations, further negative correlations are apparent between the Clint assays (RLM and HLM), permeability assay (MDR1-ER), and the solubility assay (Sol), indicating that the ADME downstream tasks are connected to each other [145, 146, 147], making the ADME dataset suitable for multi-task learning [68].

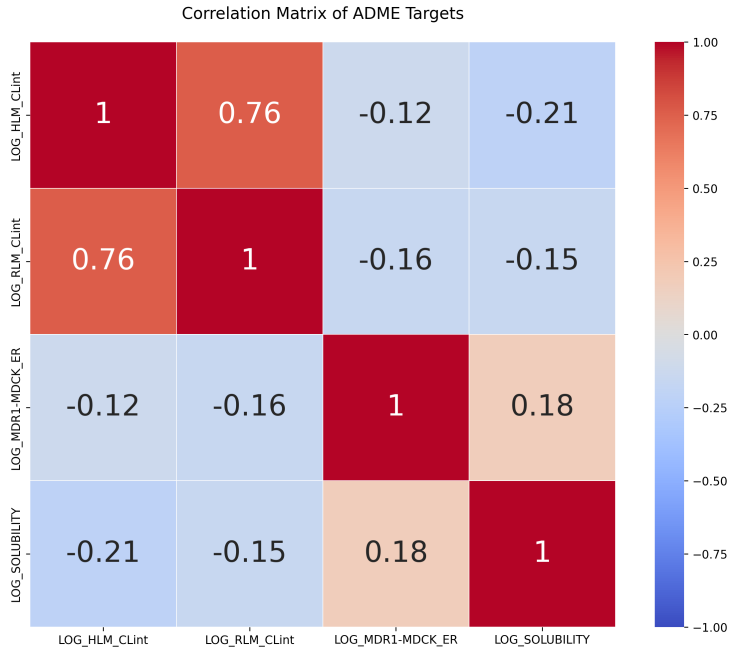

Figure 4.9: The correlations between downstream tasks reveal that ADME properties are related to each other which is in particular visible for RLM and HLM.

Moreover, the results for the endpoint Sol contain lower test R2 and higher test MAE scores compared to the other endpoints (See Figures 4.1-4.8). This reflects inherent limitations in representing solubility through molecular features alone, which isn't an uncommon problem for ADME prediction tasks [148, 95]. Similarly, the results for the endpoint RLM also have higher test MAE values compared to the other endpoint results. A possible explanation could be described by the correlation between RLM, Sol, and the other endpoints. Thus, the multi-task training leverages the predictions on the other endpoints [143] at the cost of the RLM and Sol performances due to the aggregated masked multi-task loss function (See Equation 3.4), as the individual loss values are combined using the mean operation. This aligns with observations in multi-task learning where negative transfer arises from conflicting gradient dynamics or misaligned task hierarchies [149, 150]. Future work could explore task-specific weighting schemes or hybrid single/multi-task architectures to mitigate these trade-offs. However, the single-task XGB test performances follow the same pattern, where the RLM and Sol test scores are generally worse than the test performance regarding the endpoints HLM

and MDR1-ER. Therefore, another hypothesis could be that the similarity-based train and test split is simply more favorable for the endpoints HLM and MDR1-ER.

Ultimately, the experiments reveal that while supervised pre-training and non-pre-trained GNNs outperform VICReg on ADME tasks, VICReg's self-supervised approach shows promise for label-free foundation models, particularly benefiting LSTMs through pre-training on SMILES strings. Despite VICReg's current limitations, potentially due to suboptimal hyperparameters or a lack of diversity in downstream tasks, its adaptability and the success of MLM-based methods highlight the viability of state-of-the-art methods from other domains such as CV (VICReg) and NLP (MLM) in chemoinformatics. Future work should explore architectural adjustments, extended training, and task-specific optimization to unlock VICReg's full potential.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This work has explored the adaptation of VICReg to the domain of computational chemistry by employing three different chemical modalities for self-supervised learning. In particular, a GIN encoder was used to assess the graph modality, an EGNN encoder was used to provide the conformer modality, and an LSTM encoder produced the chemical language representation of a molecule.

In addition to VICReg pre-training, a supervised model-agnostic pre-training approach and self-supervised, modality-specific pre-training techniques were developed: a MLM-based approach was used for the LSTM encoder and a node-attribute masking approach was used for the GNN-based architectures. All three pre-training architectures were trained on a large-scale public dataset suitable for creating chemical foundation models. The pre-trained models were evaluated using both linear and transfer learning evaluation protocols on four different ADME prediction downstream tasks. Single-task XGB and multi-task GIN, EGNN, and LSTM models were optimized on the ADME data, serving as baseline results. The following findings provide answers to the previously defined research questions (see Section 1):

(i) Based on the conducted experiments, the supervised, model-agnostic pre-training strategy generally outperformed the self-supervised, modality-specific approach.

(ii) This thesis shows that a multi-modal adaptation of VICReg to computational chemistry is both feasible and effective. By integrating a graph (GIN), a conformer (EGNN), and a chemical language (LSTM) modality, the approach successfully learns molecular representations without encountering a representation collapse, highlighting the method's flexibility and robustness. Furthermore, VICReg performed at least as good as the basline XGB for some of the endpoints using the

transfer learning evaluation protocol, indicating that the learned molecule representations are meaningful.

(iii) VICReg performed significantly better on the largemix dataset compared to the pre-training on the ADME training split. Also, the VICReg model using pre-trained weights from the supervised agnostic pre-training performed significantly better than the randomly initialized VICReg outlining that VICReg shows a lot of potential if the right experimental setting can be provided.

(iv) VICReg can compete with the baseline XGB and the self-supervised GNN results on some endpoints but the supervised model-agnostic pre-trained model strategy outperformed VICReg on every downstream task for every encoder type labeling VICReg as inferior compared to the supervised pre-training based on the conducted experimental setting. Moreover, the other self-supervised pre-training results are also at least as good as VICReg's results indicating that other self-supervised learning strategies might be preferable over VICReg depending on the encoder and the experimental setting.

Lastly, the insights gained from the rigid train-test splitting approach, hyperparameter optimization, and the integration of diverse molecule modalities suggest promising starting points for future research. Potential directions include exploring even larger and more heterogeneous compound databases since VICReg learns in a self-supervised manner and does not need expensively labeled data. Furthermore, this thesis only scratched the surface regarding a chemical foundation model based on VICReg since a variety of further experiments need to be done to uncover the method's true potential in capturing deeper and more robust chemical representations without collapsing.

5.2 Future work

The results show that simple node-attribute masking self-supervised pre-training is inferior to model-agnostic pre-training for GNN-based models. Therefore, more sophisticated self-supervised pre-training strategies could provide a better option for investigating how self-supervised pre-trained encoders affect downstream task performance in combination with VICReg. Furthermore, in this thesis, GINs, EGNNs, and LSTMs are used as the respective modality encoders due to their prominent status in the domain of chemoinformatics. Nevertheless, foundation models are often based on the transformer architecture, making them large and complex to capture intricate data patterns [8, p.47]. Thus, an interesting direction for future work would be to investigate how VICReg performance changes when using more complex, modality-specific encoders such as molecular BERT [33] for the text modality, the Equiformer [151] for the conformer modality, and the Graphformer [152] for the graph modality.

The XGB results also indicate that descriptor- and fingerprint-based featurization strategies remain powerful. Hence, another future experiment could incorporate the tabular representation as a fourth modality to explore whether a static, deterministic tabular representation positively affects the representation capability of the other encoders. Similarly, an image-based encoder could be used as a fifth modality by utilizing images of the structural formulas. Another promising approach would be to use even larger datasets for self-supervised pre-training, such as the ultralarge dataset [88], since the largemix dataset has already demonstrated that the capability of foundation models scales with the amount of data and expensive labels are not needed. Additionally, encoder parameters could be adjusted to better accommodate larger datasets, as the current hyperparameter choices are all optimized on the training split of the ADME dataset.

Additionally, more experiments on a broader diversity of downstream tasks could be conducted to gain a better understanding of the representations learned by the respective encoders, since the ADME tasks all originate from the same field of PK. For example, incorporating a classification problem such as predicting material property categories [153] would provide an insightful enhancement as a downstream task, complementing the regression-focused ADME benchmarks. This would not only test the generalizability of the learned representations across different task types but also validate whether their robustness extends to chemically distinct domains, such as material science, beyond PK.

Furthermore, an ablation study similar to that in the original VICReg paper could illustrate how different VICReg settings impact downstream performance. Important variables for such a study would include batch size, projector size, the use of shared weights for the projector, and different values of λ, μ, ν as regularization terms. Another interesting experiment could examine the benefits of bi-modal settings, where only two encoders are used, to investigate if the aligned representations differ from those produced in the VICReg experiment with three encoders.

The original VICReg paper used a siamese weight-sharing [83] encoder and projector; this approach could also be applied to the chemical language modality, since SMILES augmentations [34] are analogous to data augmentation techniques. Therefore, it would be worthwhile to pursue a VICReg approach using only one encoder with augmented views of the same SMILES string to closely mimic the original VICReg work. A similar strategy could be applied to the conformer modality, as multiple conformers can be generated from a single molecule. Additionally, knowledge-infused graph augmentation techniques could be employed to utilize siamese architectures for the graph modality [154].

Finally, another benchmark objective could involve comparing the uncertainty estimation of models trained with and without VICReg, to investigate whether pre-training with VICReg yields improved uncertainty estimates using model-agnostic uncertainty methods such as conformal prediction [155] or ordinal confidence level assignments [156].

Bibliography

[1] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[5] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[6] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[7] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[8] C. Huyen, *AI Engineering: Building Applications with Foundation Models*. O'Reilly, 2025.

[9] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[11] C. Akkus, L. Chu, V. Djakovic, S. Jauch-Walser, P. Koch, G. Loss, C. Marquardt, M. Moldovan, N. Sauter, M. Schneider, *et al.*, "Multimodal deep learning," *arXiv preprint arXiv:2301.04856*, 2023.

[12] P. P. Liang, A. Zadeh, and L.-P. Morency, "Foundations and trends in multimodal machine learning: Principles, challenges, and open questions," *arXiv preprint arXiv:2209.03430*, 2022.

[13] M. Assran, Q. Duval, I. Misra, P. Bojanowski, P. Vincent, M. Rabbat, Y. LeCun, and N. Ballas, "Self-supervised learning from images with a joint-embedding predictive architecture," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15619–15629, 2023.

[14] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*, pp. 8748–8763, PmLR, 2021.

[15] J. Lu, D. Batra, D. Parikh, and S. Lee, "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks," *Advances in neural information processing systems*, vol. 32, 2019.

[16] J. Choi, G. Nam, J. Choi, and Y. Jung, "A perspective on foundation models in chemistry," *JACS Au*, vol. 5, no. 4, pp. 1499–1518, 2025.

[17] Y. Chang, B. A. Hawkins, J. J. Du, P. W. Groundwater, D. E. Hibbs, and F. Lai, "A guide to in silico drug design," *Pharmaceutics*, vol. 15, no. 1, p. 49, 2023.

[18] T. Casalini, "Not only in silico drug discovery: Molecular modeling towards in silico drug delivery formulations," *Journal of Controlled Release*, vol. 332, pp. 390–417, 2021.

[19] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[20] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[21] J. Xu and A. Hagler, "Chemoinformatics and drug discovery," *Molecules*, vol. 7, no. 8, pp. 566–600, 2002.

[22] D. Boldini, F. Grisoni, D. Kuhn, L. Friedrich, and S. A. Sieber, "Practical guidelines for the use of gradient boosting for molecular property prediction," *Journal of Cheminformatics*, vol. 15, no. 1, p. 73, 2023.

[23] R. D. Jawarkar, S. Mali, P. K. Deshmukh, R. G. Ingle, S. A. Al-Hussain, A. A. Al-Mutairi, and M. E. Zaki, "Synergizing ga-xgboost and qsar modeling: Breaking down activity aliffs in hdac1 inhibitors," *Journal of Molecular Graphics and Modelling*, vol. 135, p. 108915, 2025.

[24] A. U. Khan *et al.*, "Descriptors and their selection methods in qsar analysis: paradigm for drug design," *Drug discovery today*, vol. 21, no. 8, pp. 1291–1302, 2016.

[25] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery," *Drug discovery today*, vol. 23, no. 6, pp. 1241–1250, 2018.

[26] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, *et al.*, "Analyzing learned molecular representations for property prediction," *Journal of chemical information and modeling*, vol. 59, no. 8, pp. 3370–3388, 2019.

[27] B. Chen, R. Barzilay, and T. Jaakkola, "Path-augmented graph transformer network," *arXiv preprint arXiv:1905.12712*, 2019.

[28] J. Li, D. Cai, and X. He, "Learning graph-level representation for drug discovery," *arXiv preprint arXiv:1709.03741*, 2017.

[29] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2022.

[30] D. Jiang, Z. Wu, C.-Y. Hsieh, G. Chen, B. Liao, Z. Wang, C. Shen, D. Cao, J. Wu, and T. 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*, vol. 13, pp. 1–23, 2021.

[31] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *International conference on machine learning*, pp. 9323–9332, PMLR, 2021.

[32] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of chemical information and computer sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[33] J. Li and X. Jiang, "Mol-bert: An effective molecular representation with bert for molecular property prediction," *Wireless Communications and Mobile Computing*, vol. 2021, no. 1, p. 7181815, 2021.

[34] T. B. Kimber, M. Gagnebin, and A. Volkamer, "Maxsmi: Maximizing molecular property prediction performance with confidence estimation using smiles augmentation and deep learning," *Artificial Intelligence in the Life Sciences*, vol. 1, p. 100014, 2021.

[35] R. Yin, R. Liu, X. Hao, X. Zhou, Y. Liu, C. Ma, and W. Wang, "Multimodal molecular representation learning via structure awareness," *arXiv preprint arXiv:2505.05877*, 2025.

[36] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020.

[37] Z. Wang, J. Mi, S. Lu, and J. He, "Multimodal-learning for predicting molecular properties: A framework based on image and graph structures," *arXiv preprint arXiv:2311.16666*, 2023.

[38] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 2, pp. 1735–1742, IEEE, 2006.

[39] T.-H. Nguyen-Vo, P. Teesdale-Spittle, J. E. Harvey, and B. P. Nguyen, "Molecular representations in bio-cheminformatics," *Memetic Computing*, vol. 16, no. 3, pp. 519–536, 2024.

[40] J. Wu, Y. Su, A. Yang, J. Ren, and Y. Xiang, "An improved multi-modal representation-learning model based on fusion networks for property prediction in drug discovery," *Computers in Biology and Medicine*, vol. 165, p. 107452, 2023.

[41] A. Bardes, J. Ponce, and Y. LeCun, "Vicreg: Variance-invariance-covariance regularization for self-supervised learning," *arXiv preprint arXiv:2105.04906*, 2021.

[42] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PmLR, 2020.

[43] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9729–9738, 2020.

[44] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21271–21284, 2020.

[45] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *Advances in neural information processing systems*, vol. 33, pp. 9912–9924, 2020.

[46] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

[47] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," *arXiv preprint arXiv:1810.00826*, 2018.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] J. Cremer, L. Medrano Sandonas, A. Tkatchenko, D.-A. Clevert, and G. De Fabritiis, "Equivariant graph neural networks for toxicity prediction," *Chemical Research in Toxicology*, vol. 36, no. 10, pp. 1561–1573, 2023.

[50] Y. Peng, Y. Lin, X.-Y. Jing, H. Zhang, Y. Huang, and G. S. Luo, "Enhanced graph isomorphism network for molecular admet properties prediction," *Ieee Access*, vol. 8, pp. 168344–168360, 2020.

[51] J. Guan and J. Liu, "Lstm molecular descriptor-free qsar application research based on genetic algorithm optimization," in *2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pp. 625–628, IEEE, 2022.

[52] L. Jing, P. Vincent, Y. LeCun, and Y. Tian, "Understanding dimensional collapse in contrastive self-supervised learning," *arXiv preprint arXiv:2110.09348*, 2021.

[53] S. Grogan and C. V. Preuss, "Pharmacokinetics," in *StatPearls [Internet]*, StatPearls Publishing, 2023.

[54] P. Wu, S. Lin, G. Cao, J. Wu, H. Jin, C. Wang, M. H. Wong, Z. Yang, and Z. Cai, "Absorption, distribution, metabolism, excretion and toxicity of microplastics in the human body and health implications," *Journal of Hazardous Materials*, vol. 437, p. 129361, 2022.

[55] W. Goodwin, "Structural formulas and explanation in organic chemistry," *Foundations of chemistry*, vol. 10, pp. 117–127, 2008.

[56] Fdardel, "deriving the smiles representation of a chemical molecule, shown example: ciprofloxacin, a fluoroquinolone antibiotic.." Wikimedia Commons, 2010. Slight edit by DMacks.

[57] R. Todeschini and V. Consonni, *Molecular descriptors for chemoinformatics: volume I: alphabetical listing/volume II: appendices, references*. John Wiley & Sons, 2009.

[58] S. Hayat, S. Wang, and J.-B. Liu, "Valency-based topological descriptors of chemical networks and their applications," *Applied Mathematical Modelling*, vol. 60, pp. 164–178, 2018.

[59] L. Xu, H.-Y. Wang, and Q. Su, "A newly proposed molecular topological index for the discrimination of cis/trans isomers and for the studies of qsar/qspr," *Computers & chemistry*, vol. 16, no. 3, pp. 187–194, 1992.

[60] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of chemical information and modeling*, vol. 50, no. 5, pp. 742–754, 2010.

[61] H. L. Morgan, "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service.," *Journal of chemical documentation*, vol. 5, no. 2, pp. 107–113, 1965.

[62] T. T. Tanimoto, *An elementary mathematical theory of classification and prediction*. International Business Machines Corporation, 1958.

[63] T. G. Kristensen, J. Nielsen, and C. N. Pedersen, "A tree-based method for the rapid screening of chemical fingerprints," *Algorithms for Molecular Biology*, vol. 5, pp. 1–10, 2010.

[64] D. Butina, "Unsupervised data base clustering based on daylight's fingerprint and tanimoto similarity: A fast and automated way to cluster small and large data sets," *Journal of Chemical Information and Computer Sciences*, vol. 39, no. 4, pp. 747–750, 1999.

[65] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[66] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[67] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[68] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, pp. 41–75, 1997.

[69] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[70] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[71] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, pp. 1310–1318, Pmlr, 2013.

[72] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Diploma, Technische Universität München*, vol. 91, no. 1, p. 31, 1991.

[73] J. Luo, Z. Zhang, Y. Fu, and F. Rao, "Time series prediction of covid-19 transmission in america using lstm and xgboost algorithms," *Results in Physics*, vol. 27, p. 104462, 2021.

[74] F. Deng, Z. Chen, Y. Liu, S. Yang, R. Hao, and L. Lyu, "A novel combination neural network based on convlstm-transformer for bearing remaining useful life prediction," *Machines*, vol. 10, no. 12, p. 1226, 2022.

[75] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[76] L. Tunstall, L. Von Werra, and T. Wolf, *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[77] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[78] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[79] P. Maheshwari, "Self-supervised learning for graphs." `https://medium.com/stanford-cs224w/self-supervised-learning-for-graphs-963e03b9f809`, 2021. Accessed: 2023-04-29.

[80] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," *arXiv preprint arXiv:1905.12265*, 2019.

[81] P. Tang, C. Xie, and H. Duan, "Node and edge dual-masked self-supervised graph representation," *Knowledge and Information Systems*, vol. 66, no. 4, pp. 2307–2326, 2024.

[82] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[83] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, 1993.

[84] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate research*, vol. 30, no. 1, pp. 79–82, 2005.

[85] G. Casella and R. Berger, *Statistical inference*. CRC press, 2024.

[86] K. Backhaus, B. Erichson, and R. Weiber, *Fortgeschrittene multivariate Analysemethoden: eine anwendungsorientierte Einführung*. Springer-Verlag, 2015.

[87] D. C. Howell, *Statistical methods for psychology*. PWS-Kent Publishing Co, 1992.

[88] D. Beaini, S. Huang, J. A. Cunha, Z. Li, G. Moisescu-Pareja, O. Dymov, S. Maddrell-Mander, C. McLean, F. Wenkel, L. Müller, *et al.*, "Towards foundational models for molecular learning on large-scale multi-task datasets," *arXiv preprint arXiv:2310.04292*, 2023.

[89] M. Nakata and T. Shimazaki, "Pubchemqc project: a large-scale first-principles electronic structure database for data-driven chemistry," *Journal of chemical information and modeling*, vol. 57, no. 6, pp. 1300–1308, 2017.

[90] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, *et al.*, "Pubchem 2025 update," *Nucleic Acids Research*, vol. 53, no. D1, pp. D1516–D1525, 2025.

[91] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22118–22133, 2020.

[92] M. C. for Molecular Therapeutics (Massachusetts General Hospital), "Mgh (cmt) growth inhibition assay protocol (3 compound doses) – dna staining," 2014.

[93] A. Subramanian, R. Narayan, S. M. Corsello, D. D. Peck, T. E. Natoli, X. Lu, J. Gould, J. F. Davis, A. A. Tubelli, J. K. Asiedu, *et al.*, "A next generation connectivity map: L1000 platform and the first 1,000,000 profiles," *Cell*, vol. 171, no. 6, pp. 1437–1452, 2017.

[94] M. Sypetkowski, F. Wenkel, F. Poursafaei, N. Dickson, K. Suri, P. Fradkin, and D. Beaini, "On the scalability of gnns for molecular graphs," *Advances in Neural Information Processing Systems*, vol. 37, pp. 19870–19906, 2024.

[95] C. Fang, Y. Wang, R. Grater, S. Kapadnis, C. Black, P. Trapa, and S. Sciabola, "Prospective validation of machine learning algorithms for absorption, distribution, metabolism, and excretion prediction: An industrial perspective," *Journal of Chemical Information and Modeling*, vol. 63, no. 11, pp. 3263–3274, 2023.

[96] G. A. Landrum and S. Riniker, "Combining ic50 or k i values from different sources is a source of significant noise," *Journal of chemical information and modeling*, vol. 64, no. 5, pp. 1560–1567, 2024.

[97] J. R. Ash, C. Wognum, R. Rodríguez-Pérez, M. Aldeghi, A. C. Cheng, D.-A. Clevert, O. Engkvist, C. Fang, D. J. Price, J. M. Hughes-Oliver, *et al.*, "Practically significant method comparison protocols for machine learning in small molecule drug discovery.," 2024.

[98] L. Di, E. H. Kerns, and G. T. Carter, "Drug-like property concepts in pharmaceutical design," *Current pharmaceutical design*, vol. 15, no. 19, pp. 2184–2194, 2009.

[99] X.-T. D. Tran, T.-L. Phan, V.-T. To, N.-V. N. Tran, N.-N. S. Nguyen, D.-N. H. Nguyen, N.-T. N. Tran, and T. N. Truong, "Integration of the butina algorithm and ensemble learning strategies for the advancement of a pharmacophore ligand-based model: an in silico investigation of apelin agonists," *Frontiers in Chemistry*, vol. 12, p. 1382319, 2024.

[100] Chemaxon, "Bemis-murcko clustering," 2024. Accessed: 2024-04-28.

[101] G. W. Bemis and M. A. Murcko, "The properties of known drugs. 1. molecular frameworks," *Journal of medicinal chemistry*, vol. 39, no. 15, pp. 2887–2893, 1996.

[102] J. A. Napoli, M. Reutlinger, P. Brandl, W. Wang, J. Hert, and P. Desai, "Multitask deep learning models of combined industrial absorption, distribution, metabolism, and excretion datasets to improve generalization," *Molecular Pharmaceutics*, vol. 22, no. 4, pp. 1892–1900, 2025.

[103] J. Wenzel, H. Matter, and F. Schmidt, "Predictive multitask deep neural network models for adme-tox properties: learning from large data sets," *Journal of chemical information and modeling*, vol. 59, no. 3, pp. 1253–1268, 2019.

[104] G. Landrum, "Rdkit: Open-source cheminformatics software," 2016.

[105] K. Atz, F. Grisoni, and G. Schneider, "Geometric deep learning on molecular representations," *Nature Machine Intelligence*, vol. 3, no. 12, pp. 1023–1032, 2021.

[106] H. P. Latscha, U. Kazmaier, and H. A. Klein, *Organische Chemie: Chemie-Basiswissen II*. Springer-Verlag, 2008.

[107] J. M. Blaney and J. S. Dixon, "Distance geometry in molecular modeling," *Reviews in computational chemistry*, pp. 299–335, 1994.

[108] S. Riniker and G. A. Landrum, "Better informed distance geometry: using what we know to improve conformation generation," *Journal of chemical information and modeling*, vol. 55, no. 12, pp. 2562–2574, 2015.

[109] T. Seidel, C. Permann, O. Wieder, S. M. Kohlbacher, and T. Langer, "High-quality conformer generation with conforge: algorithm and performance assessment," *Journal of Chemical Information and Modeling*, vol. 63, no. 17, pp. 5549–5570, 2023.

[110] T. A. Halgren, "Mmff vi. mmff94s option for energy minimization studies," *Journal of computational chemistry*, vol. 20, no. 7, pp. 720–729, 1999.

[111] S. Wang, J. Witek, G. A. Landrum, and S. Riniker, "Improving conformer generation for small rings and macrocycles based on distance geometry and experimental torsional-angle preferences," *Journal of chemical information and modeling*, vol. 60, no. 4, pp. 2044–2058, 2020.

[112] Z. Xie, X. Evangelopoulos, Ö. H. Omar, A. Troisi, A. I. Cooper, and L. Chen, "Fine-tuning gpt-3 for machine learning electronic and functional properties of organic molecules," *Chemical science*, vol. 15, no. 2, pp. 500–510, 2024.

[113] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[114] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[115] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[116] V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue, and Z. Nado, "Deep learning tuning playbook," 2023. Version 1.

[117] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[118] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[119] Y. Bai, E. Yang, B. Han, Y. Yang, J. Li, Y. Mao, G. Niu, and T. Liu, "Understanding and improving early stopping for learning with noisy labels," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24392–24403, 2021.

[120] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[121] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[122] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, PMLR, 2016.

[123] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[124] N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter, "Tabpfn: A transformer that solves small tabular classification problems in a second," *arXiv preprint arXiv:2207.01848*, 2022.

[125] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18932–18943, 2021.

[126] F. Kanavati and M. Tsuneki, "Partial transfusion: on the expressive influence of trainable batch norm parameters for transfer learning," in *Medical Imaging with Deep Learning*, pp. 338–353, PMLR, 2021.

[127] D. Misra, "Mish: A self regularized non-monotonic activation function," *arXiv preprint arXiv:1908.08681*, 2019.

[128] X. Jiang, X. Cheng, and Z. Li, "Why pre-training is beneficial for downstream classification tasks?," *arXiv preprint arXiv:2410.08455*, 2024.

[129] D. Hendrycks, K. Lee, and M. Mazeika, "Using pre-training can improve model robustness and uncertainty," in *International conference on machine learning*, pp. 2712–2721, PMLR, 2019.

[130] M. Xu, H. Wang, B. Ni, H. Guo, and J. Tang, "Self-supervised graph-level representation learning with local and global structure," in *International conference on machine learning*, pp. 11548–11558, PMLR, 2021.

[131] S. Liu, M. F. Demirel, and Y. Liang, "N-gram graph: Simple unsupervised representation for graphs, with applications to molecules," *Advances in neural information processing systems*, vol. 32, 2019.

[132] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[133] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[134] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[135] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, "Massively multitask networks for drug discovery," *arXiv preprint arXiv:1502.02072*, 2015.

[136] O. Laufkötter, N. Sturm, J. Bajorath, H. Chen, and O. Engkvist, "Combining structural and bioactivity-based fingerprints improves prediction performance and scaffold hopping capability," *Journal of cheminformatics*, vol. 11, pp. 1–14, 2019.

[137] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

[138] Y. Wang, J. Wang, Z. Cao, and A. Barati Farimani, "Molecular contrastive learning of representations via graph neural networks," *Nature Machine Intelligence*, vol. 4, no. 3, pp. 279–287, 2022.

[139] N. Navarin, D. V. Tran, and A. Sperduti, "Pre-training graph neural networks with kernels," *arXiv preprint arXiv:1811.06930*, 2018.

[140] Y. Bai, H. Ding, Y. Qiao, A. Marinovic, K. Gu, T. Chen, Y. Sun, and W. Wang, "Unsupervised inductive whole-graph embedding by preserving graph proximity," in *Proceedings of the seventh international conference on learning representations (ICLR 2019)*, 2019.

[141] Y. Liu, J. Cao, C. Liu, K. Ding, and L. Jin, "Datasets for large language models: A comprehensive survey," *arXiv preprint arXiv:2402.18041*, 2024.

[142] S. Chithrananda, G. Grand, and B. Ramsundar, "Chemberta: large-scale self-supervised pretraining for molecular property prediction," *arXiv preprint arXiv:2010.09885*, 2020.

[143] P. Shah, V. B. Siramshetty, E. Mathé, and X. Xu, "Developing robust human liver microsomal stability prediction models: Leveraging inter-species correlation with rat data," *Pharmaceutics*, vol. 16, no. 10, p. 1257, 2024.

[144] V. Pade and S. Stavchansky, "Link between drug absorption solubility and permeability measurements in caco-2 cells," *Journal of pharmaceutical sciences*, vol. 87, no. 12, pp. 1604–1607, 1998.

[145] C. Lipinski, "a, lombardo, f., dominy, bw & feeney, pj experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings," *Adv. Drug Deliv. Rev*, vol. 46, no. 3, pp. 00129–0, 2001.

[146] L. Di, P. Artursson, A. Avdeef, L. Z. Benet, J. B. Houston, M. Kansy, E. H. Kerns, H. Lennernäs, D. A. Smith, and K. Sugano, "The critical role of passive permeability in designing successful drugs," *ChemMedChem*, vol. 15, no. 20, pp. 1862–1874, 2020.

[147] L. Z. Benet, C. M. Hosey, O. Ursu, and T. I. Oprea, "Bddcs, the rule of 5 and drugability," *Advanced drug delivery reviews*, vol. 101, pp. 89–98, 2016.

[148] P. Llompart, C. Minoletti, S. Baybekov, D. Horvath, G. Marcou, and A. Varnek, "Will we ever be able to accurately predict solubility?," *Scientific Data*, vol. 11, no. 1, p. 303, 2024.

[149] A. Lakkapragada, E. Sleiman, S. Surabhi, and D. P. Wall, "Mitigating negative transfer in multi-task learning with exponential moving average loss weighting strategies (student abstract)," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 16246–16247, 2023.

[150] D. Li, H. L. Nguyen, and H. R. Zhang, "Identification of negative transfers in multitask learning using surrogate models," *arXiv preprint arXiv:2303.14582*, 2023.

[151] Y.-L. Liao and T. Smidt, "Equiformer: Equivariant graph attention transformer for 3d atomistic graphs," *arXiv preprint arXiv:2206.11990*, 2022.

[152] J. Yang, Z. Liu, S. Xiao, C. Li, D. Lian, S. Agrawal, A. Singh, G. Sun, and X. Xie, "Graphformers: Gnn-nested transformers for representation learning on textual graph," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28798–28810, 2021.

[153] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer, *et al.*, "Graph neural networks for

materials science and chemistry," *Communications Materials*, vol. 3, no. 1, p. 93, 2022.

[154] H. An, X. Liu, W. Cai, and X. Shao, "Explainable graph neural networks with data augmentation for predicting p k a of c–h acids," *Journal of Chemical Information and Modeling*, vol. 64, no. 7, pp. 2383–2392, 2023.

[155] A. N. Angelopoulos and S. Bates, "A gentle introduction to conformal prediction and distribution-free uncertainty quantification," *arXiv preprint arXiv:2107.07511*, 2021.

[156] S. Kearnes and P. Riley, "Ordinal confidence level assignments for regression model predictions," *Journal of Chemical Information and Modeling*, vol. 64, no. 24, pp. 9299–9305, 2024.

[157] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[158] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning." `https://github.com/Lightning-AI/lightning`, Mar. 2019.

[159] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[160] M. M. McKerns, L. Strand, T. Sullivan, A. Fang, and M. A. Aivazis, "Building a framework for predictive science," *arXiv preprint arXiv:1202.1056*, 2012.

[161] JGraph, "draw.io." `https://github.com/jgraph/drawio`, Oct. 2021.

Appendix A

Implementation Details

A.1 Data

A.1.1 ADME Data

In Vitro Essays

All details regarding the in vitro essays can be found in the paper from *Fang et al.* [95].

hPPB and rPPB

$$\%\text{Unbound} = \frac{\text{avgPAR in the buffer side}}{\text{avgPAR in the plasma side}} \times 100 \tag{A.1}$$

RLM and HLM

$$CL_{int} = \frac{0.693}{T_{1/2}} \times \frac{\text{incubation volume}}{\text{mg of microsomal protein}} \tag{A.2}$$

MDR1-ER

$$P_{app} = (\frac{dC_r}{dt}) \times \frac{V_r}{(A \times C_E)} \tag{A.3}$$

$$\text{Mass balance} = 100 \times \frac{((V_r \times C_r^{\text{final}}) + (V_d \times C_d^{\text{final}}))}{(V_d \times C_E)} \tag{A.4}$$

Statistics of Train/Test Split

	count	mean	std	min	25%	50%	75%	max
LOG HLM activity	2475	1.33	0.63	0.68	0.68	1.22	1.82	3.37
LOG RLM activity	2449	2.28	0.75	1.03	1.71	2.35	2.86	3.97
LOG MDR1 ER activity	2112	0.37	0.68	-1.16	-0.17	0.12	0.86	2.73
LOG Sol activity	1771	1.27	0.68	-1.00	1.19	1.55	1.69	2.15
butina cluster	2813	327.40	365.16	0.00	50.00	192.00	490.00	1605.00

Table A.1: Descriptive statistics of the evaluation data (train split)

	count	mean	std	min	25%	50%	75%	max
LOG HLM activity	607	1.27	0.58	0.68	0.68	1.14	1.73	3.00
LOG RLM activity	600	2.16	0.74	1.03	1.56	2.21	2.72	3.97
LOG MDR1 ER activity	528	0.50	0.72	-0.85	-0.13	0.30	1.04	2.18
LOG Sol activity	402	1.20	0.72	-1.00	1.09	1.51	1.67	2.18
butina cluster	703	1061.58	345.61	120.00	817.00	1114.00	1289.50	1607.00

Table A.2: Descriptive statistics of the evaluation data (test split)

A.2 Hyperparameter Tuning

A.2.1 GIN

Architecture Tuning

	count	mean	std	min	25%	50%	75%	max
Val MAE	66	0.3561	0.0209	0.3335	0.3398	0.3487	0.3663	0.4107
params encoder dim	66	316.2424	110.3647	133.0000	231.2500	316.5000	405.7500	507.0000
params encoder dropout	66	0.1169	0.0700	0.0030	0.0564	0.1253	0.1736	0.2372
params encoder layer	66	3.6212	1.4333	1.0000	3.0000	4.0000	5.0000	5.0000
params encoder output	66	704.3182	191.6995	283.0000	546.5000	742.5000	837.7500	1020.0000
params predictor dim	66	417.6212	231.9407	150.0000	245.2500	345.5000	549.0000	991.0000
params predictor dropout	66	0.2115	0.1007	0.0019	0.1455	0.2277	0.2972	0.3454
params predictor layers	66	3.2879	0.7599	2.0000	3.0000	3.0000	4.0000	4.0000
Val MAE LOG HLM activity MAE	66	0.3420	0.0194	0.3204	0.3271	0.3348	0.3517	0.3950
Val MAE LOG MDR1 ER activity MAE	66	0.3325	0.0214	0.3049	0.3163	0.3249	0.3401	0.3823
Val MAE LOG RLM activity MAE	66	0.4054	0.0222	0.3780	0.3878	0.3984	0.4161	0.4636
Val MAE LOG Sol activity MAE	66	0.3445	0.0218	0.3189	0.3262	0.3374	0.3559	0.4013

Table A.3: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for GIN. The parameters are used to tune the encoder/predictor architecture.

Optimizer Tuning

	count	mean	std	min	25%	50%	75%	max
Val MAE	34	0.3545	0.0337	0.3315	0.3339	0.3403	0.3656	0.5025
params beta1	34	0.9653	0.0148	0.9503	0.9525	0.9595	0.9753	0.9960
params beta2	34	0.9731	0.0136	0.9522	0.9628	0.9720	0.9797	0.9973
params epsilon	34	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
params lr	34	0.0016	0.0021	0.0001	0.0002	0.0006	0.0020	0.0087
params wd	34	0.0007	0.0018	0.0000	0.0000	0.0001	0.0003	0.0081
Val MAE LOG HLM activity MAE	34	0.3429	0.0325	0.3166	0.3220	0.3288	0.3581	0.4750
Val MAE LOG MDR1 ER activity MAE	34	0.3323	0.0360	0.3065	0.3117	0.3209	0.3372	0.5013
Val MAE LOG RLM activity MAE	34	0.4045	0.0383	0.3729	0.3804	0.3860	0.4175	0.5623
Val MAE LOG Sol activity MAE	34	0.3381	0.0287	0.3174	0.3209	0.3260	0.3475	0.4704

Table A.4: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for GIN. The parameters are used to tune the optimizer parameters.

A.2.2 EGNN

Architecture Tuning

	count	mean	std	min	25%	50%	75%	max
Val MAE	66	0.3778	0.0387	0.3341	0.3421	0.3788	0.3935	0.5298
params encoder dim	66	271.2121	106.5597	129.0000	173.0000	268.5000	359.7500	498.0000
params encoder layer	66	3.3939	1.4558	1.0000	2.0000	3.0000	4.0000	6.0000
params encoder out	66	551.3030	224.2448	265.0000	369.0000	491.0000	731.7500	1017.0000
params predictor batchnorm	66	0.6515	0.4801	0.0000	0.0000	1.0000	1.0000	1.0000
params predictor dim	66	665.2424	270.3726	164.0000	420.0000	746.0000	891.2500	1012.0000
params predictor dropout	66	0.3143	0.1442	0.0057	0.1994	0.3469	0.4242	0.4934
params predictor layers	66	4.1212	0.9690	2.0000	4.0000	4.0000	5.0000	5.0000
Val MAE LOG HLM activity MAE	66	0.3620	0.0357	0.3196	0.3290	0.3612	0.3768	0.5021
Val MAE LOG MDR1 ER activity MAE	66	0.3606	0.0506	0.3071	0.3173	0.3568	0.3876	0.5507
Val MAE LOG RLM activity MAE	66	0.4302	0.0394	0.3844	0.3967	0.4319	0.4459	0.5916
Val MAE LOG Sol activity MAE	66	0.3583	0.0334	0.3224	0.3309	0.3545	0.3699	0.4737

Table A.5: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for EGNN. The parameters are used to tune the encoder/predictor architecture.

Optimizer Tuning

	count	mean	std	min	25%	50%	75%	max
Val MAE	34	0.3698	0.0414	0.3314	0.3378	0.3515	0.3999	0.4503
params beta1	34	0.8585	0.0586	0.8010	0.8128	0.8334	0.8977	0.9857
params beta2	34	0.9340	0.0269	0.9023	0.9128	0.9284	0.9504	0.9945
params epsilon	34	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
params lr	34	0.0004	0.0003	0.0000	0.0001	0.0002	0.0007	0.0010
params wd	34	0.0006	0.0018	0.0000	0.0000	0.0000	0.0001	0.0081
Val MAE LOG HLM activity MAE	34	0.3558	0.0424	0.3124	0.3219	0.3378	0.3847	0.4313
Val MAE LOG MDR1 ER activity MAE	34	0.3416	0.0392	0.3068	0.3134	0.3205	0.3727	0.4143
Val MAE LOG RLM activity MAE	34	0.4224	0.0443	0.3796	0.3874	0.4079	0.4522	0.5213
Val MAE LOG Sol activity MAE	34	0.3592	0.0403	0.3179	0.3294	0.3419	0.3857	0.4341

Table A.6: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for EGNN. The parameters are used to tune the optimizer parameters.

A.2.3 LSTM

Architecture Tuning

	count	mean	std	min	25%	50%	75%	max
Val MAE	66	0.3853	0.0313	0.3482	0.3667	0.3768	0.3922	0.4959
params encoder dim	66	394.1818	104.2841	160.0000	340.0000	432.0000	480.0000	512.0000
params encoder dropout	66	0.3303	0.1391	0.0054	0.2465	0.3523	0.4585	0.4960
params encoder layer	66	2.2576	0.7905	1.0000	2.0000	2.0000	3.0000	3.0000
params encoder output	66	618.7121	197.4318	259.0000	476.7500	571.5000	751.2500	995.0000
params n heads	66	8.2121	5.7660	2.0000	2.5000	8.0000	16.0000	16.0000
params predictor batchnorm	66	0.1818	0.3887	0.0000	0.0000	0.0000	0.0000	1.0000
params predictor dim	66	550.5758	200.4236	164.0000	405.0000	554.5000	675.2500	940.0000
params predictor dropout	66	0.2678	0.1350	0.0072	0.1649	0.2604	0.3848	0.4928
params predictor layers	66	3.9848	1.0596	2.0000	3.0000	4.0000	5.0000	5.0000
Val MAE LOG HLM activity MAE	66	0.3929	0.0320	0.3618	0.3743	0.3822	0.3953	0.5091
Val MAE LOG MDR1 ER activity MAE	66	0.4110	0.0405	0.3656	0.3818	0.4013	0.4194	0.5562
Val MAE LOG RLM activity MAE	66	0.4724	0.0332	0.4393	0.4536	0.4624	0.4742	0.5975
Val MAE LOG Sol activity MAE	66	0.3854	0.0313	0.3482	0.3669	0.3769	0.3924	0.4957

Table A.7: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for LSTM. The parameters are used to tune the encoder/predictor architecture.

Optimizer Tuning

	count	mean	std	min	25%	50%	75%	max
Val MAE	34	0.3794	0.0213	0.3457	0.3627	0.3707	0.3913	0.4301
params beta1	34	0.8538	0.0589	0.8010	0.8099	0.8252	0.8903	0.9857
params beta2	34	0.9334	0.0293	0.9005	0.9073	0.9240	0.9571	0.9945
params epsilon	34	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
params lr	34	0.0004	0.0003	0.0000	0.0001	0.0002	0.0007	0.0010
params wd	34	0.0008	0.0019	0.0000	0.0000	0.0001	0.0002	0.0081
Val MAE LOG HLM activity MAE	34	0.3961	0.0366	0.3612	0.3707	0.3820	0.4003	0.4862
Val MAE LOG MDR1 ER activity MAE	34	0.4078	0.0406	0.3572	0.3846	0.3948	0.4143	0.5225
Val MAE LOG RLM activity MAE	34	0.4761	0.0335	0.4399	0.4498	0.4681	0.4857	0.5502
Val MAE LOG Sol activity MAE	34	0.3794	0.0213	0.3461	0.3625	0.3706	0.3913	0.4304

Table A.8: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for LSTM. The parameters are used to tune the optimizer parameters.

A.2.4 XGB

RLM

	count	mean	std	min	25%	50%	75%	max
Val MAE	100	0.4540	0.0333	0.4302	0.4365	0.4404	0.4507	0.6107
params colsample bytree	100	0.8303	0.1132	0.6082	0.7416	0.8513	0.9278	0.9981
params learning rate	100	0.0267	0.0246	0.0011	0.0069	0.0201	0.0377	0.0939
params max depth	100	6.1000	3.3530	2.0000	4.0000	5.0000	7.2500	15.0000
params min child weight	100	10.7900	5.6627	1.0000	6.0000	11.0000	16.0000	20.0000
params n estimators	100	932.0000	357.4969	150.0000	687.5000	1000.0000	1250.0000	1500.0000
params reg alpha	100	1.7746	3.9859	0.0012	0.0064	0.0765	1.2375	19.5767
params reg lambda	100	1.6164	3.5500	0.0011	0.0059	0.0355	0.5106	17.2959
params subsample	100	0.7829	0.1052	0.6066	0.7049	0.7603	0.8763	0.9887

Table A.9: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for XGB for endpoint RLM

HLM

	count	mean	std	min	25%	50%	75%	max
Val MAE	100	0.3891	0.0301	0.3663	0.3727	0.3769	0.3885	0.5227
params colsample bytree	100	0.8015	0.1101	0.6082	0.7296	0.8079	0.8778	0.9993
params learning rate	100	0.0303	0.0268	0.0011	0.0059	0.0201	0.0464	0.0976
params max depth	100	6.4500	3.2702	2.0000	4.0000	6.0000	8.0000	15.0000
params min child weight	100	12.4100	5.4848	1.0000	8.7500	14.0000	17.0000	20.0000
params n estimators	100	1018.5000	410.0169	100.0000	700.0000	1150.0000	1362.5000	1500.0000
params reg alpha	100	1.6143	3.9441	0.0010	0.0064	0.0855	0.6952	19.5767
params reg lambda	100	1.6840	3.6554	0.0010	0.0042	0.0214	0.4967	17.2959
params subsample	100	0.7816	0.1064	0.6048	0.6958	0.7976	0.8612	0.9887

Table A.10: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for XGB for endpoint HLM

MDR1-ER

	count	mean	std	min	25%	50%	75%	max
Val MAE	100	0.3782	0.0365	0.3508	0.3596	0.3638	0.3739	0.5462
params colsample bytree	100	0.8016	0.1133	0.6045	0.7134	0.8042	0.8853	0.9943
params learning rate	100	0.0296	0.0262	0.0011	0.0060	0.0214	0.0447	0.0884
params max depth	100	5.8400	3.4749	2.0000	3.0000	5.0000	7.2500	15.0000
params min child weight	100	7.3800	5.9996	1.0000	2.0000	5.0000	12.0000	20.0000
params n estimators	100	954.5000	404.4010	100.0000	600.0000	1000.0000	1312.5000	1500.0000
params reg alpha	100	2.8534	5.2419	0.0010	0.0124	0.3665	2.1874	24.0251
params reg lambda	100	2.3730	4.6625	0.0011	0.0042	0.0438	1.8905	19.5485
params subsample	100	0.8305	0.1157	0.6066	0.7334	0.8412	0.9410	0.9935

Table A.11: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for XGB for endpoint MDR1-ER

Sol

	count	mean	std	min	25%	50%	75%	max
Val MAE	100	0.3921	0.0196	0.3777	0.3811	0.3848	0.3925	0.4885
params colsample bytree	100	0.7536	0.1199	0.6009	0.6428	0.7387	0.8534	0.9943
params learning rate	100	0.0155	0.0184	0.0011	0.0042	0.0082	0.0191	0.0841
params max depth	100	8.5200	3.6639	2.0000	5.0000	9.0000	11.0000	15.0000
params min child weight	100	12.7800	5.9571	1.0000	8.7500	14.0000	18.0000	20.0000
params n estimators	100	984.0000	398.7911	150.0000	637.5000	1050.0000	1350.0000	1500.0000
params reg alpha	100	1.6692	3.9260	0.0010	0.0056	0.1368	1.1306	19.5767
params reg lambda	100	2.3037	4.3608	0.0011	0.0136	0.0803	2.1761	20.5340
params subsample	100	0.7474	0.1137	0.6034	0.6426	0.7185	0.8407	0.9887

Table A.12: Descriptive statistics of the butina based 5-fold cross validation hyperparameter tuning for XGB for endpoint Sol

A.3 Resulting Architecture

A.3.1 GIN

	Value
params encoder activation	relu
params encoder dim	243
params encoder dropout	0.0759
params encoder layer	5
params encoder output	563
params predictor activation	leaky
params predictor dim	264
params predictor dropout	0.3388
params predictor layers	4
Val MAE	0.3315
params beta1	0.9504
params beta2	0.9555
params epsilon	0.0000
params lr	0.0004
params wd	0.0000
Val MAE LOG HLM activity MAE	0.3188
Val MAE LOG MDR1 ER activity MAE	0.3077
Val MAE LOG RLM activity MAE	0.3795
Val MAE LOG Sol activity MAE	0.3198

Table A.13: Resulting encoder/predictor/optimizer parameters for GIN.

A.3.2 EGNN

	Value
params encoder activation	leaky
params encoder aggregation	sum
params encoder dim	248
params encoder layer	4
params encoder out	436
params encoder pooling	add
params predictor activation	silu
params predictor batchnorm	1
params predictor dim	965
params predictor dropout	0.3203
params predictor layers	4
Val MAE	0.3314
params beta1	0.8127
params beta2	0.9214
params epsilon	0.0000
params lr	0.0010
params wd	0.0000
Val MAE LOG HLM activity MAE	0.3166
Val MAE LOG MDR1 ER activity MAE	0.3068
Val MAE LOG RLM activity MAE	0.3813
Val MAE LOG Sol activity MAE	0.3201

Table A.14: Resulting encoder/predictor/optimizer parameters for EGNN.

A.3.3 LSTM

	Value
params encoder dim	384
params encoder dropout	0.3805
params encoder layer	3
params encoder output	548
params n heads	8
params predictor activation	leaky
params predictor batchnorm	0
params predictor dim	608
params predictor dropout	0.4598
params predictor layers	4
Val MAE	0.3457
params beta1	0.8132
params beta2	0.9032
params epsilon	0.0000
params lr	0.0007
params wd	0.0000
Val MAE LOG HLM activity MAE	0.3612
Val MAE LOG MDR1 ER activity MAE	0.3572
Val MAE LOG RLM activity MAE	0.4455
Val MAE LOG Sol activity MAE	0.3461

Table A.15: Resulting encoder/predictor/optimizer parameters for LSTM.

A.3.4 XGB

RLM

	Value
Val MAE	0.4302
params colsample bytree	0.7551
params learning rate	0.0186
params max depth	5.0000
params min child weight	15.0000
params n estimators	1250.0000
params reg alpha	0.0040
params reg lambda	0.0259
params subsample	0.6960

Table A.16: Resulting XGB architecture for endpoint RLM.

HLM

	Value
Val MAE	0.3663
params colsample bytree	0.7703
params learning rate	0.0375
params max depth	5.0000
params min child weight	13.0000
params n estimators	1350.0000
params reg alpha	0.0046
params reg lambda	0.0198
params subsample	0.7810

Table A.17: Resulting XGB architecture for endpoint HLM.

MDR1-ER

	Value
Val MAE	0.3508
params colsample bytree	0.8340
params learning rate	0.0168
params max depth	4.0000
params min child weight	3.0000
params n estimators	1500.0000
params reg alpha	0.0010
params reg lambda	0.0024
params subsample	0.9466

Table A.18: Resulting XGB architecture for endpoint MDR1-ER.

Sol

	Value
Val MAE	0.3777
params colsample bytree	0.6081
params learning rate	0.0080
params max depth	11.0000
params min child weight	20.0000
params n estimators	1250.0000
params reg alpha	0.0032
params reg lambda	0.0057
params subsample	0.6236

Table A.19: Resulting XGB architecture for endpoint Sol.

A.4 Used Hardware and Software

A.4.1 Hardware

Mltiple Intel Xeon Platinum Processors were used for cpu related tasks while multiple NVIDIA A100 GPUs with each 80 GB VRAM were used for the neural network related tasks.

A.4.2 Software

The xgboost python package version 3.0.0 was used for the XGB implementation. Graph featurizers were implemented using the Chemprop [26] package version 2.1.2. The neural networks were implemented with PyTorch [157] version 2.6.0, PyTorch lightning [158] version 2.5.1, and PyTorch geometric [159] version 2.6.1. Compound processing related tasks like descriptor calculation, butina clustering, and conformor creation are utilized by the RDKit [104] version 2024.9.6. Optuna [113] version 4.2.1 was used for hyperparameter optimization. The graph and conformer features are retrived by using multiprocessing with the pathos [160] python package version 0.3.3. All self-made Figures are made by using the software "draw.io" [161] (also knwon as "diagrams.net").

Appendix B

Further Results

B.1 Effect of Pre-training

B.1.1 Aggregated

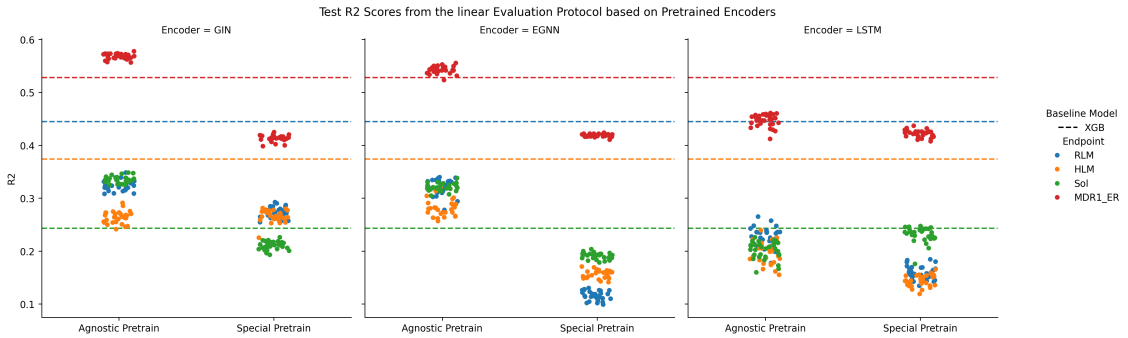

Figure B.1: Test R2 scores resulting from linear evaluation protocol based on different pre-training strategies for different ADME endpoints.

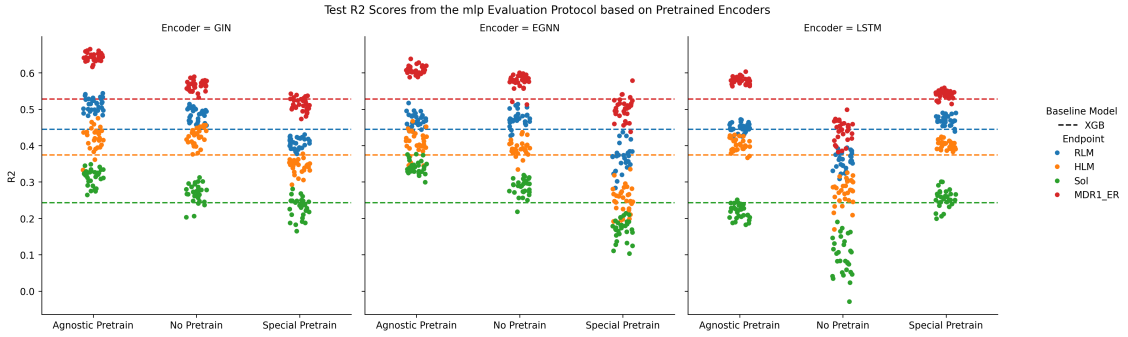

Figure B.2: Test R2 scores resulting from transfer learning evaluation protocol based on different pre-training strategies for different ADME endpoints.

B.1.2 GIN

Linear

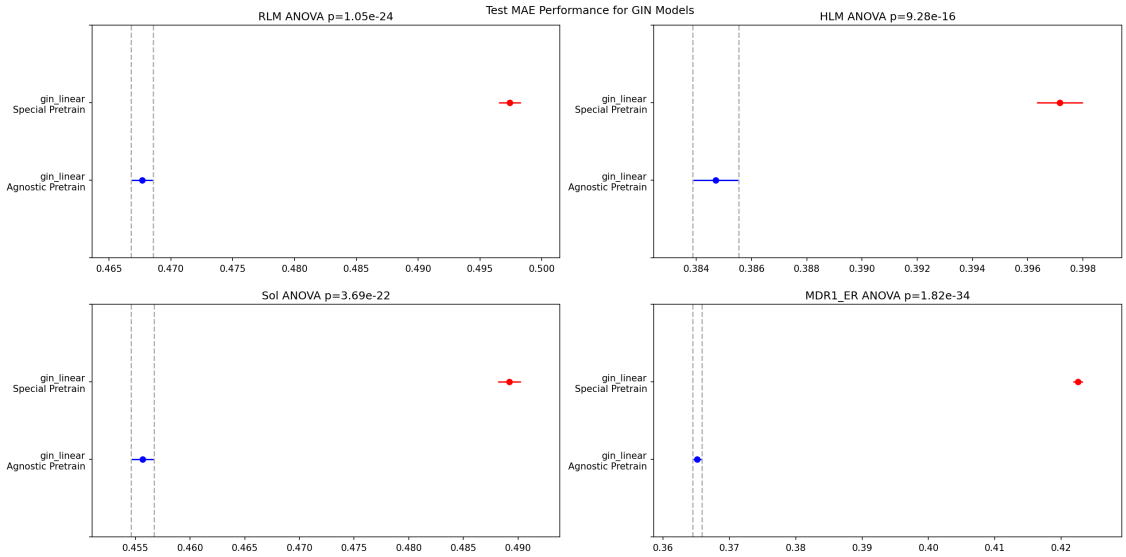

Figure B.3: Tukey plot to assess significant differences between pre-training strategies for the linear evaluation protocol regarding the test MAE.

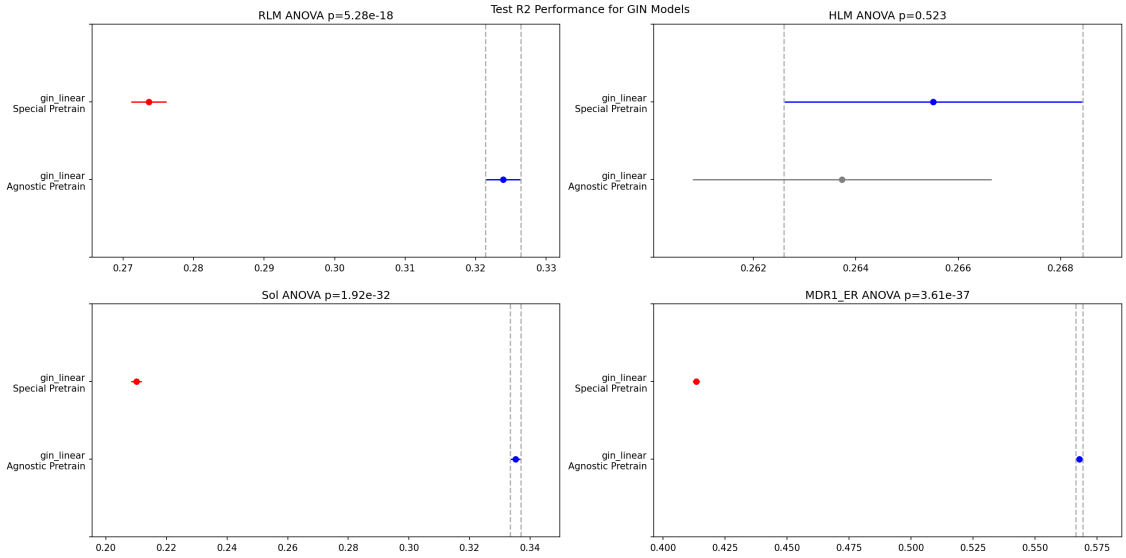

Figure B.4: Tukey plot to assess significant differences between pre-training strategies for the linear evaluation protocol regarding the test R2.

Transfer Learning

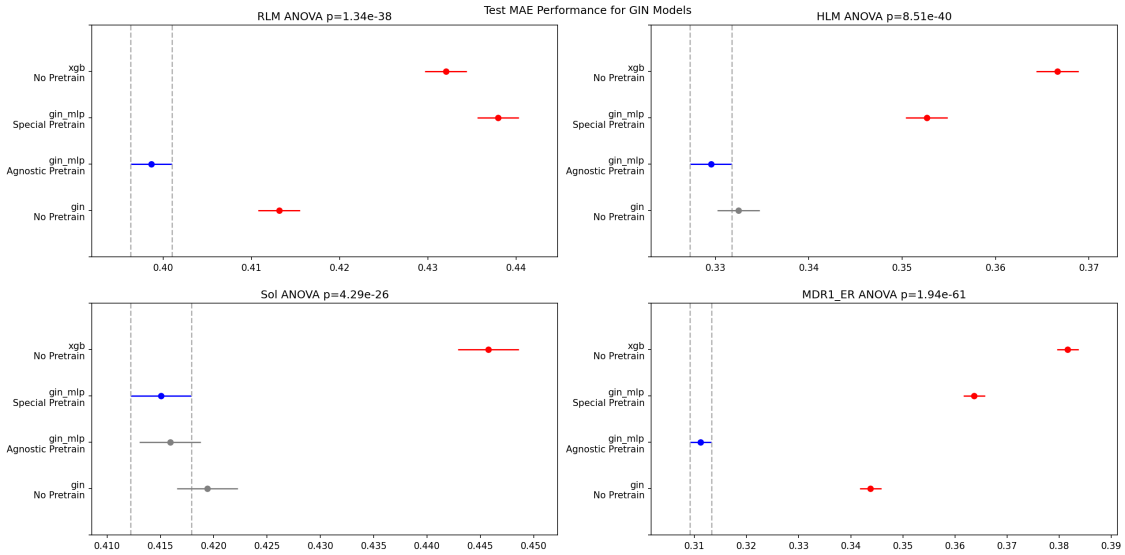

Figure B.5: Tukey plot to assess significant differences between pre-training strategies for the transfer learning evaluation protocol regarding the test MAE.

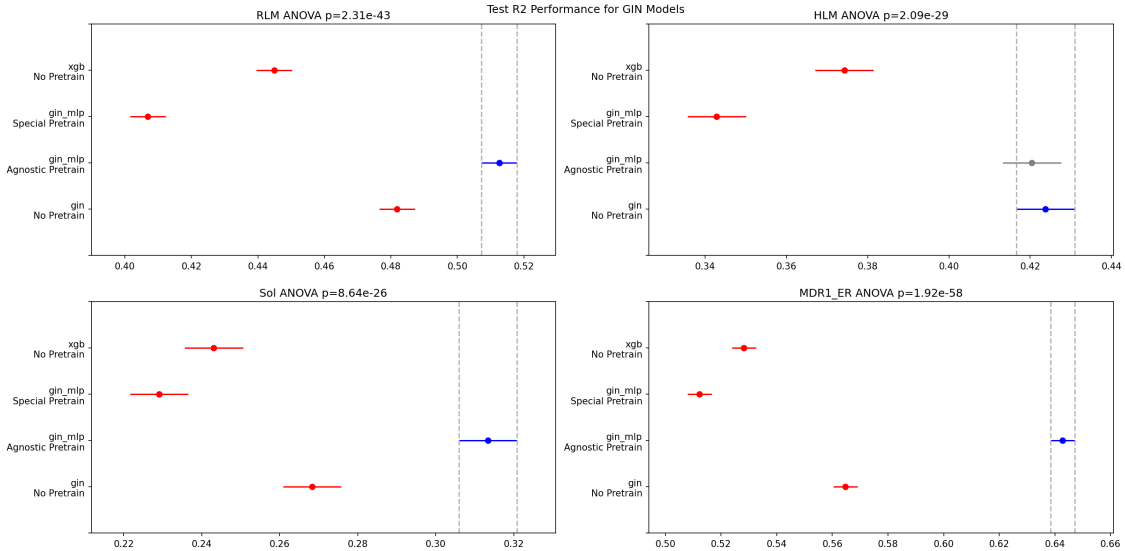

Figure B.6: Tukey plot to assess significant differences between pre-training strategies for the transfer learning evaluation protocol regarding the test R2.

B.1.3 EGNN

Linear

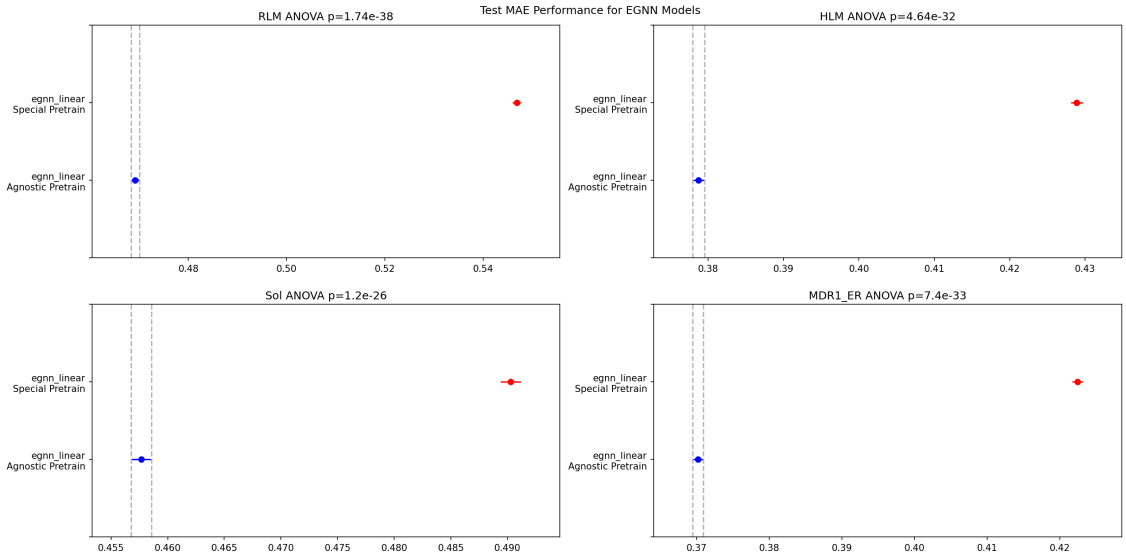

Figure B.7: Tukey plot to assess significant differences between pre-training strategies for the linear evaluation protocol regarding the test MAE.

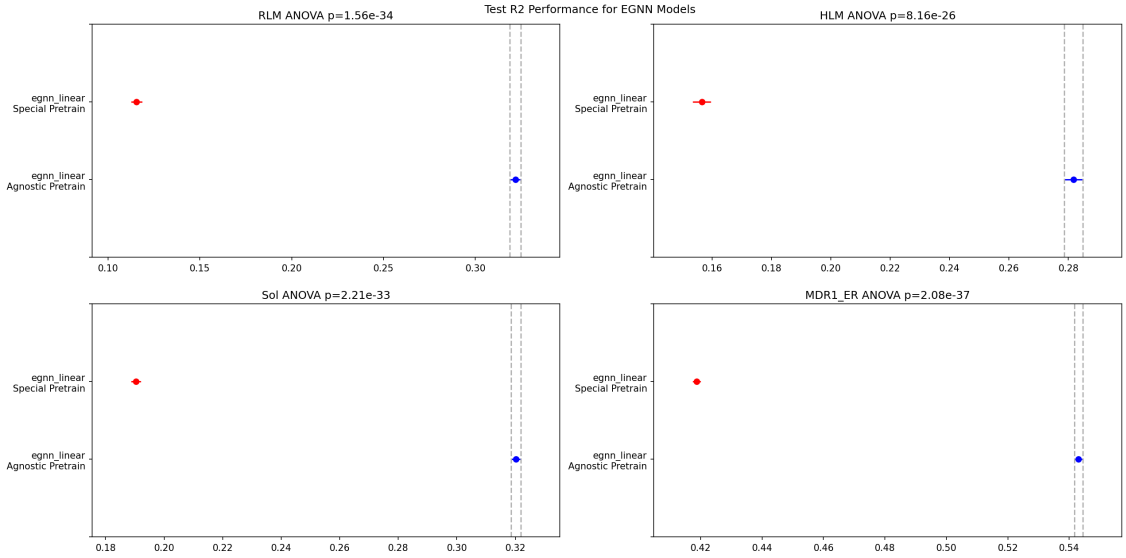

Figure B.8: Tukey plot to assess significant differences between pre-training strategies for the linear evaluation protocol regarding the test R2.

Transfer Learning

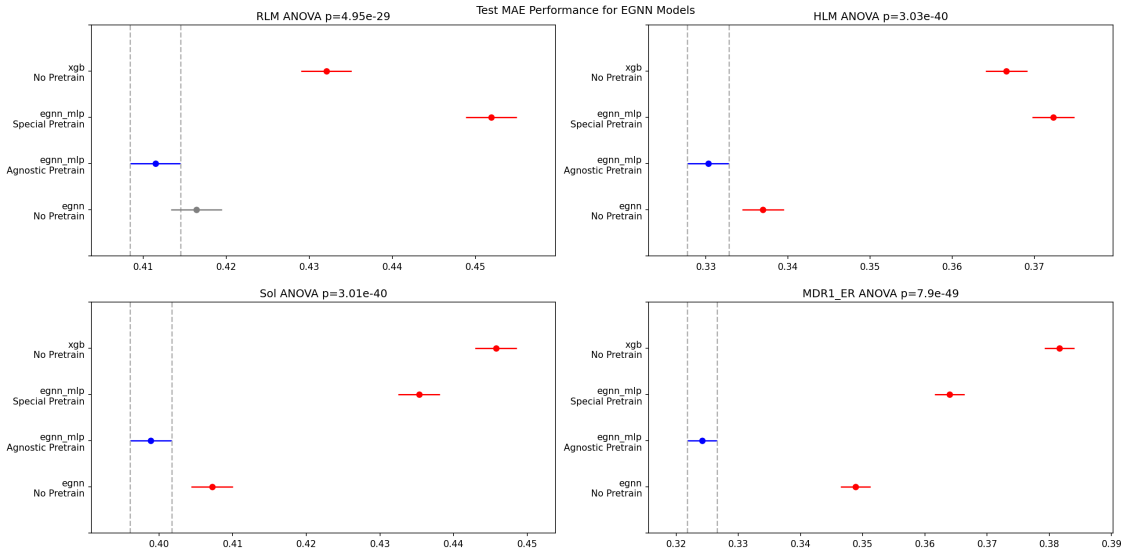

Figure B.9: Tukey plot to assess significant differences between pre-training strategies for the transfer learning evaluation protocol regarding the test MAE.

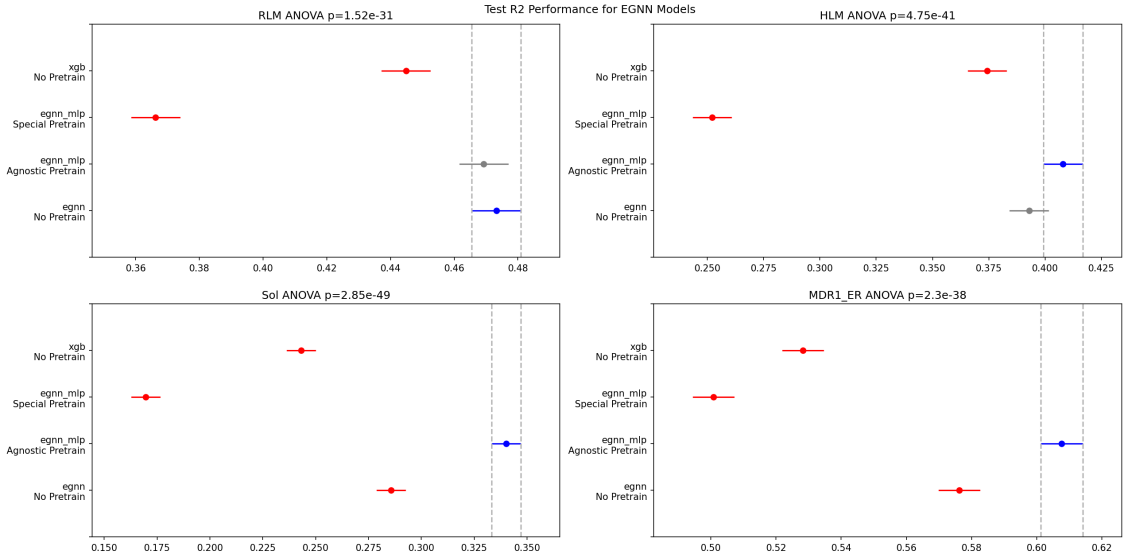

Figure B.10: Tukey plot to assess significant differences between pre-training strategies for the transfer learning evaluation protocol regarding the test R2.

B.1.4 LSTM

Linear

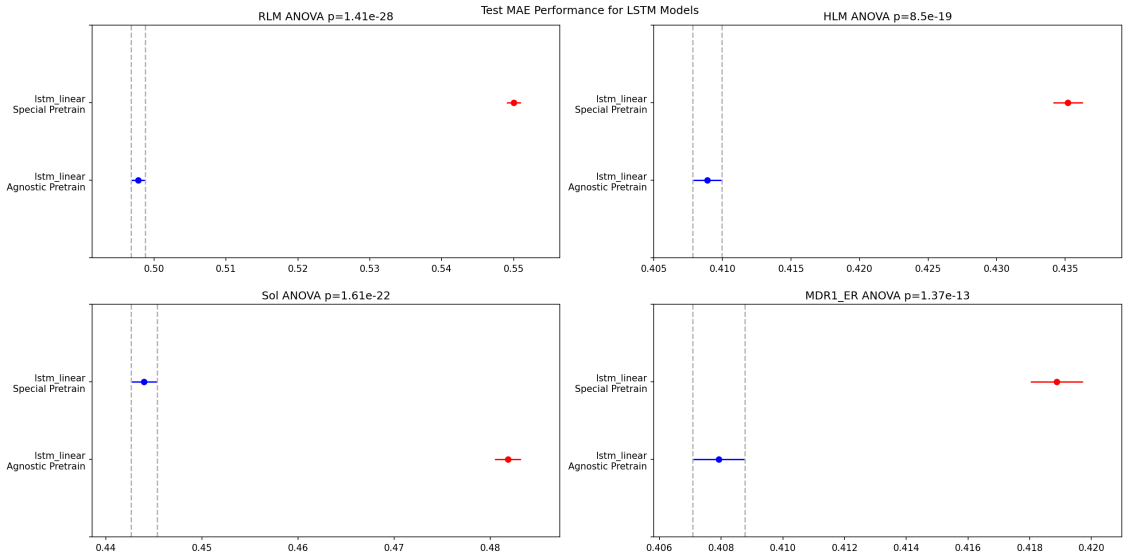

Figure B.11: Tukey plot to assess significant differences between pre-training strategies for the linear evaluation protocol regarding the test MAE.

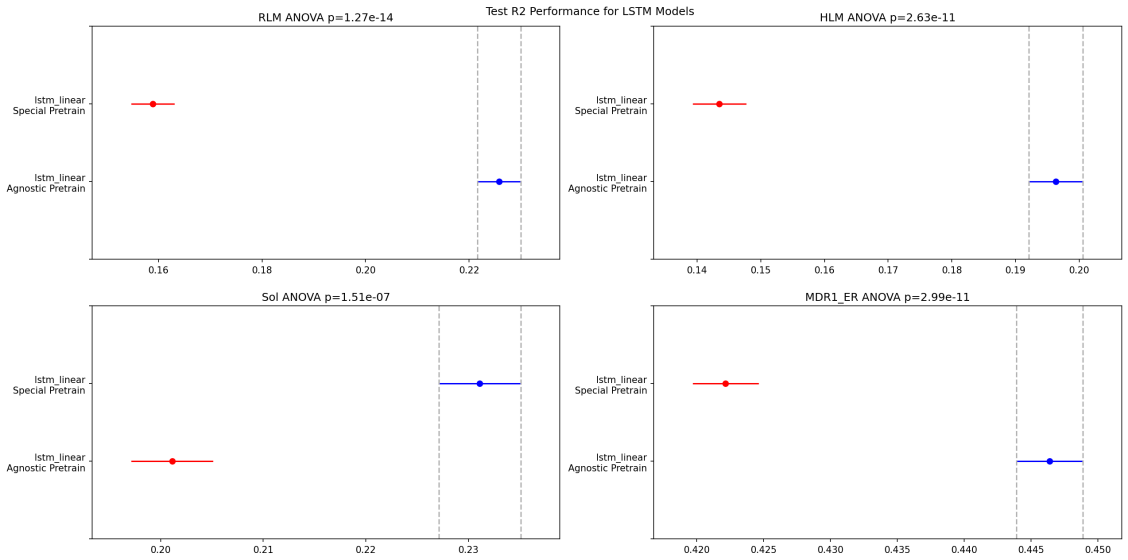

Figure B.12: Tukey plot to assess significant differences between pre-training strategies for the linear evaluation protocol regarding the test R2.

Transfer Learning

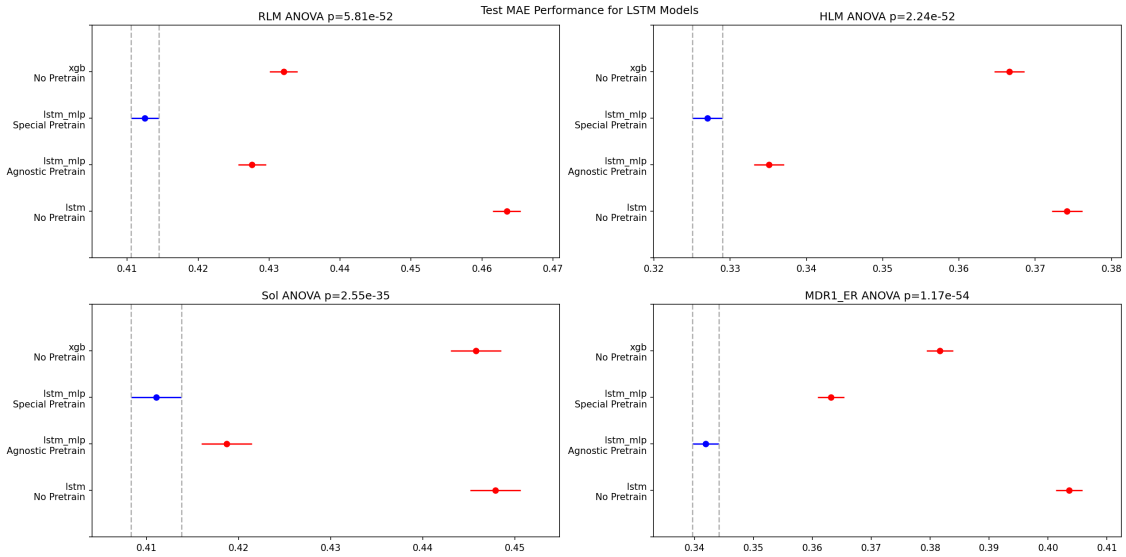

Figure B.13: Tukey plot to assess significant differences between pre-training strategies for the transfer learning evaluation protocol regarding the test MAE.

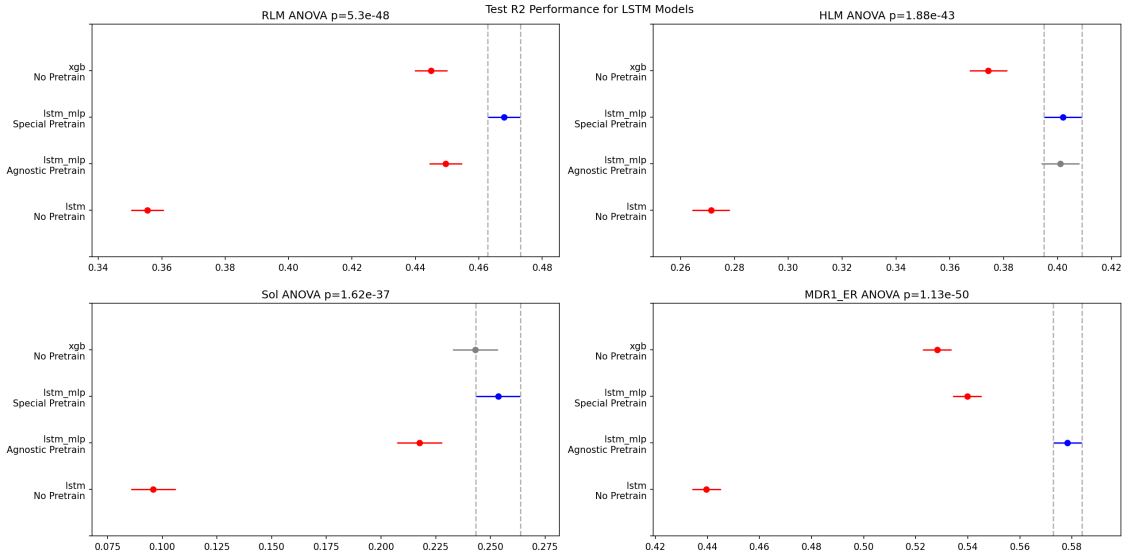

Figure B.14: Tukey plot to assess significant differences between pre-training strategies for the transfer learning evaluation protocol regarding the test R2.

B.2 Different VICReg Strategies

B.2.1 No Pre-training vs Pre-training

Aggregated

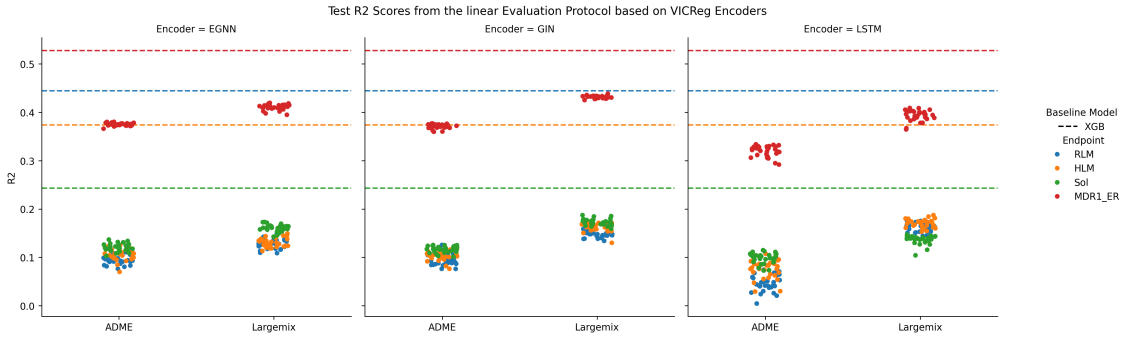

Figure B.15: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the linear evaluation protocol. The label "ADME" refers to a pre-training on the ADME training split, while "Largemix" refers to the pre-training on the largemix dataset. The y-scale refers to the test R2 coming from an evaluation of the ADME dataset.

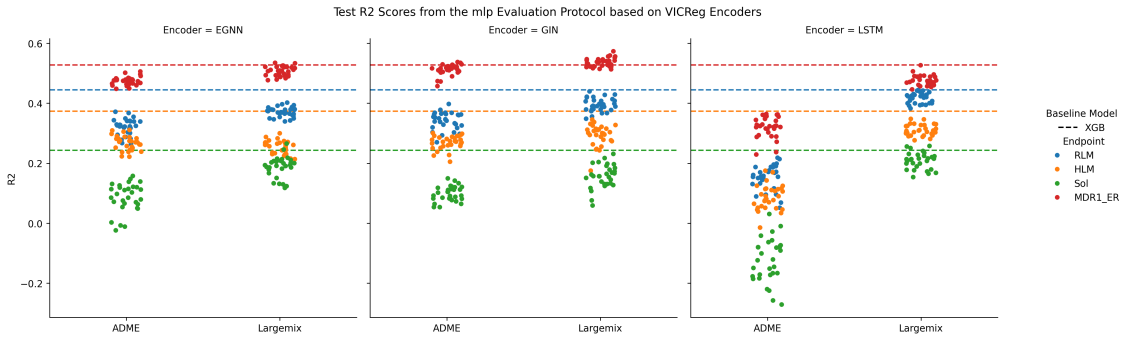

Figure B.16: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the transfer learning evaluation protocol. The label "ADME" refers to a pre-training on the ADME training split, while "Largemix" refers to the pre-training on the largemix dataset. The y-scale refers to the test R2 coming from an evaluation of the ADME dataset.

GIN

Linear Evaluation

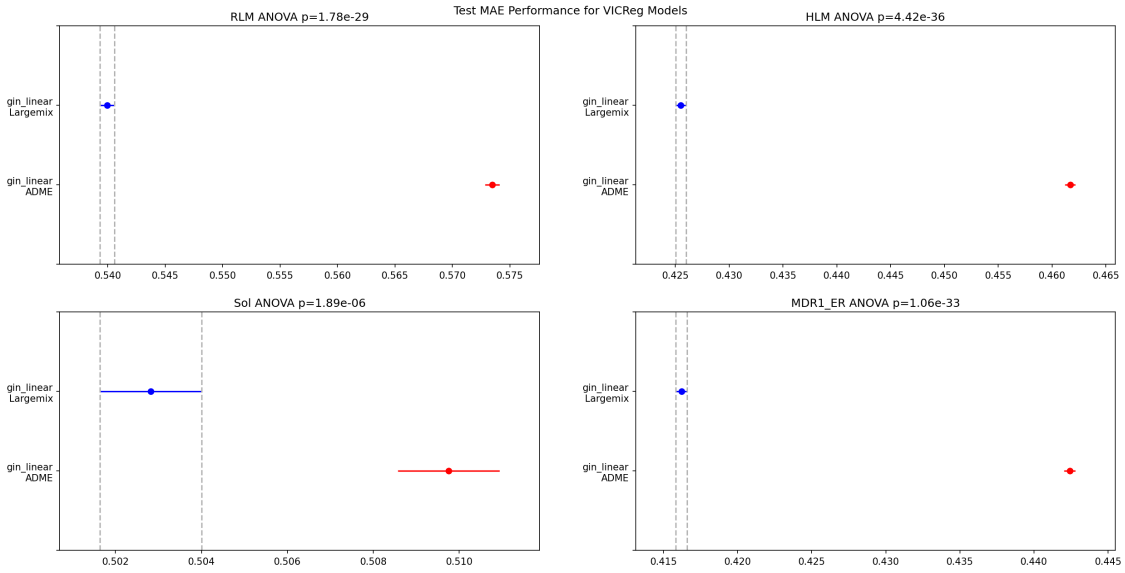

Figure B.17: Tukey plot to show significant test MAE differences on the ADME dataset among GIN encoders depending on the VICReg strategy regarding linear evaluation.

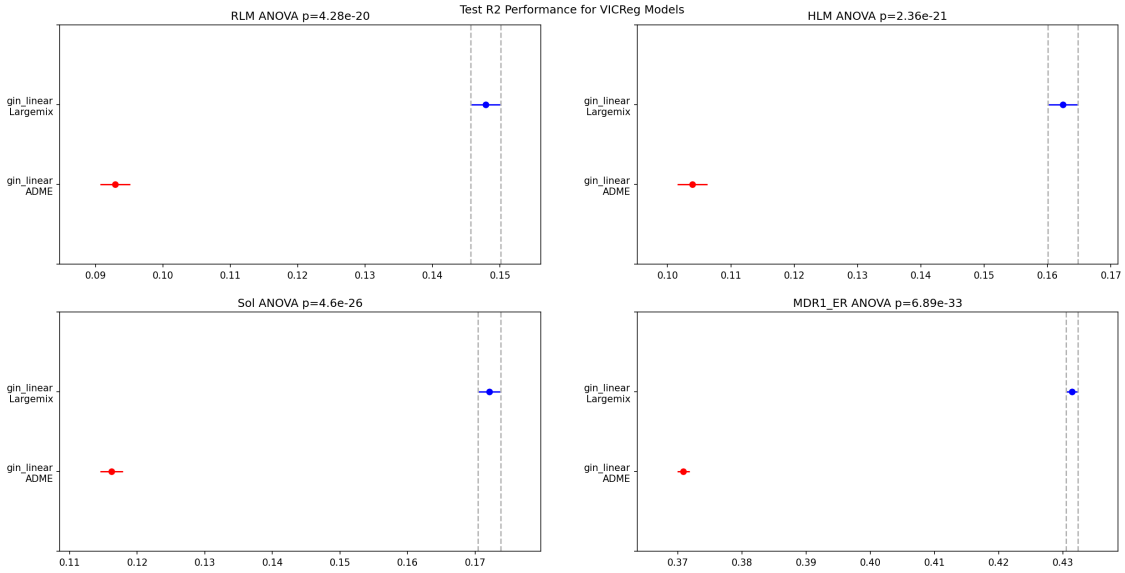

Figure B.18: Tukey plot to show significant test R2 differences on the ADME dataset among GIN encoders depending on the VICReg strategy regarding linear evaluation.

Transfer Learning

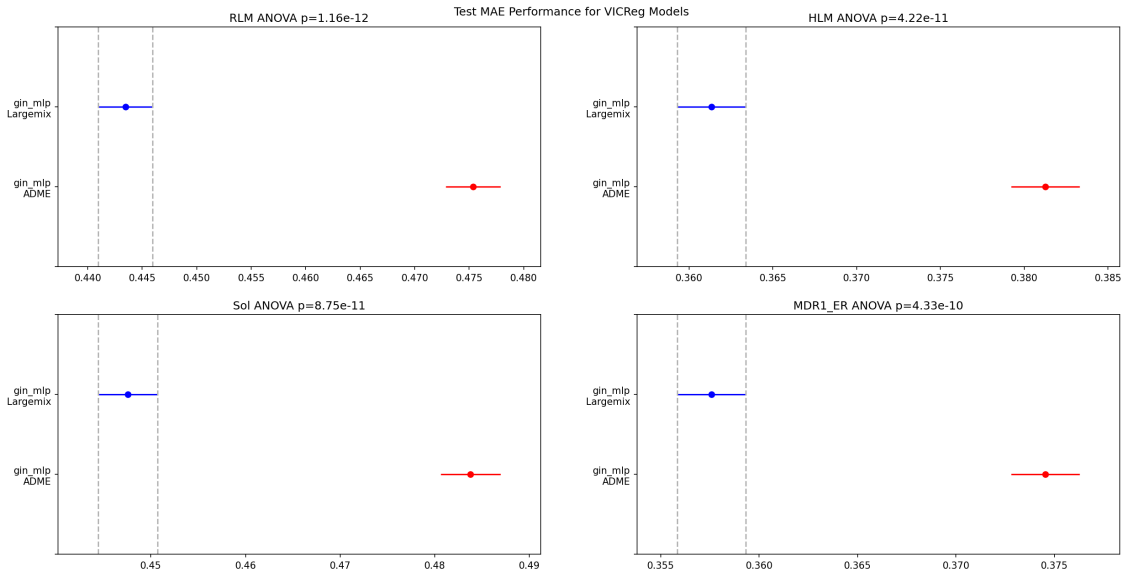

Figure B.19: Tukey plot to show significant test MAE differences on the ADME dataset among GIN encoders depending on the VICReg strategy regarding transfer learning.

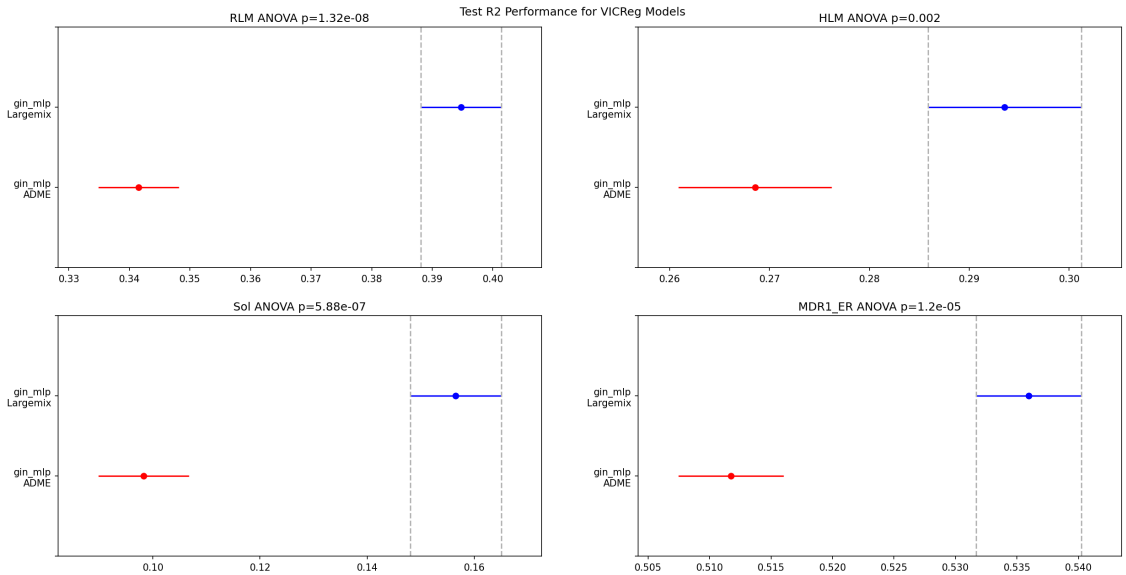

Figure B.20: Tukey plot to show significant test R2 differences on the ADME dataset among GIN encoders depending on the VICReg strategy regarding transfer learning.

EGNN

Linear Evaluation

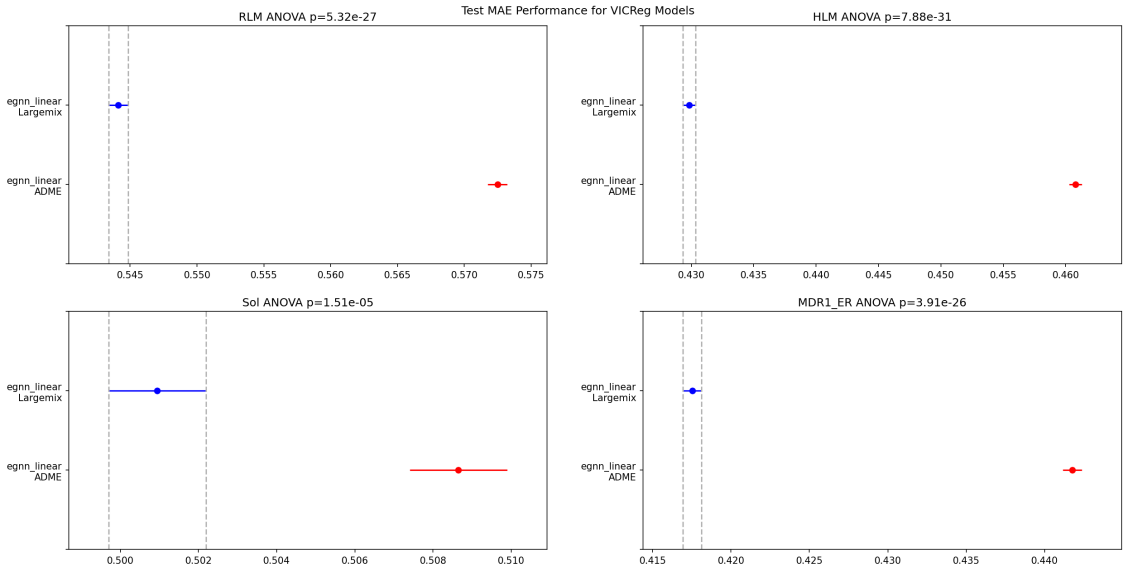

Figure B.21: Tukey plot to show significant test MAE differences on the ADME dataset among EGNN encoders depending on the VICReg strategy regarding linear evaluation.

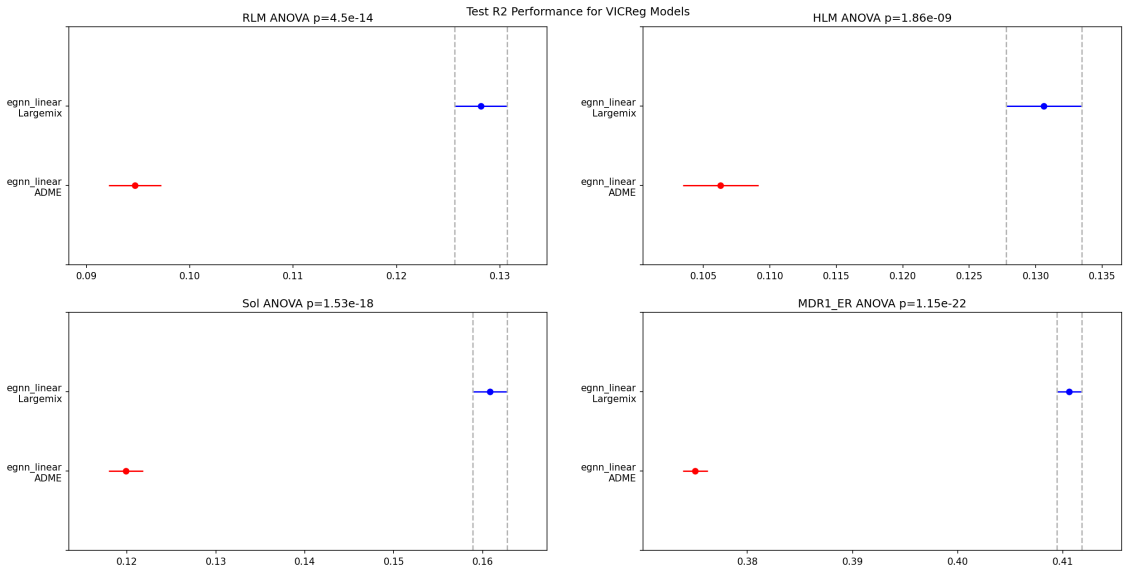

Figure B.22: Tukey plot to show significant test R2 differences on the ADME dataset among EGNN encoders depending on the VICReg strategy regarding linear evaluation.

Transfer Learning

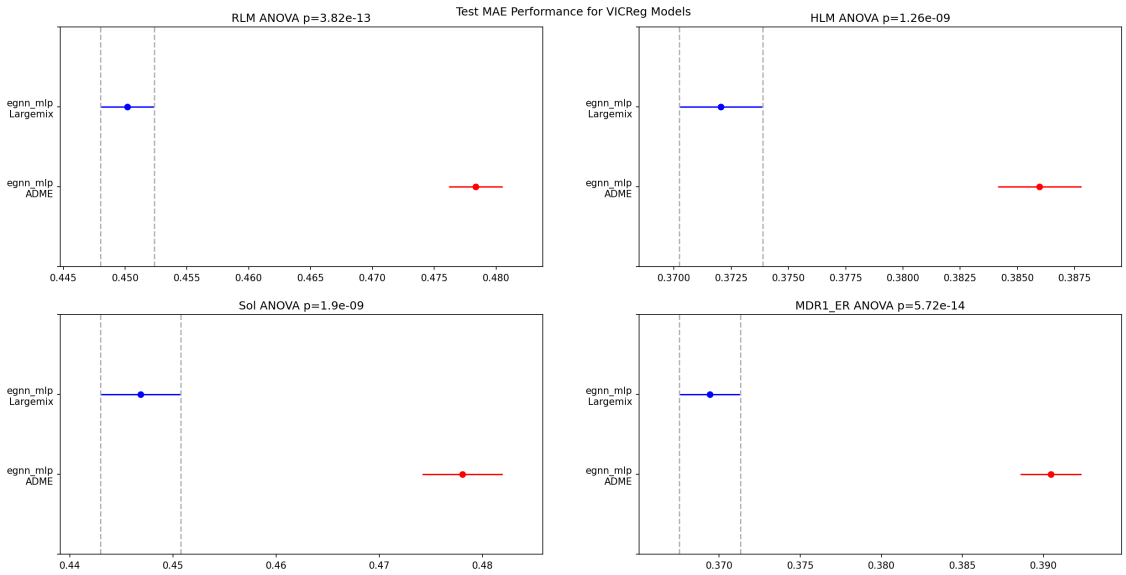

Figure B.23: Tukey plot to show significant test MAE differences on the ADME dataset among EGNN encoders depending on the VICReg strategy regarding transfer learning.

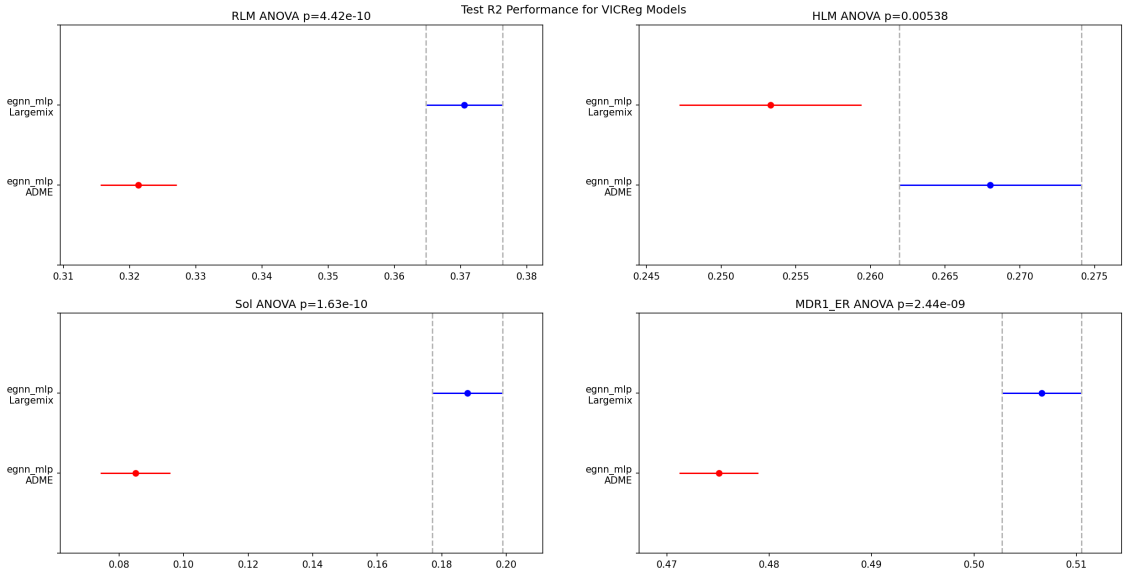

Figure B.24: Tukey plot to show significant test R2 differences on the ADME dataset among EGNN encoders depending on the VICReg strategy regarding transfer learning.

LSTM

Linear Evaluation

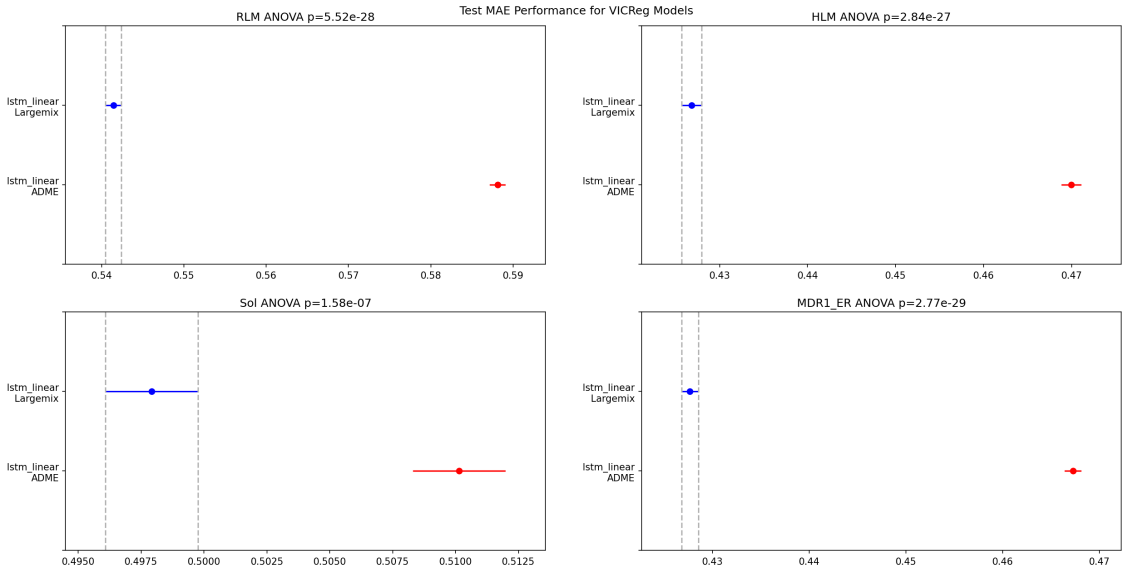

Figure B.25: Tukey plot to show significant test MAE differences on the ADME dataset among LSTM encoders depending on the VICReg strategy regarding linear evaluation.

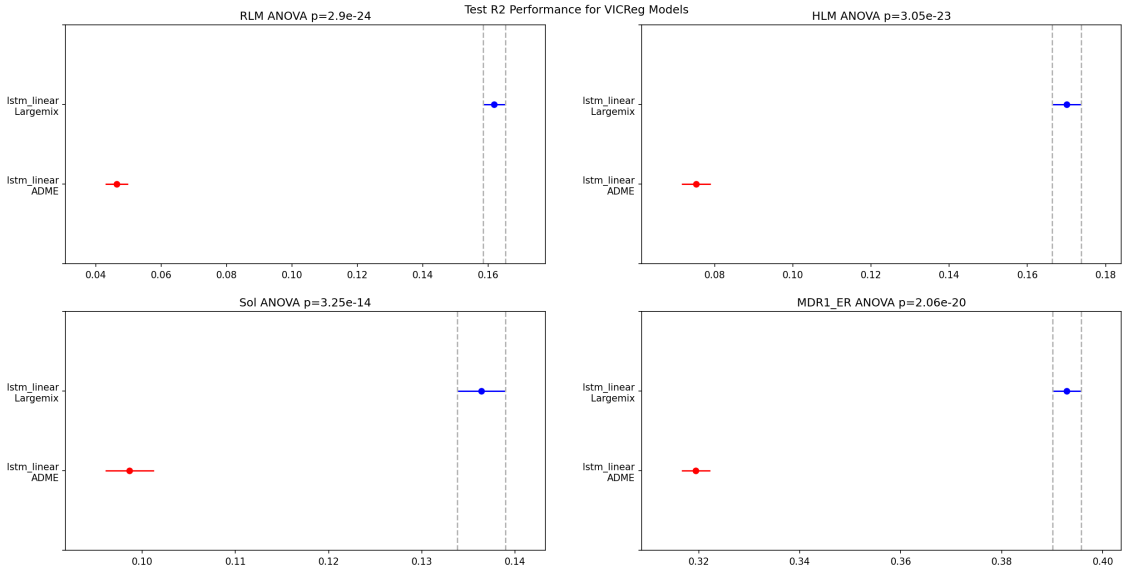

Figure B.26: Tukey plot to show significant test R2 differences on the ADME dataset among LSTM encoders depending on the VICReg strategy regarding linear evaluation.

Transfer Learning

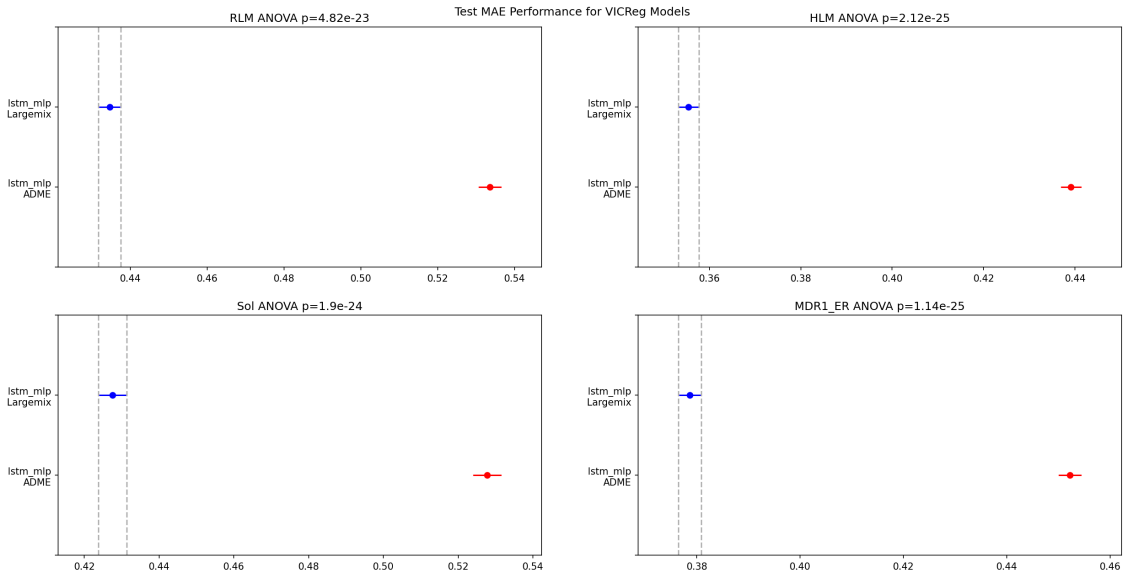

Figure B.27: Tukey plot to show significant test MAE differences on the ADME dataset among LSTM encoders depending on the VICReg strategy regarding transfer learning.

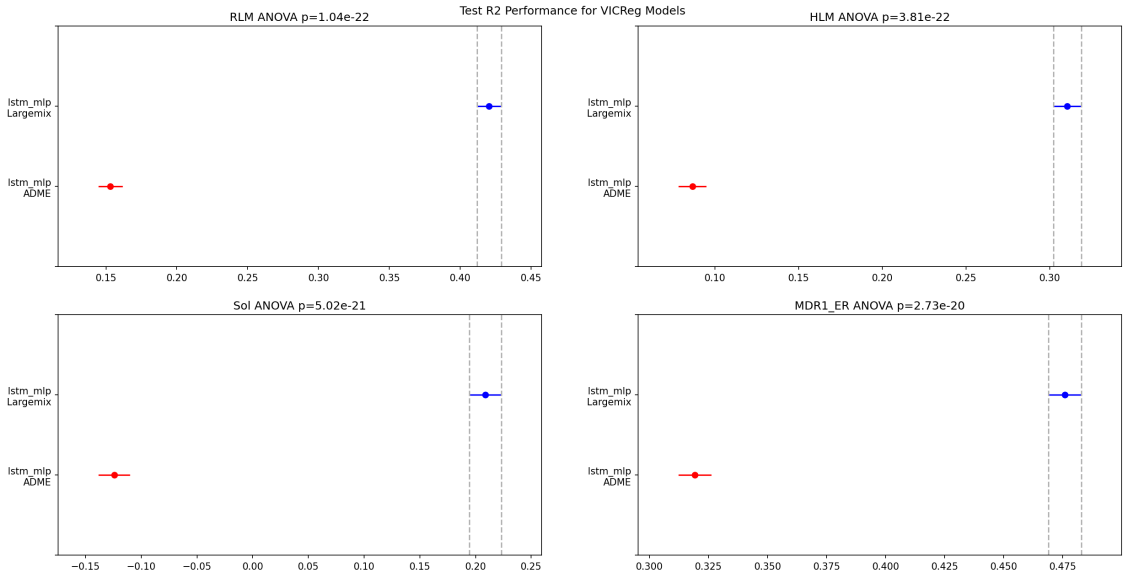

Figure B.28: Tukey plot to show significant test R2 differences on the ADME dataset among LSTM encoders depending on the VICReg strategy regarding transfer learning.

B.2.2 Agnostic Pre-training vs Specific Pre-training

Aggregated

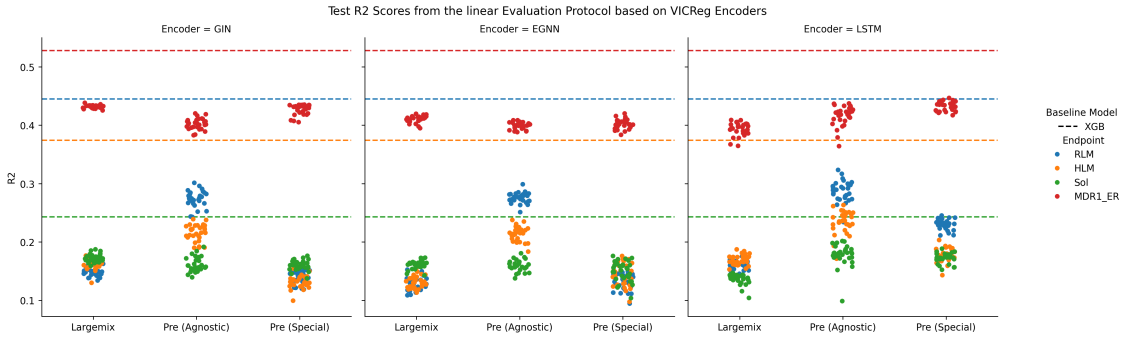

Figure B.29: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the linear evaluation protocol. Largemix refers to the VICReg training on the largmix dataset, Pre (Agnostic) refers to the use agnostic pre-trained encoder for the VICReg training, while Pre (Special) shows the results of VICReg with modalitiy specific pre-trained encoders. The y-scale refers to the test R2 coming from an evaluation of the ADME dataset.

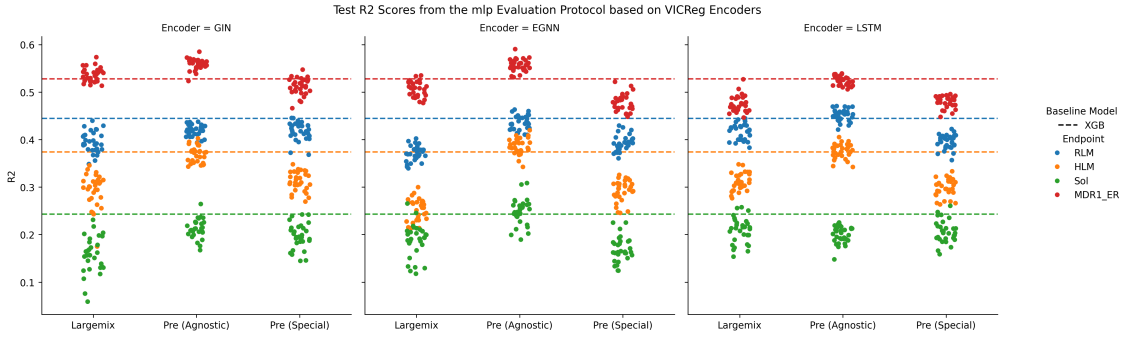

Figure B.30: Aggregated VICReg results for each encoder on each endpoint (color). Each point refers to a different random seed for the transfer learning evaluation protocol. Largemix refers to the VICReg training on the largmix dataset, Pre (Agnostic) refers to the use agnostic pre-trained encoder for the VICReg training, while Pre (Special) shows the results of VICReg with modalitiy specific pre-trained encoders. The y-scale refers to the test R2 coming from an evaluation of the ADME dataset.

GIN

Linear Evaluation

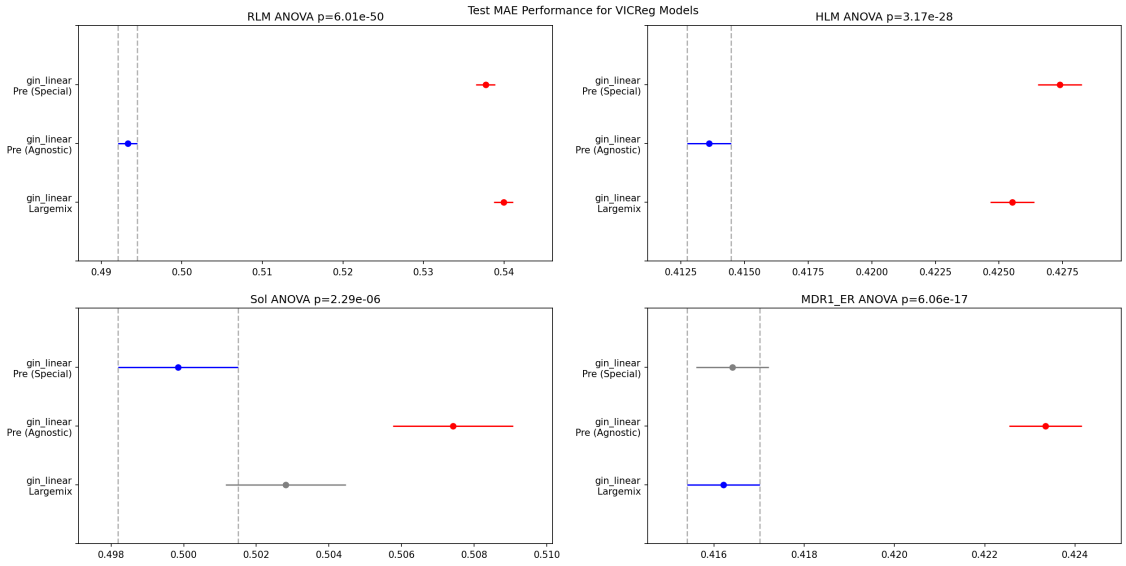

Figure B.31: Tukey plot to show significant test MAE differences on the ADME dataset among GIN encoders depending on the VICReg initializing strategy regarding linear evaluation.

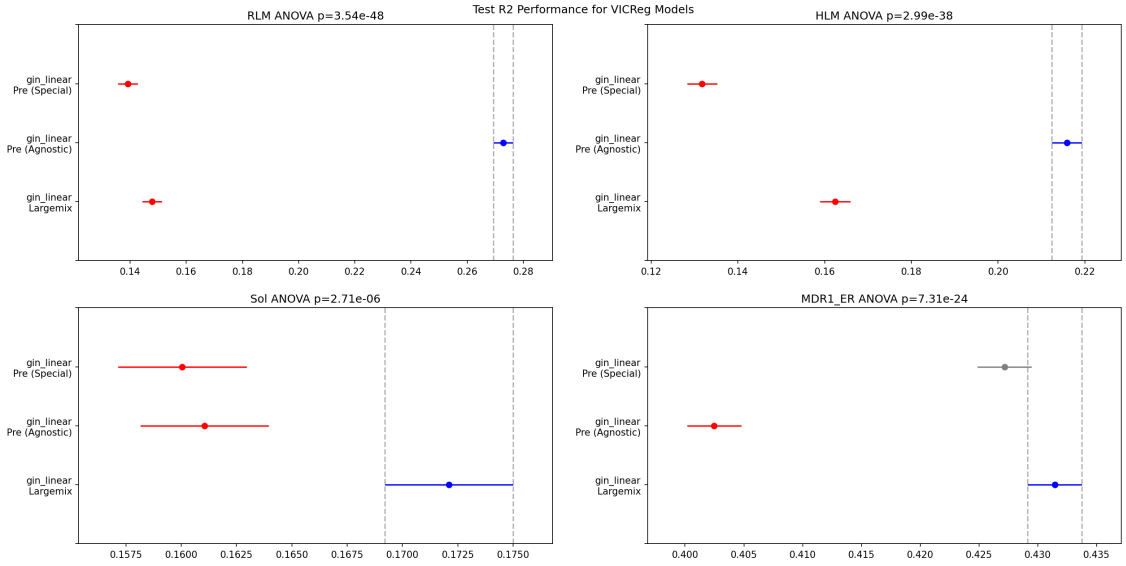

Figure B.32: Tukey plot to show significant test R2 differences on the ADME dataset among GIN encoders depending on the VICReg initializing strategy regarding linear evaluation.

Transfer Learning

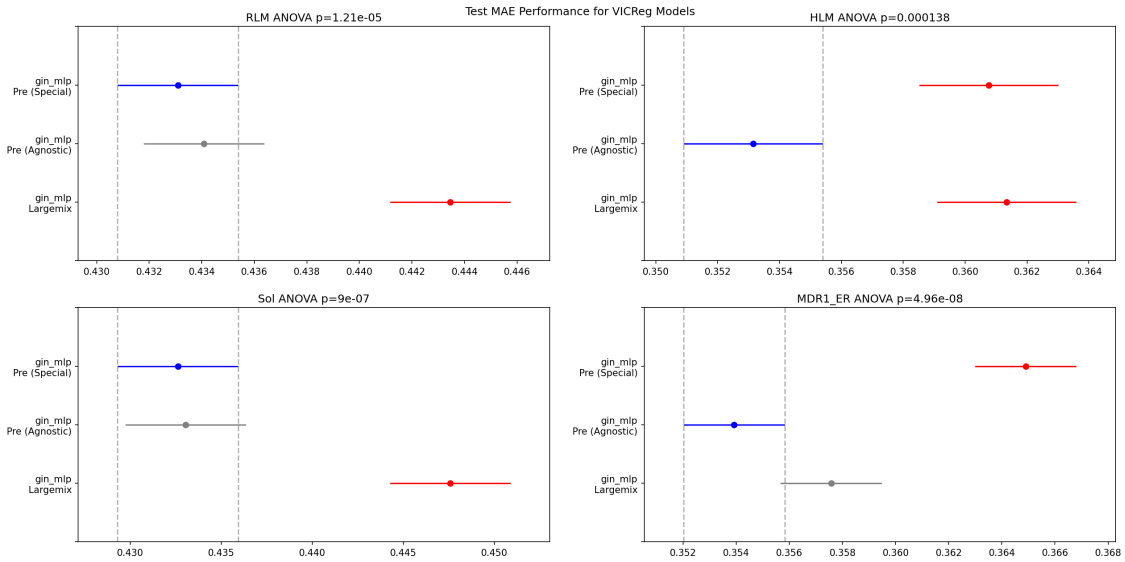

Figure B.33: Tukey plot to show significant test MAE differences on the ADME dataset among GIN encoders depending on the VICReg initializing strategy regarding transfer learning evaluation.

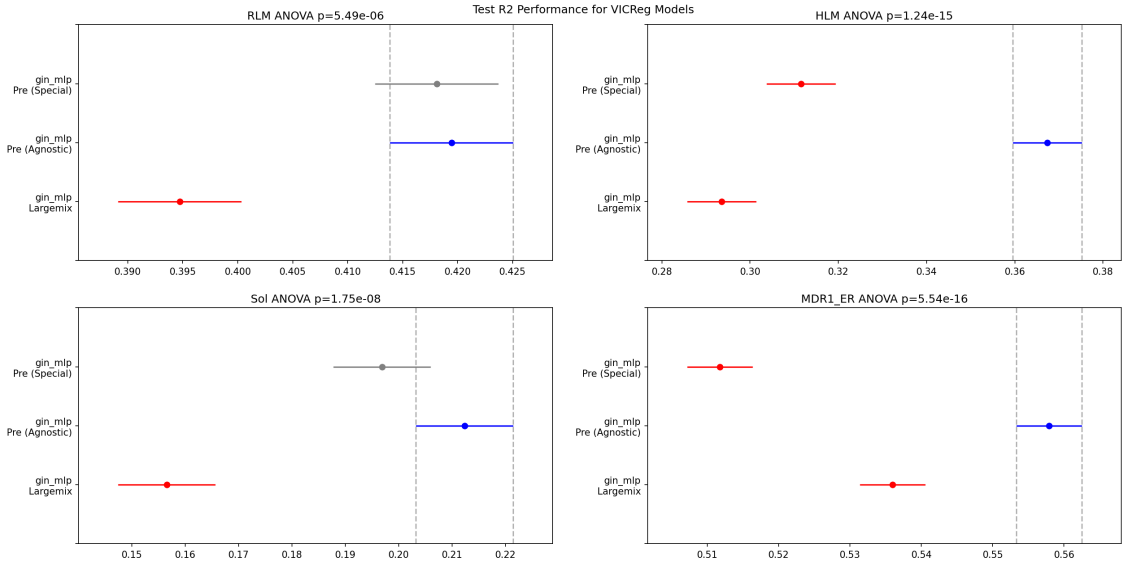

Figure B.34: Tukey plot to show significant test R2 differences on the ADME dataset among GIN encoders depending on the VICReg initializing strategy regarding transfer learning evaluation.

EGNN

Linear Evaluation

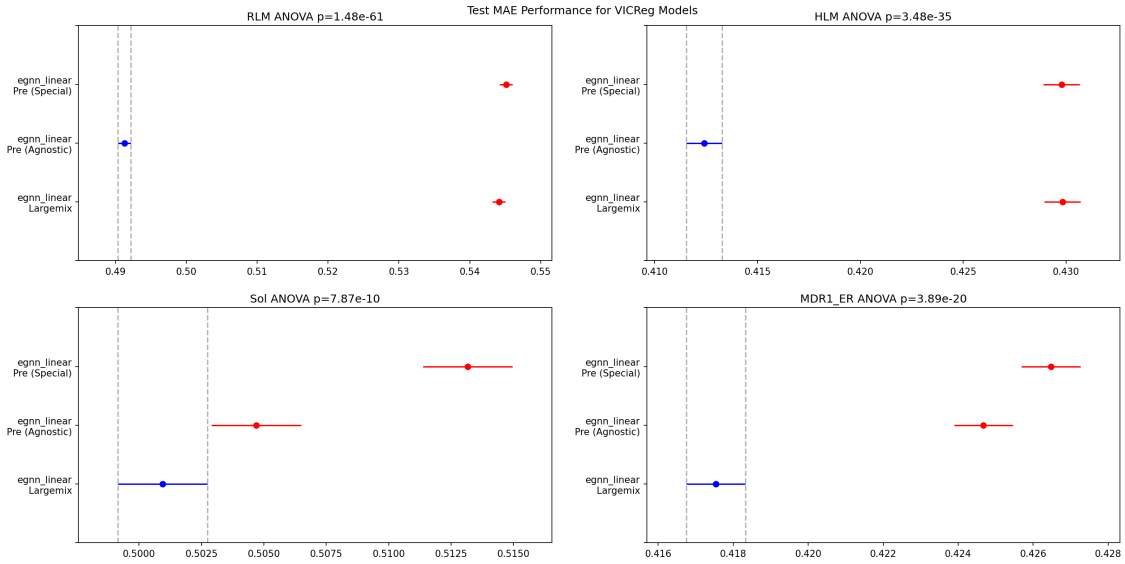

Figure B.35: Tukey plot to show significant test MAE differences on the ADME dataset among EGNN encoders depending on the VICReg initializing strategy regarding linear evaluation.

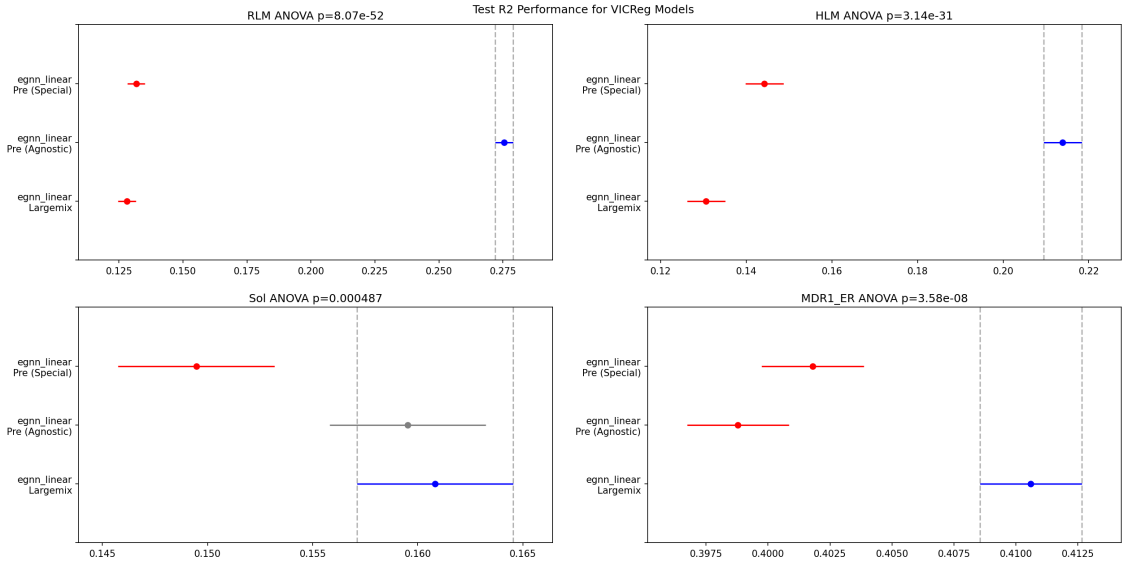

Figure B.36: Tukey plot to show significant test R2 differences on the ADME dataset among EGNN encoders depending on the VICReg initializing strategy regarding linear evaluation.

Transfer Learning

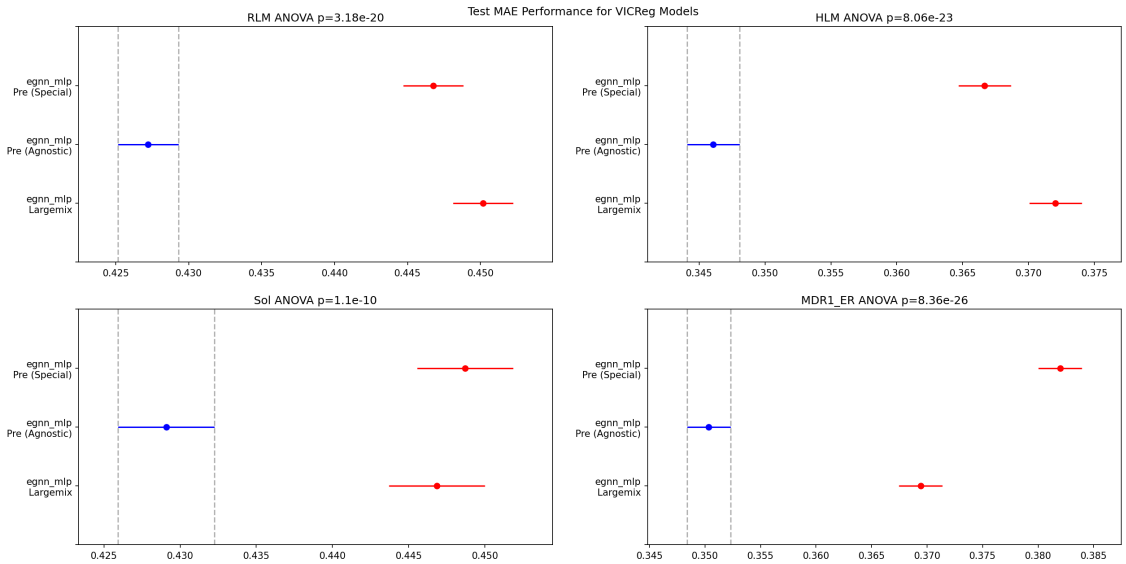

Figure B.37: Tukey plot to show significant test MAE differences on the ADME dataset among EGNN encoders depending on the VICReg initializing strategy regarding transfer learning evaluation.

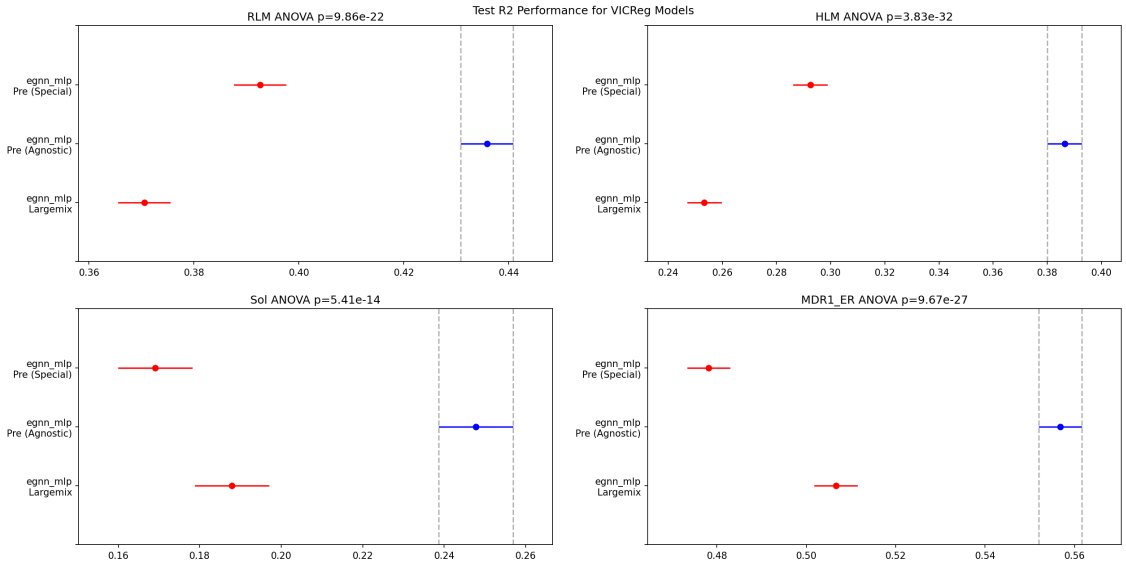

Figure B.38: Tukey plot to show significant test R2 differences on the ADME dataset among EGNN encoders depending on the VICReg initializing strategy regarding transfer learning evaluation.

LSTM

Linear Evaluation

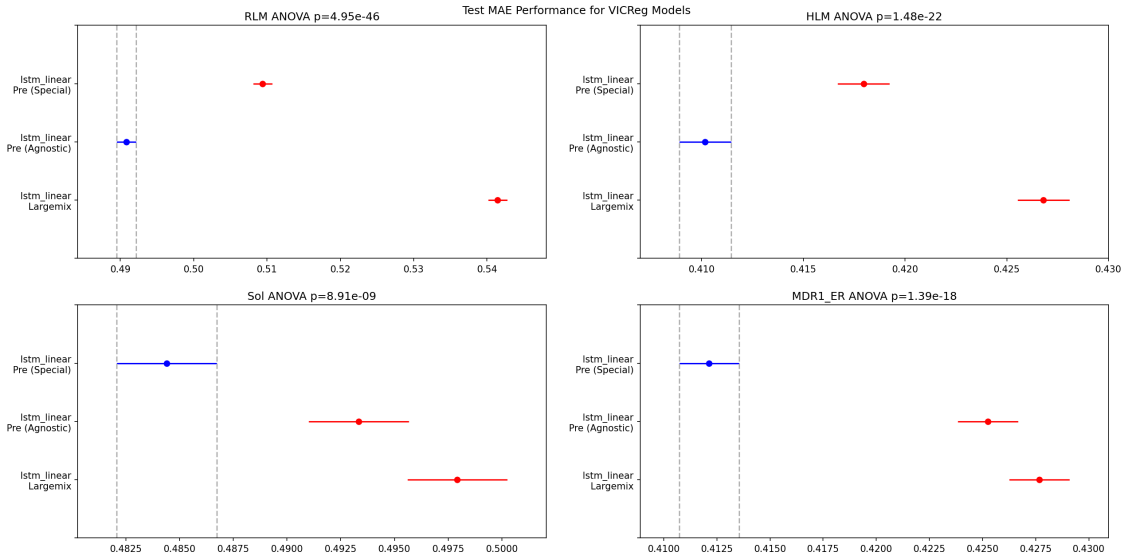

Figure B.39: Tukey plot to show significant test MAE differences on the ADME dataset among LSTM encoders depending on the VICReg initializing strategy regarding linear evaluation.

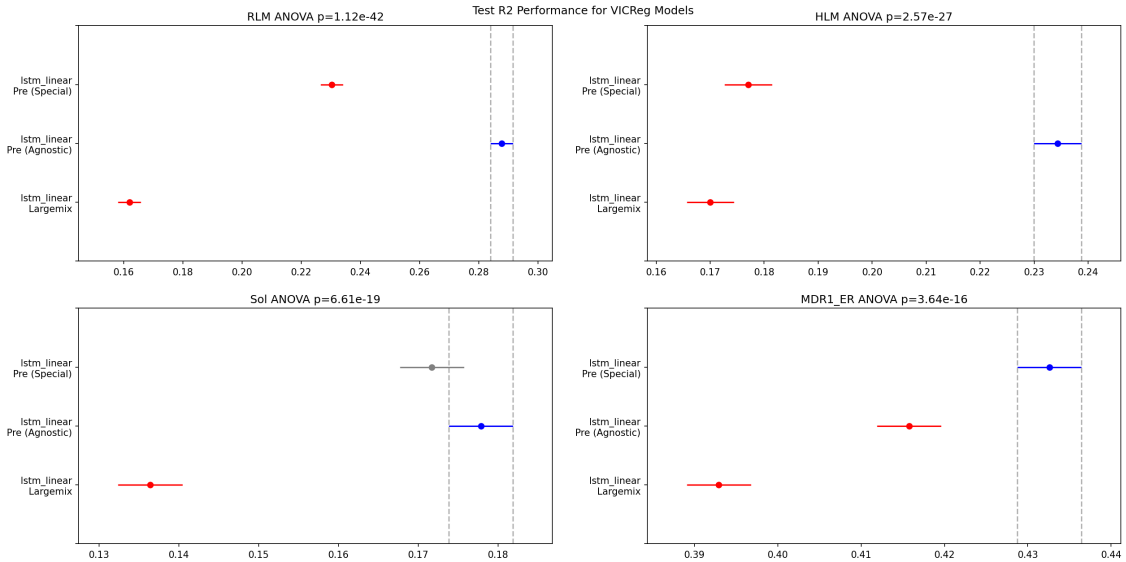

Figure B.40: Tukey plot to show significant test R2 differences on the ADME dataset among LSTM encoders depending on the VICReg initializing strategy regarding linear evaluation.

Transfer Learning

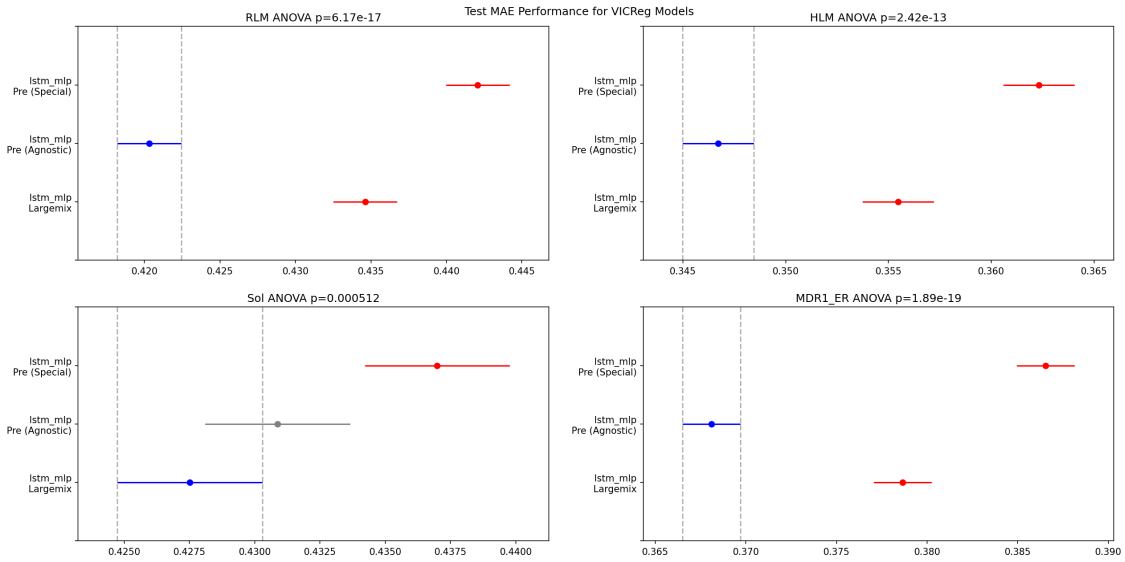

Figure B.41: Tukey plot to show significant test MAE differences on the ADME dataset among LSTM encoders depending on the VICReg initializing strategy regarding transfer learning evaluation.

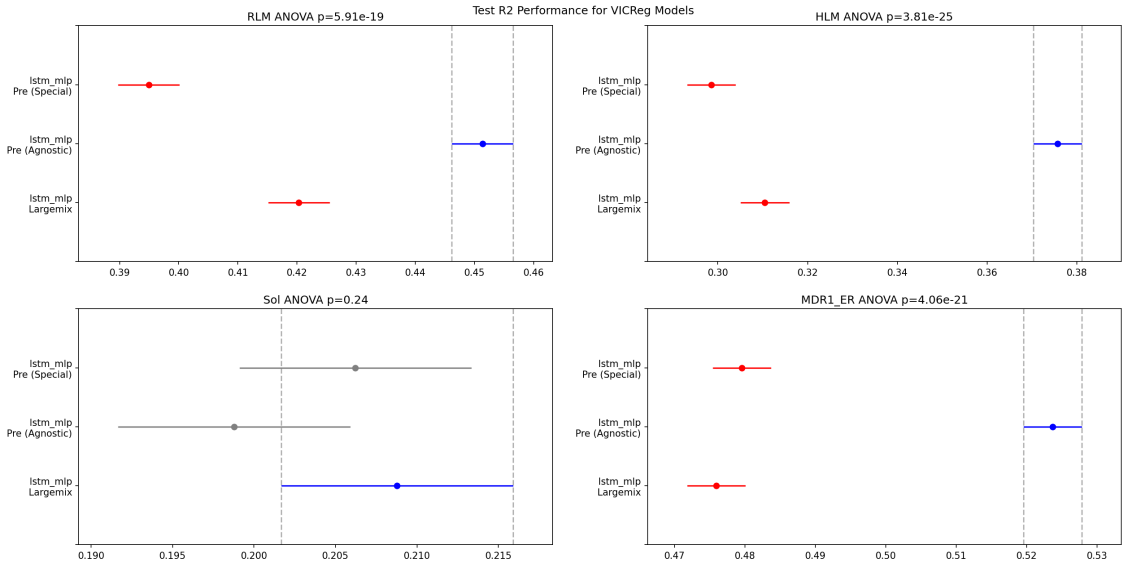

Figure B.42: Tukey plot to show significant test R2 differences on the ADME dataset among LSTM encoders depending on the VICReg initializing strategy regarding transfer learning evaluation.

B.3 Across Experiments

Aggregated

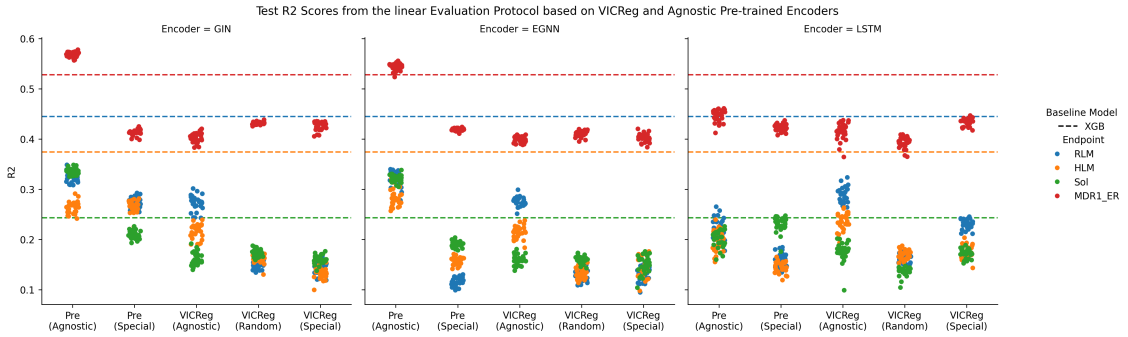

Figure B.43: Aggregated VICReg and pre-training results without VICReg are shown for each encoder on each endpoint (color-coded). Each point corresponds to a different random seed used in the linear evaluation protocol. Pre (Agnostic) refers to supervised agnostic pre-training, while Pre (Special) refers to modality-specific pre-trained encoders. VICReg (Agnostic) denotes the use of an agnostic pre-trained encoder for VICReg training, VICReg (Random) indicates VICReg pre-training with randomly initialized encoders, and VICReg (Special) shows results from VICReg with modality-specific pre-trained encoders. The y-axis represents the test R2 score obtained from evaluation on the ADME dataset.

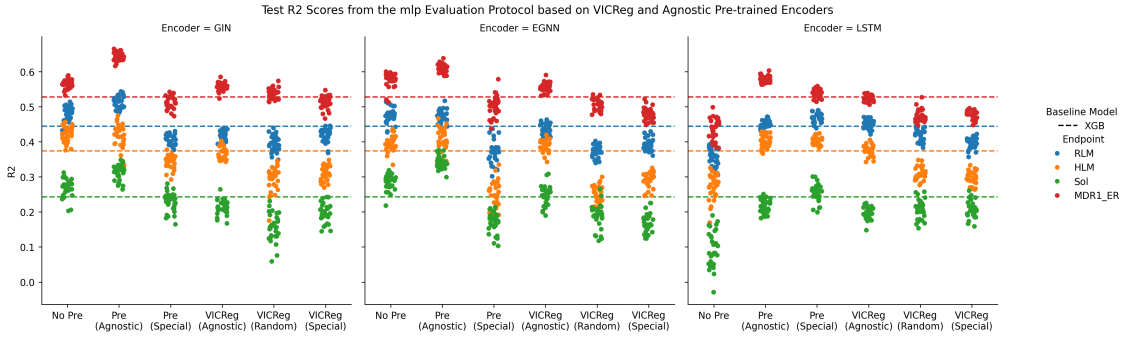

Figure B.44: Aggregated VICReg and pre-trained results without VICReg for each encoder on each endpoint (color). Each point corresponds to a different random seed used in the linear evaluation protocol. No Pre shows the results without pre-training but with optimized predictor and optimizer parameters. Pre (Agnostic) refers to supervised agnostic pre-training, while Pre (Special) refers to modality-specific pre-trained encoders. VICReg (Agnostic) denotes the use of an agnostic pre-trained encoder for VICReg training, VICReg (Random) indicates VICReg pre-training with randomly initialized encoders, and VICReg (Special) shows results from VICReg with modality-specific pre-trained encoders. The y-axis represents the test R2 score obtained from evaluation on the ADME dataset.

GIN

Linear Evaluation

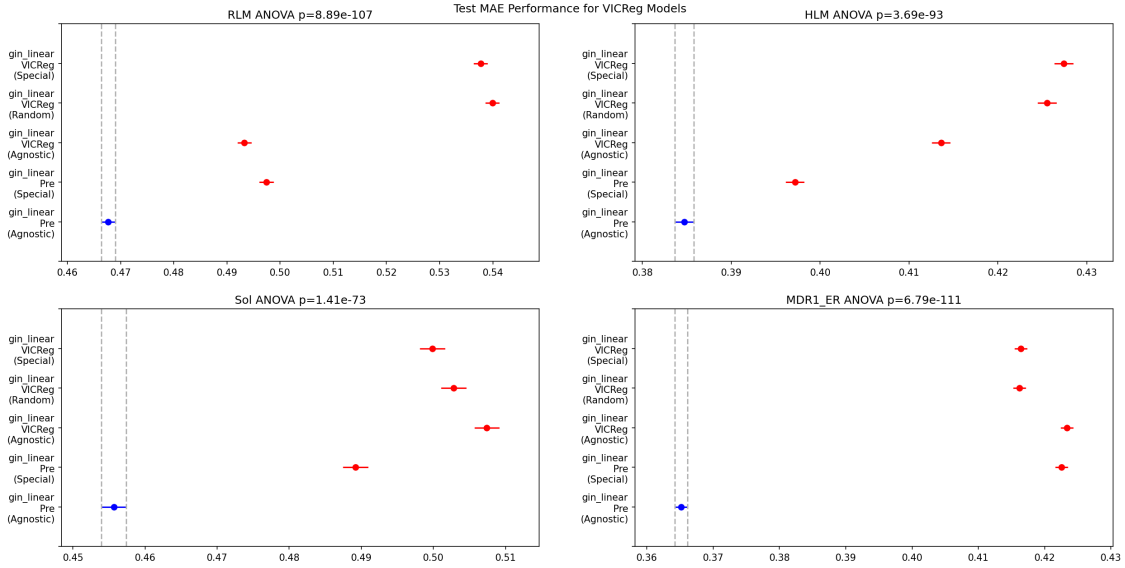

Figure B.45: Tukey plot to show significant test MAE differences on the ADME dataset among GIN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding linear evaluation.

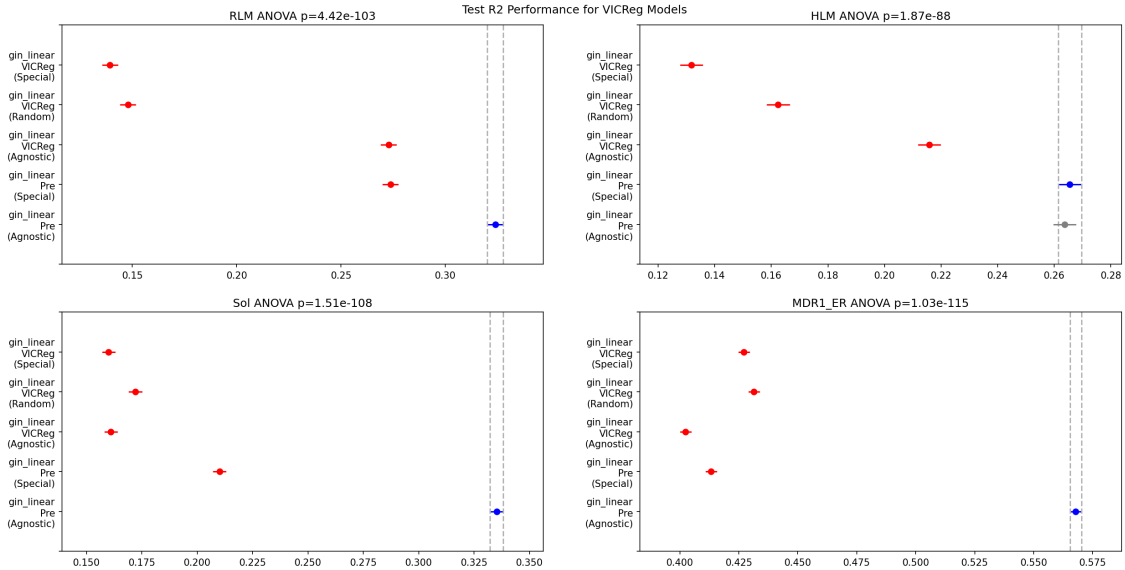

Figure B.46: Tukey plot to show significant test R2 differences on the ADME dataset among GIN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding linear evaluation.

Transfer Learning

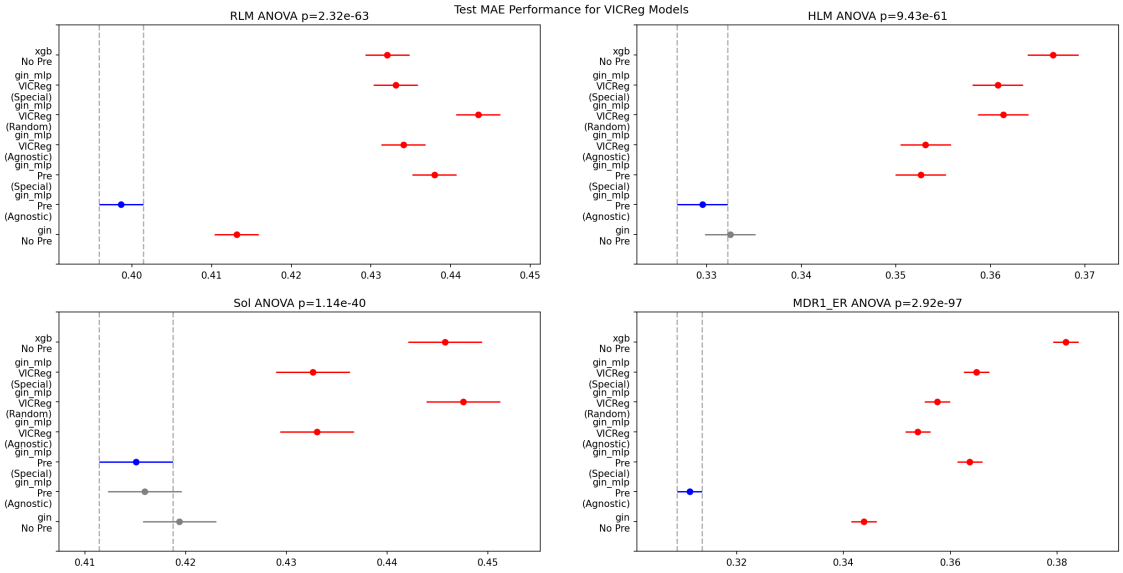

Figure B.47: Tukey plot to show significant test MAE differences on the ADME dataset among GIN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding transfer learning evaluation.

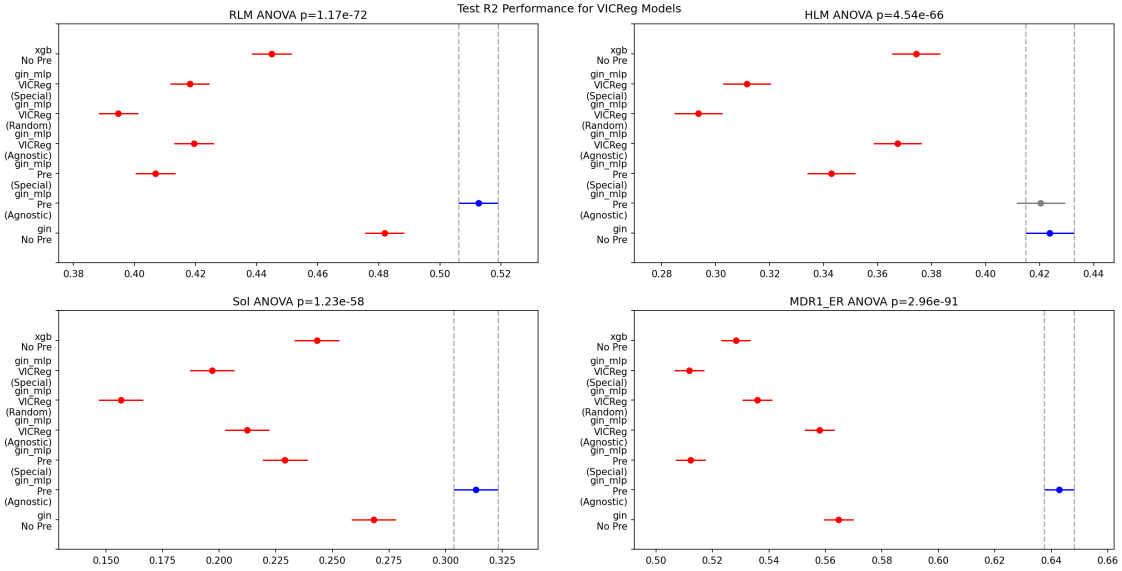

Figure B.48: Tukey plot to show significant test R2 differences on the ADME dataset among GIN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding transfer learning evaluation.

EGNN

Linear Evaluation

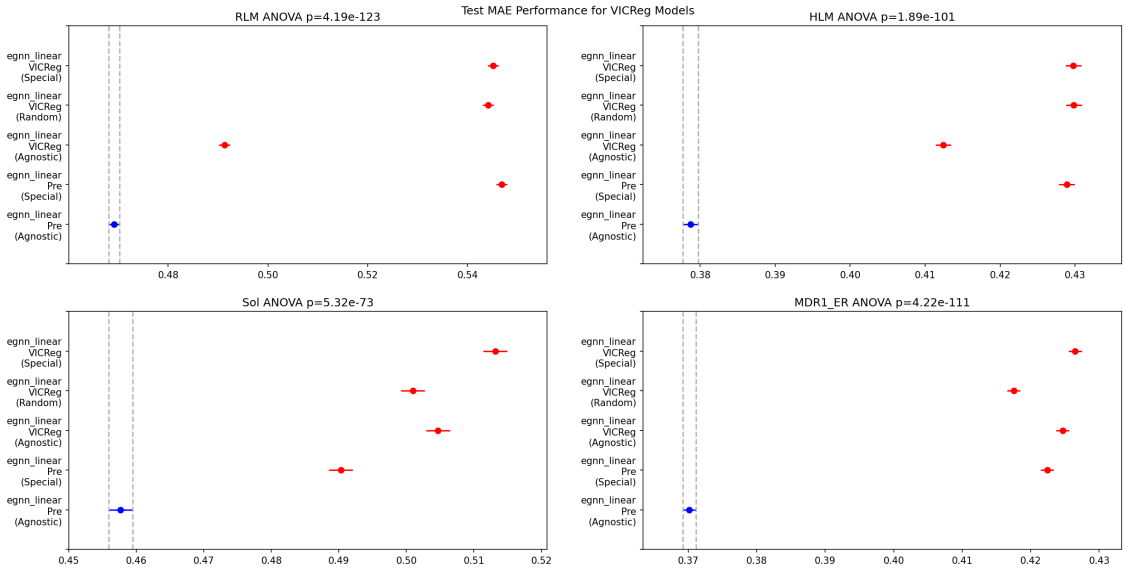

Figure B.49: Tukey plot to show significant test MAE differences on the ADME dataset among EGNN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding linear evaluation.

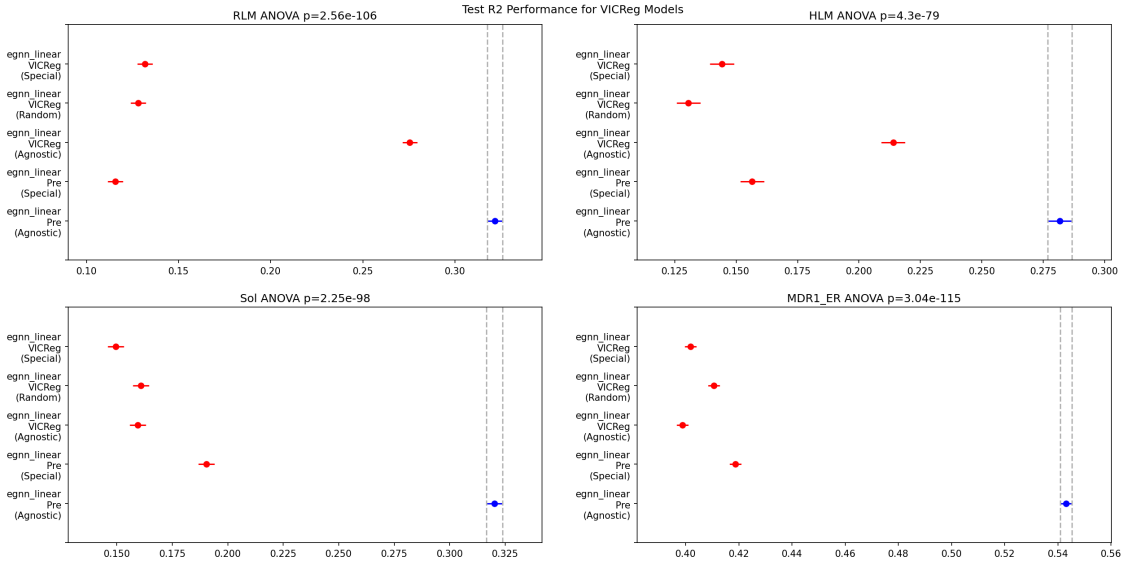

Figure B.50: Tukey plot to show significant test R2 differences on the ADME dataset among EGNN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding linear evaluation.

Transfer Learning

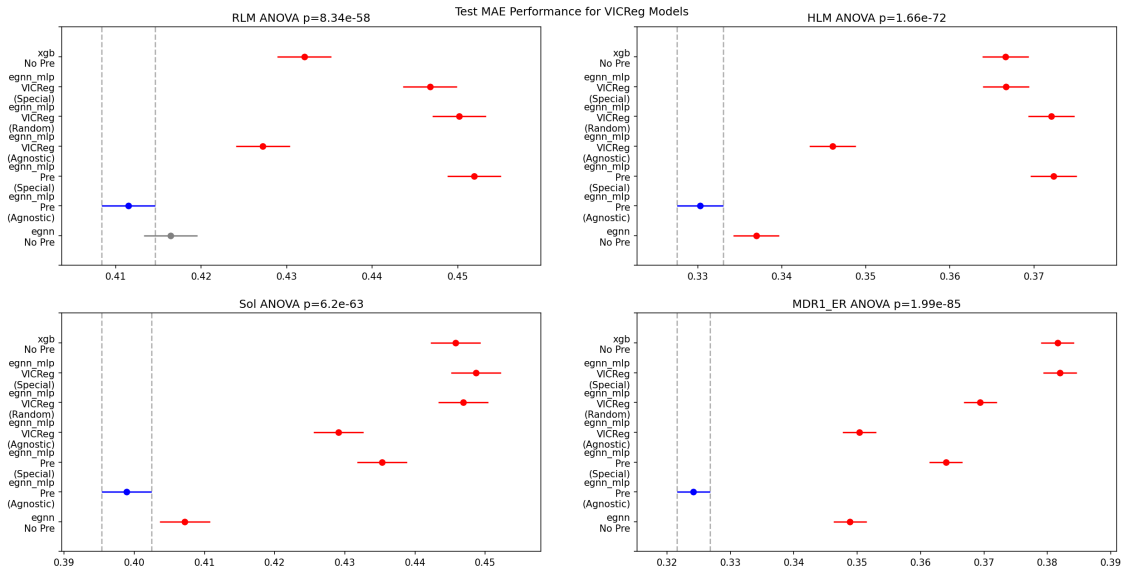

Figure B.51: Tukey plot to show significant test MAE differences on the ADME dataset among EGNN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding transfer learning evaluation.

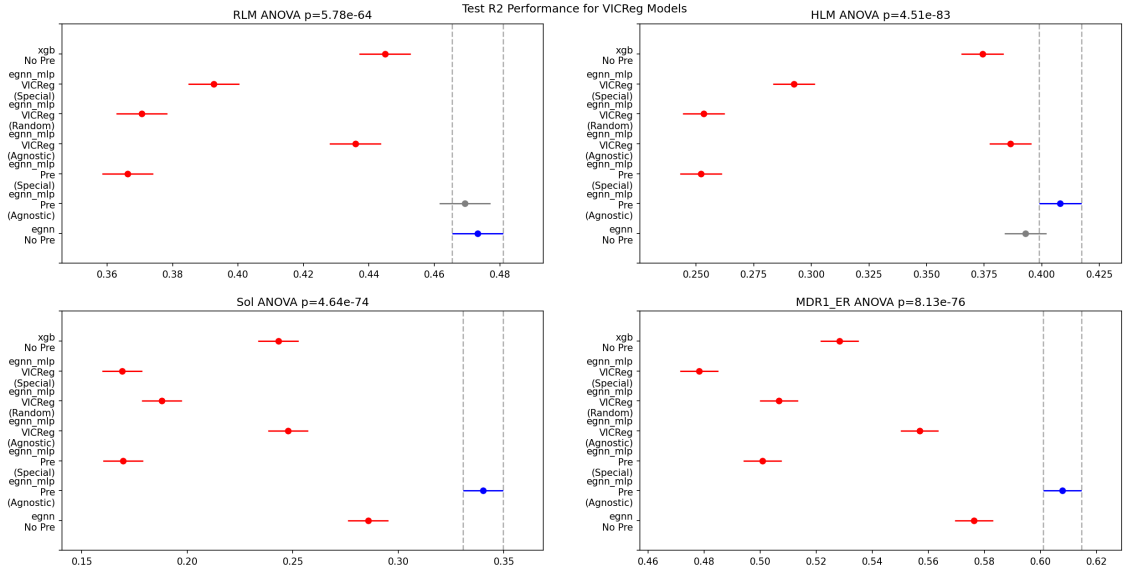

Figure B.52: Tukey plot to show significant test R2 differences on the ADME dataset among EGNN encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding transfer learning evaluation.

LSTM

Linear Evaluation

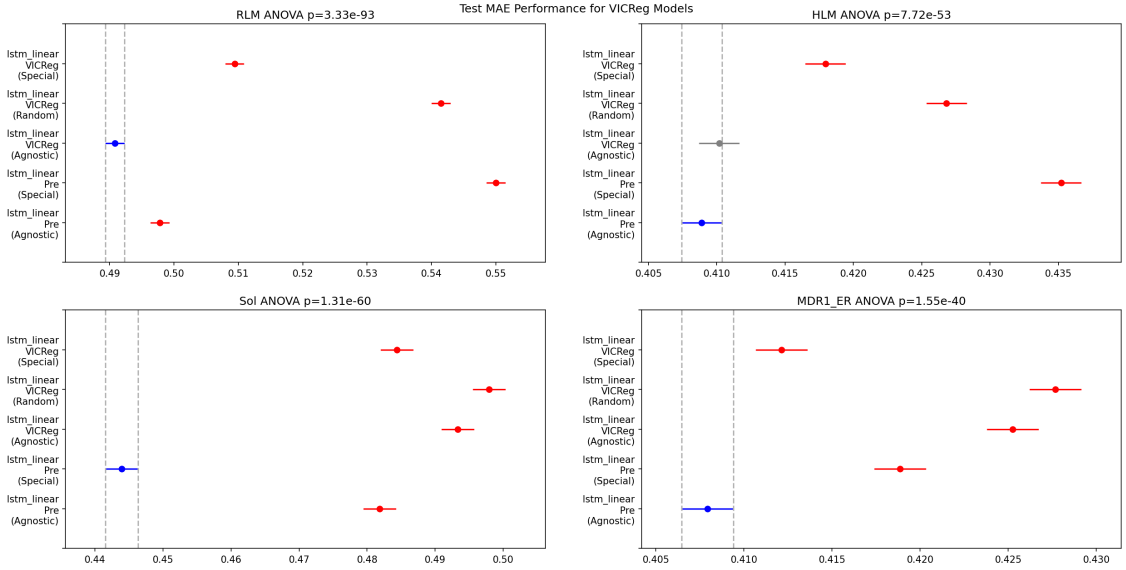

Figure B.53: Tukey plot to show significant test MAE differences on the ADME dataset among LSTM encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding linear evaluation.

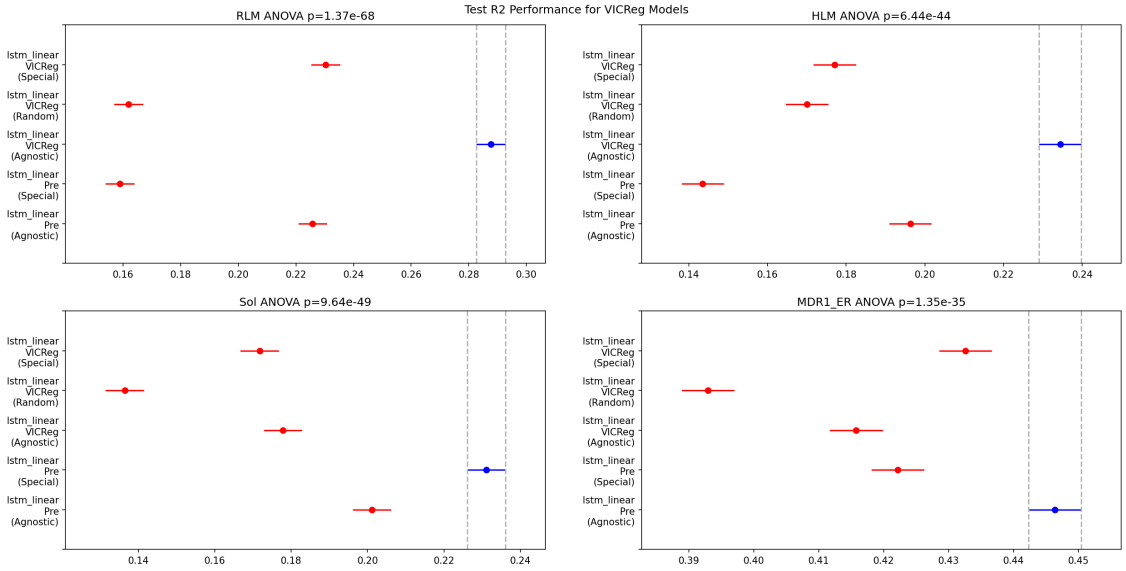

Figure B.54: Tukey plot to show significant test R2 differences on the ADME dataset among LSTM encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding linear evaluation.

Transfer Learning

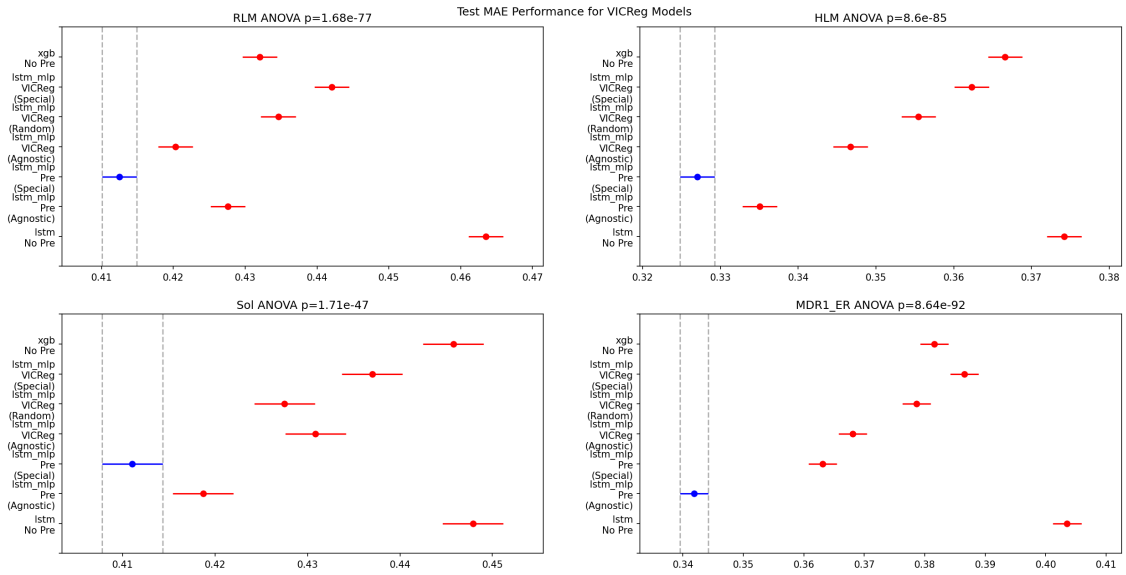

Figure B.55: Tukey plot to show significant test MAE differences on the ADME dataset among LSTM encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding transfer learning evaluation.

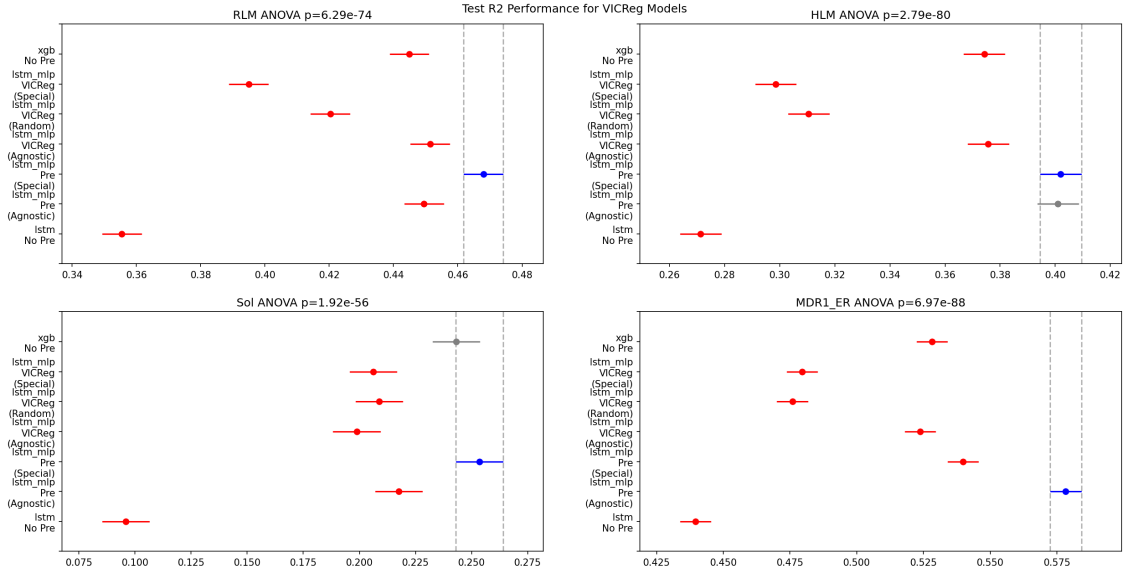

Figure B.56: Tukey plot to show significant test R2 differences on the ADME dataset among LSTM encoders depending on the VICReg with initializing strategies and conventional pre-training without VICReg regarding transfer learning evaluation.