

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Інститут комп'ютерних наук та інформаційних технологій
Кафедра систем штучного інтелекту

Гурбич Олександр Вікторович

Кваліфікаційна наукова праця
на правах рукопису
УДК 004.891

**МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ ХІМІЧНИХ
СПОЛУК ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ**

122 - комп'ютерні науки

Дисертація на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ /Гурбич О.В./

Науковий керівник
Матвійчук Ярослав Миколайович
доктор технічних наук, професор

Львів – 2023

АНОТАЦІЯ

Гурбич О.В. Методи та засоби аналізу хімічних сполук засобами штучного інтелекту. – Кваліфікаційна наукова праця на правах рукопису. Дисертація на здобуття наукового ступеня кандидата технічних наук (доктора філософії) за спеціальністю 122 "Комп'ютерні науки". – Національний університет "Львівська політехніка", Львів, 2022.

Зміст анотації. Дисертаційна робота присвячена розробці методів та засобів штучного інтелекту для інформаційної розробки та відбору найбільш перспективних молекул-кандидатів у лікарські речовини. Незважаючи на вдосконалення таких технологій як високопродуктивний скринінг, біотехнологія та комбінаторна хімія, вартість виведення на ринок нового препарату з поправкою на інфляцію подвоюється кожні дев'ять років. Рентабельність фармацевтичних досліджень невпинно спадає. Тому гостро постає потреба оптимізації процесу розробки нових ліків, зокрема, методами штучного інтелекту. Запропоновані у роботі методи поєднуються у єдину інформаційну систему для розробки лікарських речовин із заданими фізико-хімічними та біологічними властивостями, а також прогнозуванням придатності до синтезу у лабораторії.

У *першому* розділі проаналізовано існуючі методи та підходи машинного навчання до прогнозування молекулярної спорідненості, фактичного виходу продукту хімічної реакції та інформаційної розробки молекулярних структур. За результатами огляду сформовано завдання дисертаційної роботи.

У *другому* розділі запропоновано метод прогнозування молекулярної спорідненості між рецептором (велика біомолекула) та лігандами (малі органічні молекули). Основу складає метод мета-навчання: моделі другого рівня вивчають оптимальну комбінацію окремих моделей першого рівня у двох послідовних ансамблях - класифікаційному та регресійному. Кожен із базових ансамблів містить по шість моделей машинного навчання, які поєднуються методом нашарування.

У *третьому* розділі розроблено метод прогнозування виходу продукту хімічної реакції. Хімічний вихід - це відсоток реагентів, перетворених на цільовий продукт реакції. Прогнозування виходу продукту хімічної

реакції виконується за допомогою нової архітектури графової нейронної мережі. Мережа поєднує структурну інформацію про учасників хімічного перетворення, а також ознаки молекули та реакції. Ефективність графової нейронної мережі порівнювалася із логістичною регресією, методом опорних векторів, градієнтним бустингом та нейронними мережами-трансформерами. Порівнювалися такі молекулярні представлення як бінарні вбудування Моргана, вбудування SMILESVec та текстові представлення. Моделі були натреновані та оцінені на двох публічних наборах даних із одного класу реакцій та одному наборі даних від компанії Enamine із десяти типів реакцій. Дослідження було доповнене аналізом даних, результатів і помилок, а також впливу просторових факторів, побічних реакцій, ефективності виділення та очищення.

У *четвертому* розділі запропоновано метод, що поєднує кілька глибоких нейронних мереж для створення молекулярних структур із заданими властивостями. Генерування доповнюється виправленням хімічної будови молекул за допомогою рекурентної нейронної мережі з механізмом уваги. Запропонований ансамбль дозволяє створювати нові лікарські речовини, контролюючи ступінь розчинності, показник синтетичної доступності, площу полярної поверхні тощо.

У *п'ятому* розділі розроблено архітектурну діаграму інформаційної системи для створення лікарських речовин із бажаними біологічними та фізико-хімічними властивостями. Інформаційна система поєднує методи, описані у попередніх розділах: розробка молекул-кандидатів та комплексний контроль можливості їх синтезу у лабораторії, молекулярної маси, розчинності, топологічної полярної площі поверхні, очікуваного виходу продукту та молекулярної спорідненості до обраного рецептору.

Основні результати дисертації опубліковано у 10 наукових працях, зокрема: **три** статті – у наукових фахових періодичних виданнях України; **п'ять** - у закордонних фахових періодичних виданнях; **дві** публікації матеріалів міжнародних науково-технічних конференцій.

Ключові слова: машинне навчання, глибокі нейронні мережі, графові нейронні мережі, мета-навчання, молекулярна спорідненість, вихід хімічної реакції, генерація молекул-кандидатів.

ABSTRACT

Gurbych O.V. Methods of artificial intelligence for *in silico* drug design. Qualifying scientific work on manuscript rights.

Dissertation for getting the Doctor of Philosophy (Ph.D.) scientific degree in specialty 122 - Computer Science, Lviv Polytechnic National University, Ministry of Education and Science of Ukraine Lviv, 2022.

Supervisor: D.Sci., Prof., Matviychuk Y.M.

Despite technological advances in drug discovery, such as high-throughput screening, biotechnology, and combinatorial chemistry, the inflation-adjusted drug-to-market cost doubles every nine years [1]. The profitability of pharmaceutical research is falling and has become negative in this decade [2]. Therefore, optimizing drug development is urgently needed, particularly using artificial intelligence methods. A set of artificial intelligence methods that form an integrated approach to virtual drug design is suggested. The approach guides the generation of molecular structures by their biological affinity to the target and the level of synthetic accessibility.

The paper also proposes a way to facilitate trained neural networks that form the basis of the approach.

The dissertation is about methods and means of artificial intelligence for *in silico* generation and selection of the most promising candidate molecules for medicinal substances. Despite technological advances such as high-throughput screening, biotechnology, and combinatorial chemistry, the inflation-adjusted cost of bringing a new drug to market doubles every nine years [1]. The profitability of pharmaceutical research is falling and has become negative in recent years [2]. Therefore, optimizing drug development is urgently needed, particularly using artificial intelligence methods. The methods proposed in this work are combined into a single closed cycle of design of medicinal substances with specified physicochemical and biological properties and synthetic availability.

In the *first* section, the existing methods and approaches of machine learning for predicting molecular affinity, the actual yield of the product of a chemical reaction, and the generation of molecular structures are analyzed. Based on the results of the review, the task of the dissertation work was formed.

The *second* section proposes a meta-learning technology to predict the

molecular affinity between a receptor (large biomolecule) and ligands (small organic molecules) [3]. Meta-models study the optimal combination of individual basic models in two consecutive ensembles - classification and regression [4]. Each of the ensembles contains six machine learning models, which are combined using the stacking method [5].

In the *third* chapter, a technology is developed for predicting the yield of a chemical reaction in which a candidate molecule is synthesized. Chemical yield is the percentage of the reactants converted to the desired products. Chemists use predictive algorithms to select high-yielding reactions and score synthesis routes, saving time and reagents. This study suggests a novel graph neural network architecture for chemical yield prediction. The network combines structural information about participants of the transformation as well as molecular and reaction-level descriptors. It works with incomplete chemical reactions and generates reactants- product atom mapping. We show that the network benefits from advanced information by comparing it with several machine learning models and molecular representations. Models included Logistic Regression, Support Vector Machine, CatBoost, and Bidirectional Encoder Representations from Transformers. Molecular representations included ECFP Morgan fingerprints, SMILESVec embeddings, and textual. Classification and regression objectives were assessed for each model and feature set. The goal of each classification model was to separate zero- and non-zero-yielding reactions. The models were trained and evaluated on a proprietary dataset of ten reaction types. Also, the models were benchmarked on two public single-reaction type datasets. The study was supplemented with an analysis of data, results, errors, and the impact of steric factors, side reactions, isolation, and purification efficiency [6].

The *fourth* chapter proposes a method that combines several deep neural networks to generate unique molecules with given properties and constraints imposed by the models discussed in the previous two chapters. The generation is complemented by correcting the chemical structure of molecules with errors using a recurrent neural network with an attention mechanism. An analysis of chemical properties and an assessment of similarity to medicinal substances was carried out for the generated molecular structures. The proposed ensemble creates new unique medicinal substances by controlling the degree of solubil-

ity, synthetic availability, biological affinity to the target receptor, and other imposed constraints [7, 8].

In the *fifth* section, the software architecture of the platform for generating medicinal substances with the desired biological and physicochemical properties is developed. The platform combines the technologies described in the previous sections: generation of candidate molecules and comprehensive control of their synthetic accessibility, molecular weight, solubility, topological polar surface area, expected product yield in the synthesis reaction, and molecular affinity to the selected receptor.

The main scientific results of the dissertation were published in 10 works, in particular: **three** articles - in specialized scientific periodicals of Ukraine; **five** - in foreign professional journals; **two** publications - in materials of international scientific conferences.

Keywords: drug discovery, molecules, machine learning, deep learning, graph neural networks, meta-learning, molecular affinity, chemical reaction yield, chemical structures generation.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у фахових виданнях України:

1. Oleksandr Gurbych and Maksym Prymachenko. “Method for reductive pruning of neural networks and its applications”. In: Computer Systems and Information Technologies 3 (2022), pp. 40–48. DOI: 10.31891/csit-2022-3-5
2. Олександр Гурбич. “Метод машинного навчання для створення нових лікарських речовин із заданими властивостями”. In: Наук. вісник Ужгород. ун-ту 40.1 (2022), pp. 126–145. DOI: 10.24144/2616-7700.2022.1(40).126-145
3. Олександр Гурбич. “Метод мета-навчання для визначення молекулярної спорідненості”. In: Вісник Хмельницького національного університету 307.2 (2022), pp. 14–24. DOI: 10.31891/2307-5732-2022-307-2-14-24

Статті у закордонних фахових періодичних виданнях:

4. Dzvenymyra Yarish, Sofiya Garkot, Oleksandr Grygorenko, Dmytro Radchenko, Yurii Moroz, and Oleksandr Gurbych. “Advancing molecular graphs with descriptors for the prediction of chemical reaction yields”. In: Journal of Computational Chemistry 43.28 (2022), pp. 1887–1935. DOI: 10.1002/jcc.27016
5. Tymofii Nikolaienko, Oleksandr Gurbych, and Maksym Druchok. “Complex machine learning model needs complex testing: Examining predictability of molecular binding affinity by a graph neural network”. In: Journal of Computational Chemistry 43.10 (2022), pp. 728–739. DOI: 10.1002/jcc.26831
6. Maksym Druchok, Dzvenymyra Yarish, Sofiya Garkot, Tymofii Nikolaenko, and Oleksandr Gurbych. “Ensembling machine learning models to boost molecular affinity prediction”. In:

Computational Biology and Chemistry 93 (2021). DOI: 10.1016/j.compbiolchem.2021.107529

7. Maksym Druchok, Dzvenymyra Yarish, Oleksandr Gurbych, and Mykola Maksymenko. “Toward efficient generation, correction, and properties control of unique drug-like structures”. In: Journal of Computational Chemistry 42.11 (2021), pp. 746–760. DOI: 10.1002/jcc.26494
8. Grygoriy Dolgonos, Alexey Tsukanov, Sergey Psakhie Psakhie, Oleg Lukin, Oleksandr Gurbych, and Alexander Shivanyuk Shivanyuk. “Theoretical studies of capsular complexes of C₂V-symmetrical resorcin[4]arene tetraesters with tetramethylammonium cation”. In: Computational and Theoretical Chemistry 1159 (2019), pp. 12–17. DOI: 10.1016/j.comptc.2019.05.006

Матеріали конференцій:

9. Maksym Druchok, Dzvenymyra Yarish, Oleksandr Gurbych, and Mykola Maksymenko. “Towards Efficient Generation, Correction and Properties Control of Unique Drug-like Structures”. In: ChemRxiv (2019). DOI: 10.26434/chemrxiv.9941858.v1
10. Oleksandr Gurbych, Maksym Druchok, Dzvenymyra Yarish, and Sofiya Garkot. “High throughput screening with machine learning”. In: Proceedings of the Advances in Neural Information Processing Systems 32 (NeurIPS 2019) (Vancouver, Canada). Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Dec. 2019. ISBN: 9781713807933. DOI: 10.48550/arXiv.2012.08275

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	13
ВСТУП	16
1 РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА ПІДХОДІВ МАШИННОГО НАЧАННЯ ДО ПРОГНОЗУВАННЯ МОЛЕКУЛЯРНИХ ВЛАСТИВОСТЕЙ.	21
1.1 Аналіз методів машинного навчання для прогнозування молекулярної спорідненості.	21
1.2 Аналіз методів машинного навчання для прогнозування фактичного виходу продукту хімічної реакції.	27
1.2.1 Реакційна здатність.	27
1.2.2 Хімічні дескриптори.	27
1.2.3 Публічно доступні набори даних.	28
1.2.4 Моделі-трансформери.	30
1.2.5 Графові нейронні мережі.	31
1.3 Аналіз методів машинного навчання для генерації молекулярних структур.	32
1.3.1 Повнозв'язні нейронні мережі.	32
1.3.2 Згорткові нейронні мережі.	33
1.3.3 Графові нейронні мережі.	33
1.3.4 Автокодувальники та генеративні змагальні нейронні мережі.	34
1.3.5 виправлення хімічних помилок.	34
1.4 Актуальність та постановка завдань.	35
1.4.1 Прогнозування молекулярної спорідненості.	35
1.4.2 Прогнозування виходу продукту хімічної реакції. . .	36
1.4.3 Генерація молекулярних структур із заданими властивостями.	36
1.4.4 Платформа для розробки нових лікарських речовин.	37
1.5 Висновки.	38
2 РОЗДІЛ 2. МЕТОД ПРОГНОЗУВАННЯ МОЛЕКУЛЯРНОЇ СПОРІДНЕНОСТІ.	39

2.1	Підготовка даних.	39
2.1.1	Підготовка класифікаційного набору даних.	39
2.1.2	Підготовка регресійного набору даних.	40
2.1.3	Результати підготовки даних.	41
2.1.4	Розподіл функціональних груп.	42
2.1.5	Інженерія ознак.	44
2.1.6	Підготовка даних для редуکتивного спрощення.	45
2.2	Методологія.	46
2.2.1	Мета-навчання.	50
2.2.2	Метод опорних векторів.	51
2.2.3	Випадковий ліс.	52
2.2.4	Градiєнтний бустинг.	52
2.2.5	Нейронна мережа прямого поширення.	53
2.2.6	Графова нейронна мережа.	54
2.2.7	Двоспрямовані трансформери.	54
2.2.8	Редуکتивне спрощення.	56
2.3	Результати.	60
2.3.1	Поодинокі регресійні та класифікаційні моделі.	60
2.3.2	Стекінг поодиноких моделей та мета-навчання.	61
2.3.3	Редуکتивне спрощення для моделей молекулярної спорідненості.	63
2.4	Висновки.	65

3 РОЗДІЛ 3. МЕТОД ПРОГНОЗУВАННЯ ВИХОДУ ПРОДУКТУ ХІМІЧНОЇ РЕАКЦІЇ. 67

3.1	Аналіз даних.	67
3.1.1	Публічні набори даних.	67
3.1.2	Пропріетарний набір даних Enamine.	69
3.2	Підготовка даних.	74
3.2.1	Підготовка даних для прогнозування виходу продукту хімічної реакції.	74
3.2.2	Підготовка даних для редуکتивного спрощення.	76
3.3	Методологія.	77
3.3.1	Лінійні моделі.	78
3.3.2	Моделі на основі градієнтного бустингу.	78

3.3.3	Моделі-трансформери.	79
3.3.4	Графові нейронні мережі.	80
3.4	Результати.	84
3.4.1	Прогнозування фактичного виходу продукту для одного класу реакцій.	84
3.4.2	Прогнозування фактичного виходу продукту для багатьох класів реакцій.	86
3.4.3	Аналіз помилок прогнозування фактичного виходу продукту.	89
3.4.4	Прогнозування виходу продукту для класу реакції поза тренувальною вибіркою.	93
3.4.5	Зменшення розмірності відбитків реакцій і аналіз проєкцій	94
3.4.6	Редуктивне спрощення моделей оцінки виходу продукту хімічної реакції.	97
3.5	Висновки.	99
4	РОЗДІЛ 4. МЕТОД РОЗРОБКИ ЛІКАРСЬКИХ МОЛЕКУЛ ІЗ ЗАДАНИМИ ВЛАСТИВОСТЯМИ.	100
4.1	Опис та підготовка даних.	100
4.1.1	Спосіб представлення молекул.	100
4.1.2	Підготовка даних для моделі-генератора молекулярних структур.	101
4.1.3	Підготовка даних для моделі виправлення помилок.	102
4.2	Методологія.	102
4.2.1	Архітектура та тренування моделі-генератора молекулярних структур.	103
4.2.2	Модель для виправлення хімічних помилок.	105
4.2.3	Відбір нових точок з латентного простору.	106
4.2.4	Молекулярна динаміка для перевірки очікуваної розчинності.	107
4.3	Результати.	108
4.3.1	Вибір розміру набору даних для попереднього тренування автокодувальника.	108

4.3.2	Дотренування автокодувальника із додатковим модулем прогнозування розчинності.	108
4.3.3	Тренування та оцінка моделей для виправлення помилок хімічної будови.	109
4.3.4	Аналіз створених молекулярних структур.	110
4.3.5	Оцінка прогнозованої розчинності.	115
4.4	Висновки.	118
5	РОЗДІЛ 5. АРХІТЕКТУРНА ДІАГРАМА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ РОЗРОБКИ ЛІКАРСЬКИХ РЕЧОВИН.	119
5.1	Концептуальна архітектурна діаграма інформаційної системи для розробки лікарських речовин.	119
5.2	Детальна архітектурна діаграма інформаційної системи для розробки лікарських речовин.	122
5.3	Висновки.	125
	ВИСНОВКИ	126
	СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	128
I	Додаток А. АКТИ ВПРОВАДЖЕННЯ	160
II	Додаток Б. ПРОГРАМНИЙ КОД МЕТОДУ ПРОГНОЗУВАННЯ СПОРІДНЕНОСТІ	171
i	Підготовка даних	171
ii	Завантаження графів	177
iii	Архітектура графової нейронної мережі для прогнозування молекулярної спорідненості	183
iv	Налаштування графової нейронної мережі	185
v	Алгоритм тренування графової нейронної мережі	185
III	Додаток В. ПРОГРАМНИЙ КОД МЕТОДУ ПРОГНОЗУВАННЯ ВИХОДУ ПРОДУКТУ ХІМІЧНОЇ РЕАКЦІЇ	189

i	Архітектура графової нейронної мережі	189
ii	Підготовка даних	199
iii	Інженерія ознак	206
iv	Алгоритм тренування графової нейронної мережі	215
v	Алгоритм перехресної оцінки графової нейронної мережі . .	222
vi	Алгоритм прогнозування виходу продукту хімічної реакції .	225

IV Додаток Г. ПРОГРАМНИЙ КОД МЕТОДУ РОЗРОБКИ МОЛЕКУЛ-КАНДИДАТІВ У ЛІКАРСЬКІ РЕЧОВИНИ 231

i	Підготовка даних	231
ii	Архітектура автокодувальника	235
iii	Налаштування автокодувальника	236
iv	Алгоритм тренування автокодувальника	237
v	Архітектура класифікаційної моделі	238
vi	Налаштування класифікаційної моделі	238
vii	Алгоритм тренування класифікаційної моделі	239
viii	Архітектура регресійної моделі	240
ix	Налаштування регресійної моделі	241
x	Алгоритм тренування регресійних моделей	241
xi	Архітектура моделі виправлення помилок	242
xii	Налаштування моделі виправлення помилок	245
xiii	Тренування моделі виправлення помилок	246
xiv	Каскад моделей	247
xv	Розробка молекул-кандидатів у лікарські речовини	249
xvi	Прогнозування властивостей молекул-кандидатів у лікарські речовини	250

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. ШІ - Штучний Інтелект;
2. МН - Машинне Навчання;
3. НМ - Нейронна Мережа;
4. ШНМ - Штучна Нейронна Мережа;
5. ЗНМ - Згортова Нейронна Мережа;
6. ML - machine learning;
7. AI - artificial intelligence;
8. SL - Supervised Learning;
9. UL - Unsupervised Learning;
10. RL - reinforcement learning;
11. DRL - deep reinforcement learning;
12. RF - Random Forest;
13. CB - CatBoost;
14. AE - Autoencoder;
15. VAE - Variational Autoencoder;
16. GLM - Generalized Linear Model;
17. NB - Naive Bayes;
18. BN - Bayesian Network;
19. FNN - Feed-Forward Neural Network;
20. CNN - Convolutional Neural Network;
21. SVM - Support Vector Machine;

22. BERT - Bidirectional Encoder Representations from Transformers;
23. GNN - Graph Neural Network;
24. GAN - Generative Adversarial Network;
25. MPNN - message-passing neural network;
26. RD-MPNN - Directed Message-Passing Neural Network for chemical Reaction properties prediction;
27. LSTM - Long Short-Term Memory;
28. ALSTM - Attention-based Long Short-Term Memory;
29. SOTA - State-Of-The-Art;
30. MSE - Mean Squared Error;
31. RMSE - Root Mean Squared Error;
32. MAE - Mean Average Error;
33. AUC - Area Under the ROC Curve;
34. ROC - Receiver Operating Characteristic curve;
35. TPR - True Positive Rate;
36. FPR - False Positive Rate;
37. ACC - Accuracy;
38. PRC - Precision;
39. RCL - Recall;
40. IoU - Intersection over Union;
41. LogLoss - Logistic Loss;
42. RBF - Radial Basis Function;
43. ReLU - Rectified Linear Unit;

44. PReLU - Parametric Rectified Linear Unit;
45. IQR - Interquartile Range;
46. SMILES - Simplified Molecular-Input Line-Entry System;
47. SMARTS - SMILES Arbitrary Target Specification;
48. FASTA - FAST-All;
49. ECFP4 - Extended Connectivity Fingerprint with maximum diameter 4;
50. TPSA - Topological Polar Surface Area;
51. LogP - a Log of octanol-water Partition coefficient;
52. LogS - a 10-based Logarithm of the Solubility of a molecule in water;
53. SAS - Synthetic Accessibility Score;
54. MW - Molecular Weight;
55. QSAR - Quantitative Structure-Activity Relationship;
56. ADME - Absorption, Distribution, Metabolism, Excretion;
57. ADMET - Absorption, Distribution, Metabolism, Excretion, Toxicity;
58. SMOTE - Synthetic Minority Oversampling Technique;
59. DFT - Density-Functional Theory;
60. BBBP - Blood-Brain Barrier Permeability;
61. USPTO - the United States Patent and Trademark Office;

ВСТУП

Актуальність теми. Створення нових лікарських речовин — це ітеративний процес, що включає такі фази як відкриття, розрахункові випробування, преклінічні та клінічні дослідження, перевірка та затвердження регулюючими органами. Лише після цього лікарські молекули надходять у виробництво та назначаються хворим [13, 14, 15]. Виведення однієї лікарської речовини на ринок займає 10-15 років та коштує 2,6 мільярда доларів США [16, 17]. Найбільші витрати при цьому йдуть на синтез та випробування молекул-кандидатів, які були відбраковані на наступних стадіях через токсичність, низьку селективність тощо. Офіційна статистика свідчить, що лише 12% нових молекул, які надходять у клінічні випробування, зрештою отримують схвалення Управління з контролю за харчовими продуктами й ліками США (FDA). Лише до 5% з 7000 рідкісних захворювань відомі доступні ліки. Сучасна наука дослідила лише крихітну частину потенційних лікарських речовин: синтезовано лише 10^8 малих молекул з понад 10^{60} можливих [18, 19]. Синтез нових молекул-кандидатів вимагає збільшення інвестицій у дослідження і розробки [20], головним чином через складність пошуку молекул із необхідними властивостями.

Методи машинного навчання довели свою результативність у багатьох областях [21]. Глибокі нейронні мережі (DNN) успішно використовуються при розпізнаванні природньої мови, зображень та відео, проте універсальність глибоких нейронних мереж у вивченні представлень даних дозволяє розширити сфери їхнього застосування до наукових проблем [22, 23]. Хімічна та фармацевтична промисловість виявляють значний інтерес до методів машинного навчання та глибоких нейронних мереж, які допомагають підвищити ефективність розробки нових матеріалів [24, 25, 26]. У цьому відношенні доречно згадати нещодавню роботу команди Жаворонкова [27], у якій повідомляється про створення лікарської речовини з використанням машинного навчання усього за 21 день, що безпрецедентно скорочує доклінічний етап виробництва нових ліків.

Тому завдання дисертаційної роботи з розроблення методів та засобів аналізу хімічних сполук засобами штучного інтелекту та наступне

поєднання створених елементів у вигляді розрахункової платформи повного циклу дизайну ліків є актуальним.

Зв'язок роботи з науковими програмами, планами і темами.

Дисертація виконувалася відповідно до пріоритетних напрямків науково-дослідних робіт Національного університету “Львівська політехніка” (№ держ. реєстру 0119U002257), відповідно до координаційних планів Міністерства освіти і науки України.

Метою дисертаційної роботи є розроблення методів та засобів аналізу хімічних сполук засобами штучного інтелекту.

Досягнення поставленої мети включає наступні задачі:

1. Проаналізувати існуючі методи та підходи машинного навчання до прогнозування молекулярної спорідненості, фактичного виходу продукту хімічної реакції та генерації молекулярних структур; сформулювати завдання дисертаційної роботи.
2. Розробити метод штучного інтелекту для прогнозування молекулярної спорідненості.
3. Розробити модель прогнозування виходу продукту хімічної реакції.
4. Розробити метод розробки молекулярних структур із заданими властивостями. Розробити молекули-кандидати із заданими властивостями та перевірити властивості у симуляційному експерименті.
5. Розробити модифікацію методу редукції для спрощення та одночасного покращення ефективності нейронних мереж, видалити надлишкові ваги у перелічених вище моделях, що відносяться до нейронних мереж.
6. Розробити архітектуру системи, яка поєднує усі вищезазначені модулі у комплексну технологію дизайну молекул-кандидатів до лікарських речовин.

Об'єкт дослідження – процеси аналізу хімічних сполук засобами штучного інтелекту.

Предмет дослідження – методи, моделі та інформаційні технології аналізу хімічних сполук.

Методи дослідження. Аналіз спорідненості хімічних сполук проводився методом опорних векторів, випадковим лісом, градієнтним бустингом, графовими нейронними мережами та нейронними мережами прямого поширення, а також моделями-трансформерами, методом мета-навчання та поєднанням ансамблів моделей машинного навчання у каскади: класифікаційного та регресійного. Прогнозування виходу продукту хімічної реакції виконувалось деревами рішень, нейронними мережами прямого поширення, моделями-трансформерами. Вирішальними стали розробка та застосування новітньої архітектури графової нейронної мережі. Генерація молекулярних структур реалізована із використанням повнозв'язного автокодувальнику із додатковими регресійними та класифікаційними виходами для визначення схожості згенерованої молекули на ліки та апроксимації властивостей. Апробація стабільності та розчинності згенерованих молекул-кандидатів проводилася із використанням пакету молекулярного моделювання *DL_POLY_4*, силове поле *OPLS – AA*. Видалення надлишкових ваг повнозв'язних нейронних мереж здійснювалося за допомогою методу редуکتивного спрощення (модифікований метод редукції).

Наукова новизна одержаних результатів полягає у:

1. *Вперше* розроблено метод прогнозування молекулярної спорідненості, що поєднує послідовні ансамблі моделей машинного навчання першого рівня та узагальнені лінійні моделі другого рівня для покращеної точності. Це дає змогу збільшити відгук (Recall) класифікації на **34,9%**, коефіцієнт детермінації (R^2) регресії - на **21%** у порівнянні із середніми метриками поодиноких моделей.
2. *Вдосконалено* архітектуру графової нейронної мережі для прогнозування виходу продукту хімічної реакції шляхом додавання інформації про учасників хімічного перетворення, а також молекулярні дескриптори, що дало змогу покращити R^2 до **0.86** та *RMSE* до **10.35** порівняно з найкращими відомими методами на тому ж наборі даних: R^2 0.81, *RMSE* 12.07.

3. *Вперше* розроблено метод генерування молекулярних структур, що дозволяє контролювати одну або більше властивостей генерованих молекул, а також має послідовний модуль виправлення помилок хімічної будови. Модель виправлення хімічних помилок підвищує вихід коректних молекулярних структур на **20%**, вихід унікальних молекул - на **67%** порівняно із моделлю-генератором без модулю виправлення помилок.
4. *Отримав подальший розвиток* метод редукції, шляхом спрощення групи нейронів за одну ітерацію та застосуванням методу перехресної перевірки. Вказані модифікації дозволяють уникнути пристосування моделі до даних у тестовій вибірці та видаляти надлишкові ваги за малу кількість ітерацій. На прикладних задачах по прогнозуванню молекулярної спорідненості та виходу продукту хімічної реакції показано, що редуктивне спрощення зменшує кількість активних ваг моделей на **86.88%** та **29.21%** і одночасно з цим підвищує коефіцієнт детермінації (R^2) відповідних моделей на **2.8%** та **15.43%**, порівняно із повними версіями аналогічних моделей.

Практичне значення одержаних результатів. Розроблено мікросервісну архітектуру системи створення молекул із заданими властивостями, що дає змогу здійснювати цільовий синтез молекул-кандидатів у лікарські речовини. Усі описані методи та моделі реалізовано та апробовано у розрахункових та практичних дослідженнях. Результати дисертаційної роботи впроваджено в ТЗОВ SoftServe, а також у ряді комерційних проектів.

Особистий внесок здобувача. Основні положення та результати дисертаційної роботи одержані автором самостійно. Особисто здобувачеві належать наступні наукові результати: метод створення молекулярних структур, метод прогнозування молекулярної спорідненості, метод прогнозування виходу продукту хімічної реакції, модифікація методу редуктивного спрощення.

Апробація результатів дисертації. Методом молекулярної динаміки у силовому полі OPLS-AA було встановлено, що згенерована молекула-кандидат ($C_7H_{16}O_3$) є фізико-хімічно стабільною, а її прогнозована розчинність у воді ($\approx 0,55$ моль/л) відповідає симуляційній

межі розчинності. Результати дисертаційної роботи доповідались на конференції Advances in Neural Information Processing Systems 32 (NeurIPS 2019) у Ванкувері, Канада. Також результати доповідались на семінарах кафедри систем штучного інтелекту «Національного університету «Львівська політехніка».

1 РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА ПІДХОДІВ МАШИННОГО НАВЧАННЯ ДО ПРОГНОЗУВАННЯ МОЛЕКУЛЯРНИХ ВЛАСТИВОСТЕЙ.

У цьому розділі проаналізовано існуючі методи та підходи машинного навчання до прогнозування молекулярної спорідненості, фактичного виходу продукту хімічної реакції та генерації молекулярних структур. За результатами огляду сформовано завдання дисертаційної роботи.

1.1 Аналіз методів машинного навчання для прогнозування молекулярної спорідненості.

Початкові етапи розробки лікарських речовин вимагають локалізації рецептору, що відповідає за розвиток хвороби, вивчення молекулярного механізму і наступний дизайн та випробування молекул-кандидатів. Після того як молекула-рецептор була ідентифікована, створюється список лігандів-кандидатів. Кращі молекули зі списку обираються за найвищою спорідненістю кандидату до цільового рецептора. Спорідненість характеризує силу взаємодії між цілями та лігандами. Вона може бути кількісно описана константою інгібування K_i . Чим менше значення K_i , тим сильніше ліганд зменшує біологічну активність рецептору, і тим сильніший лікувальний ефект може бути досягнутий меншою дозою ліків. Існує багато методів експериментального вимірювання K_i , але вони є затратними за часом, ресурсами та зусиллями. Тому перед експериментальними застосовуються обчислювальні підходи - для зменшення набору лабораторних випробувань шляхом ранжування та виключення кандидатів з низькою розрахунковою спорідненістю.

Одним з таких підходів є молекулярний докінг [28, 29, 30], у якому досліджуються можливі просторові взаєморозташування рецептору і ліганду та оптимізується конформація зчеплення структур. Однак, розрахована енергія докінгу не є величиною молекулярної спорідненості, оскільки конформації з низькою оціночною енергією звязування не завжди

співпадають з експериментальними [31]. Важливо відмітити, що докінг може постраждати від неточного виявлення лігандових точок/положень, або взагалі зчепити зовсім неактивні компоненти [32].

Разом із класичними симуляціями силових полів і докінгом, технології машинного навчання стали потужним інструментом в галузі віртуального відсіювання. Одне із перших досліджень у машинному навчанні для молекулярної спорідненості опубліковане King та ін. [33]. У цій роботі ефективність нейронних мереж прямого поширення, методу К-найближчих сусідів і дерев рішень порівнювалися на 200 лігандах та двох рецепторах. Список дескрипторів лігандів включає молекулярну масу, гнучкість, полярність, поляризацію, кількість донорів/акцепторів електронів, тощо. Автори [34] оцінили спорідненість лігандів, використавши метод опорних векторів. Їх набір даних складався з кількох сотень пар ліганд-рецептор. Було згенеровано більш ніж 500 молекулярних дескрипторів, проте лише 50 з них виявилися інформативними. Важливість вибору правильних параметрів також висвітлюється у дослідженні [35], де вивчається спорідненість білків з білками.

Концепція усередненого простору взаємодій, представлена у дослідженні [36]. Для багаторецепторної бінарної класифікації активних/неактивних лігандів запропоновано набір з 439 ознак: 107 описують рецептори, 166 параметрів є MACCS відбитками [37], що описують ліганди, решта 166 є відбитками, усередненими за лігандами до одного рецептора. В даному випадку було обрано метод Баєсових адитивних регресійних дерев. У наступних підходах використано метод опорних векторів, випадковий ліс, дерева рішень та логістичну регресію. Автори досягли точності в близько 95% для бінарної класифікації активних і неактивних лігандів. Схожа концепція простору, усередненого за взаємодією, також використовується у дослідженні Неск та співавторів [38]. Автори збудували декілька регресійних моделей для прогнозування молекулярної афінності використовуючи вектори параметризованих лігандів, що усереднюються для кожного рецептора.

Також варто згадати ряд досліджень [39, 40, 41, 42, 43, 44], що перевірялись на однакових наборах даних [45, 46], зокрема кіназах. Молекулярна спорідненість у KronRLS [39] визначається через

добуток Кронекера між матрицями подібностей пар ліганд-ліганд та рецептор-рецептор. Маючи подібності лігандів та подібності рецепторів, SimBoost [40] використовує градієнтний бустинг для прогнозування молекулярної спорідненості та піділяє про середньоквадратичну похибку в 0.28 для набору даних Дейвіса [45]. Глибока нейромережа DeepDTA [41] використовує FASTA рецепторів [47, 48] та SMILES лігандів [49, 50], кодує і доповнює відсутні символи нулями. Кодування проводиться двома окремими згортковими мережами, виходи з яких конкатенуються і передаються до повноз'єднаних шарів нейронної мережі, яка прогнозує молекулярну спорідненість. DeepDTA досягає середньоквадратичної похибки 0.26-0.66 (залежно від набору кодування) на наборі даних Дейвіса. WideDTA [42] використовує чотири текстові вводи: білкова послідовність, SMILES лігандів, білкові домени і мотиви, та максимальну спільну підструктуру слів для прогнозування афінності. Ці вводи передаються у чотири окремі рукави згорткової нейронної мережі, конкатенуються і передаються через набір повноз'єднаних шарів для прогнозування спорідненості. Модель демонструє середньоквадратичну похибку 0.26 на наборі даних Дейвіса. Підхід GraphDTA [43] також покладається на роздільні входи – один для ліганда і один для рецептора. Виводи рукавів також конкатенуються і регресуються. Ліганди представляються як графи з атомами в якості вузлів та зв'язками в якості ребер. Чотири реалізації графової нейромережі перевіряються для лігандової частини, у той час як рецептори кодуються з FASTA нотації і обробляються набором конволюційних мереж. Отримана похибка MSE коливається в районі від 0.23 до 0.25 pKd на наборі даних Дейвіса. Подібно до KronRLS, модель SimCNN-DTA [44], базується на хімічних подібностях. Зокрема, для пари ліганд-рецептор обчислюються два вектори: перший складається з подібностей Танімото між молекулярними відбитками лігандів, другий – з подібностей Сміта-Вотермана між FASTA послідовностями рецепторів. Зовнішній добуток цих двох векторів являє собою двовимірну матрицю, яка слугує входними даними для двовимірної конволюційної мережі для прогнозування молекулярних спорідненостей. SimCNN-DTA перевірялась на наборах даних Дейвіса і KIBA, і показала однакову та кращу продуктивність, ніж інші дослідження у цьому абзаці.

Дослідження [51] порівнює продуктивність низки методів машинного навчання (випадковий ліс, метор опорних векторів, Гаусовий процес, нейромережа прямого поширення) для прогнозування афінності на наборі даних PDBbind (v.2015) [52]. Для всіх розглянутих підходів рецептори і ліганди описувались набором структурних і фізико-хімічних параметрів, що формують єдиний вхідний вектор для кожної пари рецептор-ліганд. Цільовими змінними були значення K_i та K_d . Найкращих результатів вдалось досягнути за допомогою випадкового лісу.

Підхід DGraphDTA [53] використовує структурну інформацію молекул і білків. Будуються два графи: один для білкового рецептора, інший - для молекули ліганда, на яких проводиться регресія для прогнозування афінності. Цікаво, що білкові графи будуються з контактних матриць білків, які прогнозуються, використовуючи послідовності FASTA моделлю PconsC4 [54].

Jimenez та співавтори [55] запропонували Kdeep – тривимірну конволюційну нейромережу (3D-CNN) для прогнозування молекулярної афінності. Кожна пара білку з лігандом описується вокселізованим об'ємом у 24 Ангстреми представленням сайту звязування. Кожен воксель описаний вісьмома фармакофорними властивостями. Автори досягнули RMSE в 1.27 рК між експериментальним і прогнозованим значенням на базовому тестовому наборі PDBbind (v.2016) [52]. DeepAtom [56] - ще один 3D-CNN фреймворк, що кодує властивості, пов'язані зі звязуванням, за допомогою вокселізованих представлень. З набором даних Astex Diverse [57] в якості навчальної вибірки, DeepAtom досягає RMSE в 1.23 в одиницях рК на тестовій вибірці PDBbind (v.2016) [52].

Дослідження Kwon та співавторів [58] обговорює підхід для прогнозування спорідненості на основі ансамблів 3D-CNNів. Ансамбль має найменші метрики похибок з MAE значенням в 1.01 ккал/моль та RMSE в 1.29 ккал/моль. Використання ансамблю мереж покращило якість прогнозів на 0.1 ккал/моль порівняно з продуктивністю поодиноких мереж. Ансамбль моделей випадкового лісу, AdaBoost-Regressor, Gradient Boosting Regressor та мережі прямого поширення, розроблений Chen та співавторами [59], прогнозує спорідненість пептидних лігандів до кількох білків повязаних з пухлинами. Chen та співавтори [59] доповідають про

доволі високі коефіцієнти детермінації R^2 в 0.81/0.9 на навчальній/тестовій вибірці і, в якості додаткового кроку валідації, виконує ряд компютерних симуляцій для обраних пар білків-лігандів. Проте, вибрані ліганди не показали очікувану сильну з'язаність. Інтегрований підхід, що використовує зчеплення лігандів на кількох структурних ансамблях для відображення рецепторної гнучкості пропонується Schneider [60]. Підхід поєднує етап лігандного зчеплення з етапом прогнозування афінності для комплексу ліганд-рецептор, застосовуючи модель випадкового лісу. Для того, щоб покращити рішення, перевіряються різні набори молекулярних дескрипторів.

З часу своєї появи нейромережі-трансформери були успішно застосовані для відображення хімічних символів у представлення хімічних функцій і властивостей. У дослідженні Schwaller [61], моделі self-attention використовуються для прогнозування продуктів хімічних реакцій, сформульованих як проблема машинного перекладу SMILES-рядків реактивів та реагентів на продукти. Рауне [62] аналізує застосування BERT-моделі для вивчення контекстних представлень хімічних сполук для прогнозування токсичності, розчинності, подібності до ліків та синтетичної доступності. У роботі Rives [63] кодування білкових послідовностей вивчаються BERT-моделлю з широкомасштабного нерозміченого набору даних і ретельно вивчається на предмет кодування різноманітних білкових аспектів. Модель MT-DTI [64] поєднує здібності моделювати послідовності двох моделей: CNN для FASTA і self-attention механізми для SMILES, для задачі прогнозування афінності. Таке поєднання нейронних архітектур дозволило досягнути неперевершених результатів на вищезгаданих наборах даних KIBA і Дейвіса.

Mottaqi [65] описують пошук ефективних інгібіторів проти рецепторів, пов'язаних з SARS-CoV-2, за допомогою методів машинного навчання. Jin [66] та Gao [67] пропонують градієнтний бустинг для пошуку інгібіторів до протеази SARS-CoV-2 3CL серед вже затверджених FDA (Управління з продовольства і медикаментів США) ліків. Так як протеаза є єдиним рецептором у цьому дослідженні, немає потреби у його параметризації. З навчальною вибіркою у 314 інгібіторів, ліганди параметризуються усередненням трьох видів молекулярних відбитків. Дослідження Nand [68]

застосовує багатоетапний конвеєр, що поєднує класифікацію активності, фільтрацію лікувальної подібності, прогнозування зчеплення і звязної спорідненості для пошуку інгібіторів протеази. Оскільки розглядався підхід з одним рецептором, тільки ліганди були описані набором дескрипторів і як вхідні дані. Обрані інгібітори перевірялися у розрахунковому експерименті за методом молекулярної динаміки. Santana [69] виконав скринінг бази даних ChEMBL [70] рекурентною нейромережою для пошуку інгібіторів проти протеази. Молекули, які були визначені як активні, були в подальшому проаналізовані з використанням молекулярного докінгу. Kowalewski та Ray [71] запропонували конвеєр для визначення ліків-кандидатів проти кількох рецепторів, повязаних з SARS-CoV-2, із особливою увагою на виборі параметрів лігандів. Натренований конвеєр в подальшому використовувався для ранжування списку тисяч відомих ліків та мільйонів доступних до синтезу молекул за зв'язною спорідненістю, токсичністю і леткістю. Модель MT-DTI [64, 72] застосовувалася для прогнозування афінності відомих противірусних ліків для шести рецепторів, повязаних з SARS-CoV-2. У роботі Kadioglu [73], відомі ліки та доступні до синтезу молекули перевіряються на здатність взаємодіяти з трьома рецепторами (шипи, капсиди і трансферази) засобами AutoDock Vina [74, 75] і підходами машинного навчання.

Огляд Ellingson [76] освітлює популярні підходи машинного навчання і перешкоди, що постають у вивченні молекулярної афінності. Окрім типових проблем з представленням, які згадуються вище, також обговорюються складності з доступністю та узгодженістю даних. Якість прогнозів спорідненості може постраждати від неузгоджених даних, на які вплинув експериментальний шум. Важливо, що деякі з обговорених досліджень фокусуються на одному рецепторі, тому в них не потрібно параметризувати рецептори. Крім того, поєднання моделей у ансамблі допомагає розширити область прогнозування через поєднання рішень для вужчого кола завдань.

1.2 Аналіз методів машинного навчання для прогнозування фактичного виходу продукту хімічної реакції.

Поширення технологій машинного навчання спричинило революцію у багатьох галузях і зараз переживає сплеск застосування в обчислювальній хімії. Найбільш поширеними задачами у цій області є генерація молекул-кандидатів, ретросинтез, оптимізація умов хімічних реакцій і прогнозування їх результатів. Такі моделі машинного навчання як варіаційні автокодувальники [77, 78], генеративні змагальні мережі [79, 80] та рекурентні нейронні мережі [81, 82] продемонстрували перспективи дизайну *de novo* ліків. Останні досягнення у глибокому підкріплюючому навчанні (Deep Reinforcement Learning, DRL) і моделях-трансформерах вивели технологію генерації молекул на якісно новий рівень [83, 84, 85, 86, 87, 88, 89, 90].

1.2.1 Реакційна здатність.

Завдання прогнозування фактичного виходу продукту належить до завдань оцінки хімічної реакційної здатності. Один із перших підходів був розроблений Kite та ін. [91], де автори використовували нейронні мережі для прогнозування селективності бажаних продуктів реакції. Досліджуваним класом реакції було окисне дегідрування етилбензолу. Наступна спроба прогнозувати «успішність реакції» була здійснена Raschiglia та ін. [92] — автори створили модель на основі методу опорних векторів (SVM) для прогнозування умов реакції, які забезпечать найвищий рівень успішності реакції. Вони отримали точність 78% для всіх типів реакцій у наборі даних і 79% для реакцій Ванадію та Селену.

1.2.2 Хімічні дескриптори.

Вибір ефективних дескрипторів для молекул та реакцій [93], [94], [95] завжди становив серйозну проблему. Ефективність дескрипторів реакцій для лінійних моделей досліджувалась у ряді робіт [96],[97],[98]. Продемонстровано високий ступінь залежності результатів моделей машинного навчання від обраного виду молекулярних представлень.

Yada [99] використовував регресію LASSO для вибору електронних і вібраційних параметрів каталізаторів реакції (фосфонових кислот). Значення параметрів були зібрані за допомогою моделювання DFT. Автори повідомили про точність 26% RMSE прогнозування виходу продукту для одного класу реакції (епоксидування алкенів, що каталізується вольфрамом). Це значно вища похибка, ніж інші дослідження для одного типу хімічної реакції. Наприклад, Doyle [100], Euke [101] і Fu [102] досягнули RMSE 7,4%, 10% і 9% відповідно. У роботі Doyle та ін. [100] використано модель випадкового лісу для вибору високопродуктивних умов для реакцій дезоксифторування сульфонілфториду. Модель досягла 7,4% середньоквадратичної помилки (RMSE) на тестовому наборі даних. Автори дійшли висновку, що для покращення результатів необхідне кодування додаткових умов реакції.

1.2.3 Публічно доступні набори даних.

Більшість підходів до прогнозування хімічного виходу зосереджені на одному класі реакції. Найбільш використовуваними наборами даних одного типу реакції є Бухвальда-Гартвіга [103] і Сузукі-Міяури [104]. Вони містять записи хімічних реакцій перехресного сполучення, що каталізуються Pd, із відповідними експериментальними виходами.

Набір даних реакції Бухвальда-Гартвіга. У роботі, що представляє набір даних реакції Бухвальда-Гартвіга [103], автори розраховали атомні, молекулярні та вібраційні дескриптори для компонентів реакції, щоб побудувати векторне представлення реакції. Ці вектори використовувалися як вхідні дані, а фактичний вихід продукту реакції використовувався як цільова змінна для випадкового лісу (RF). Модель RF досягла вищого коефіцієнту детермінації (R^2 0,92), ніж лінійна регресія та інші методи, як-от k-найближчих сусідів, SVM, узагальнена лінійна модель (Generalized Linear Model, GLM), мережа Байєса (Bayesian Network, BN) (усі з R^2 0,67) і однорівневий нейронна мережа (R^2 0,87). Однак через півроку Чуанг і Кейзер [105] продемонстрували, що результати випадкового лісу були однаковими як для випадкових ознак, так і для представлень реакцій, які використовував [103]. Робота Sandfort та ін. [106] перевершила

результати попередньої групи на тому ж наборі даних, досягнувши R^2 0,93. Автори додали численні ознаки, що характеризують реагенти, і навчили модель випадкового лісу на конкатенованих фінгерпринтах молекул-учасників реакції. Дослідження Doyle та ін. [107] включало прогнозування виходу для реакцій арилбромідів у каталізованому Нікелем перехресному сполученні. Набір реакцій, що містить арилброміди як субстрати, був отриманий від Reaxys [108] і складався з 2600 зразків. За висновками ряду експериментів було обрано ознаки згенеровані DFT. Потім було виконано кластеризацію цих векторних зображень. Ідентифікувавши 15 кластерів у даних, автори використовували центральні молекули кожної групи як навчальний набір даних для узагальненої адитивної моделі, використовуючи електронегативність арилбромідів як змінну предиктора. Регресійна модель досягла RMSE 19% на тестовій підмножині із 37 зразків.

Набір даних реакції Сузукі-Міяра. Автори [109] вивчали здатність нейронних мереж до узагальнення прогнозів виходу продукту реакції перехресного зв'язку Сузукі-Міяури. Їхня повнозв'язна нейронна мережа, навчена на компонентах реакції закодованих one-hot векторами, досягла 11% RMSE на тестовому наборі. Еуке та ін. [101] використовували відбитки Моргана реагентів і продуктів. Цей спосіб параметризації молекул дозволив їм досягти кращих результатів – 10% RMSE на десятикратній перехресній перевірці на наборі даних Сузукі-Міяури та реакції 3-бромпіридину.

Fu та ін. [102] повідомили про RMSE 8,994% для завдання вибору найефективніших каталізаторів і умов реакції для реакцій Сузукі-Міяра. Автори використали набір даних Jansen та ін. [110], який у десять разів менший, ніж Perera та ін. [104]. Fu та ін. навчили повністю пов'язану нейронну мережу з реакційними сполуками, представленими комбінацією молекулярних і квантово-механічних дескрипторів.

Одне з найповніших досліджень Дойла та ін. [111] оцінювало продуктивність різних функцій і керованих методів машинного навчання – лінійної регресії, узагальненої лінійної моделі, опорної векторної регресії, k-найближчих сусідів, випадкового лісу, XGBoost і нейронні мережі прямого зв'язку – про перехресне сполучення Сузукі-Міяури, дезоксифторування

та реакції амінування Бухвальда-Гартвіга, каталізовані Ni. Автори використовували моделювання функціоналу густини (Density-Functional Theory, DFT), щоб отримати характеристики субстратів, каталізаторів і реагентів. Для розрахунку похибки була використана методологія перехресної перевірки без однієї молекули. Залежно від методу визначення було досягнуто RMSE від 5% до 25% на наборі даних Бухвальда-Хартвіга та від 9% до 41% на наборі даних дезоксифторування. Автори прийшли до висновку, що продуктивність моделей сильно залежить від способу представлення функцій.

Набори даних із кількома класами реакцій. Кілька робіт вивчали можливість узагальнення прогнозування виходу продуктів для кількох класів реакції одночасно. Одну з перших таких спроб здійснили Skocaszynski та ін. [112]. Автори оцінили методи ML, включаючи SVM, нейронні мережі та випадковий ліс, на даних Reaxys [108], які включали 1 мільйон реакцій із відповідними фактичними виходами продукту. Близько 400 дескрипторів RDKit [113] та такі експериментальні параметри як розчинники та температура, а також бітові відбитки Моргана, було протестовано як ознаки для навчання моделей ML. Незалежно від застосованого методу ML, кількості рядків у навчальному наборі, а також природи та кількості дескрипторів для навчання моделі, точність бінарного прогнозування виходу продукту була лише $65 \pm 5\%$.

1.2.4 Моделі-трансформери.

Натхненні успіхом застосування трансформерних нейронних мереж у хемінформатиці [114], Шваллер та ін. [115] провели ряд прикладних досліджень трансформер-подібних нейронних мереж у хімії [116, 61, 117, 117]. Нещодавно ця група вчених запропонувала BERT для прогнозування виходу продукту хімічної реакції [118]. Автори поєднали представлення двонаправленого кодувальника (BERT) [119] із шаром регресії, щоб прогнозувати чисельні виходи трьох наборів реакцій, використовуючи лише текстове представлення (SMILES) реакції. Перед навчанням моделі на цільових даних енкодер BERT пройшов попереднє навчання на наборі даних реакції Pistachio [120]. Видатні оцінки R^2 0,956 і 0,81 були

досягнуті на двох однокласових наборах реакцій (Бухвальда-Хартвіга та Сузукі-Міяури). Однак, результати на наборі даних з кількома типами реакцій - USPTO [121], зібрані шляхом аналізу текстових записів патентів Сполучених Штатів, були значно гіршими: R^2 0,388. Автори пояснюють такі результати помилками у записах у наборі даних USPTO.

1.2.5 Графові нейронні мережі.

Граф — це природний спосіб кодування молекул, де вузли представляють атоми, а ребра — хімічні зв'язки. Нейронні мережі графів можуть апроксимувати специфічні для завдання молекулярні властивості за допомогою диференційованих параметрів вузлів і ребер [122]. Завдяки прийнятній параметризації молекул графові нейронні мережі (GNN) [123, 124, 125] стали перспективним методом машинного навчання для хімічних досліджень. Повідомляється, що GNN дозволили досягти видатних результатів у прогнозуванні [126] і молекулярних [127] властивостей, ЯМР-хімічного зсуву [128] та визначення взаємодії мішень-ліганд [129, 130]. Здобута популярність спонукала до розробки автоматизованої платформи [131] для швидкої оцінки GNN і порівняння з іншими підходами ML. Saebi *et al.* [132] застосував GNN із вбудованими блоками уваги для прогнозування значень фактичного виходу продукту на наборах даних реакцій Сузукі-Міяури та Бухвальда-Гартвіга. Автори об'єднали молекулярні (молекулярна маса, НОМО/LUMO тощо), атомні (електростатичний заряд, зсув ЯМР) і реакції (температура, об'єм тощо). На момент написання цієї роботи їх результати були недоступні.

Впровадження нейронних мереж спрямованої передачі повідомлень (D-MPNN) у хіміоінформатичну область є суттєвим у контексті цієї роботи. Поряд із численними роботами з модифікаціями та вдосконаленнями базової архітектури GNN [133, 134, 135, 136], Yang та ін. успішно застосували D-MPNN [137] до молекулярних завдання прогнозування властивості [138]. Grambow [139] розробив D-MPNN для прогнозування енергії активації хімічних реакцій, використовуючи графічне представлення реагентів і продуктів. D-MPNN створює навчену модель молекули, рекурсивно передаючи інформацію між елементами графа за допомогою повідомлень, пов'язаних із спрямованими ребрами

та зв'язками. Автори закодували хімічну реакцію як суму різниць графіків між продуктами та реагентами. Це представлення перегрупувань атомів було використано для оцінки енергії активації. Автори досягли RMSE $3,4 \pm 0,3$ ккал/моль на наборі даних, який охоплює діапазон енергії активації 200 ккал/моль.

1.3 Аналіз методів машинного навчання для генерації молекулярних структур.

Останні досягнення у машинному навчанні дозволяють ефективно вирішувати численні проблеми від наближення квантових хвильових функцій до прогнозування хімічних властивостей, фазових переходів і часової динаміки [140, 141, 142, 143, 144, 145, 146, 147, 148].

1.3.1 Повнозв'язні нейронні мережі.

На молекулярному рівні DNN використовуються для апроксимації квантово-механічних обчислень [141, 149], декомпозиції енергії кластерів або прогнозування наступного кроку молекулярної динаміки замість традиційних ресурсоемких процедур [150, 151, 152, 153, 154, 155]. Нещодавно, декілька симуляційних систем включили ці підходи до своїх обчислювальних інструментів [156, 157, 158, 159, 160, 161]. DNN також використовуються для прогнозування кількісних фізико-хімічних та біологічних властивостей за хімічною будовою сполук [22]. За такими моделями історично закріпилася англійська назва Quantitative Structure-Activity Relationship (QSAR). Серед модельованих властивостей можна згадати розчинність у воді або органічних розчинниках, температуру плавлення, енергії сольватації тощо [162, 163, 164]. Здатність ML пов'язувати структуру речовини із властивостями дає змогу оцінювати успішність молекул-кандидатів за показниками ADME (абсорбція, розподіл, метаболізм, екскреція) або ADMET (якщо також враховується токсичність). У цьому випадку акцент зосереджений на таких властивостях як спорідненість до рецепторів, токсичність та швидкість біологічного розпаду [165, 166, 167, 168, 169, 170, 171, 172, 173]. Своєрідним «золотим стандартом» у скринінгу схожості на ліки є так зване правило

п'яти, запропоноване Ліпінським та ін. [174], та його близькі варіації [175, 176, 177, 178], що дозволяє фармакологічним компаніям значно скоротити кількість молекул-кандидатів на ранніх стадіях розробки ліків.

1.3.2 Згорткові нейронні мережі.

У методах ML, що мають справу з молекулярними структурами, вирішальним кроком є ефективне представлення структурних даних. У вищезгаданих підходах використовуються різні формати вхідних даних: молекулярні графи, відбитки, дескриптори та їхні комбінації [169], позначення друкованими символами (наприклад, SMILES – Simplified Molecular-Input Line-Entry System). Останній спосіб представлення молекул уможливорює застосування методів обробки природної мови до проблем хімії, включаючи генерацію нових сполук [179, 180]. У роботах [181, 182] рядки SMILES перетворюються на двовимірні зображення, а потім передаються в CNN. Подібний підхід до використання зображень 2D-структур як вхідних даних також представлений в роботі [183].

1.3.3 Графові нейронні мережі.

Методи, засновані на графових нейронних мережах [184, 185], використовують графове представлення молекул. Таке представлення є природним вибором для вивчення молекулярних структур, взаємодій та синтезу [186]. Згорткові графові нейронні мережі та молекулярні графи використовуються для прогнозування розчинності, токсичності та інших властивостей сполук [186, 187]. У роботі [188] поєднано графові представлення із змагальним навчанням (Adversarial Training) та навчанням з підкріпленням для генерування молекул із бажаними властивостями. Графові нейронні мережі використовуються для прогнозування поверхні білка [189]. У дослідженні Zitnik та ін. [190] графові згорткові мережі використовуються для прогнозування можливих побічних ефектів ліків. Пропонується також генеративна мережа MolGAN [191] для створення молекулярних графів.

1.3.4 Автокодувальники та генеративні змагальні нейронні мережі.

Останні досягнення глибинного навчання значною мірою стосуються різних застосувань генеративних змагальних мереж (англ. GAN - Generative Adversarial Network) та інших глибоких генеративних моделей, здатних генерувати або реконструювати дані із заданого розподілу [192, 193]. У контексті молекулярних даних це відкриває шлях до синтезу нових структур із заданими властивостями (див., наприклад, огляд Jørgensen та ін. [194]). Автокодувальник (англ. autoencoder або AE) та варіаційний автокодувальник (англ. variational autoencoder або VAE) використовуються для відображення дискретних рядків SMILES у безперервному просторі [183, 195]. Вибір векторів-екземплярів у такому просторі та наступне їх декодування назад у рядки SMILES дозволяє отримувати нові унікальні структури. Різні DNN моделі були запропоновані та порівняні для підвищення якості векторних представлень та зменшення помилки реконструкції [194, 196, 197]. Автори [198] тестували різні архітектури GAN'ів для генерації молекул та зворотного відображення QSAR, відбором нових структур із застосуванням обмежень біологічної активності. Детальне обговорення проблеми «хімічного простору» та реконструкції молекул на основі його векторів представлено в нещодавньому дослідженні Б'єррума та Саттарова [199]. Модель на основі GAN від Guimaraes та ін. [200] вчиться генерувати молекули у представленні рядків SMILES, оптимізуючи їх властивості до набору хімічних показників.

1.3.5 Виправлення хімічних помилок.

Незважаючи на безперервну природу латентного векторного простору та нескінченні можливості вибору довільних векторів, не всі вибрані вектори відповідають хімічно коректним рядкам SMILES. Деякі з цих векторів можуть декодуватися у хімічно неправильні SMILES, тоді як інші (навіть «граматично» правильні) можуть відповідати нестабільним хімічним сполукам. Успішну спробу вирішити цю проблему було зроблено шляхом заміни звичайного VAE на Grammar VAE [180]. Іншим напрямком

вирішення проблеми дотримання "хімічної" правильності представлень є збагачення граматики SMILES контекстними атрибутами [201].

1.4 Актуальність та постановка завдань.

1.4.1 Прогнозування молекулярної спорідненості.

На початковому етапі розробки лікарських речовин локалізують рецептор - високомолекулярну біомолекулу, що відповідає за розвиток та перебіг хвороби. Після цього вивчають молекулярний механізм захворювання - розуміння механізму дозволяє розпочати етап розрахункового (*in silico*) дизайну та випробовування молекул-інгібіторів рецептору. Створюється список лігандів-кандидатів на лікарську речовину - тобто хімічну молекулу, яка впливатиме на цільовий рецептор таким чином, що виліковуватиме хворобу або її прояви. Кращі молекули зі списку кандидатів обираються за найвищою селективністю кандидату до цільового рецептора. Селективність - це властивість молекули лікарської речовини зв'язуватися із цільовим рецептором (висока спорідненість) та не зв'язуватися із іншими рецепторами людського організму (низька спорідненість), оскільки нецільові зв'язування спричинюють побічні ефекти. Спорідненість характеризує силу взаємодії між рецепторами та лігандами. Вона може бути кількісно описана константою інгібіювання K_i . Чим менше значення K_i , тим сильніше ліганд пригнічує біологічну активність рецептору, і тим сильніший лікувальний ефект може бути досягнутий меншою дозою ліків. Існує багато методів експериментального вимірювання K_i , але вони є затратними за часом, ресурсами та зусиллями. Тому перед експериментальними застосовуються обчислювальні підходи для зменшення набору лабораторних випробувань шляхом ранжування та виключення кандидатів з низькою розрахунковою спорідненістю. Поряд із класичними технологіями (молекулярна та квантова динаміка, докінг), технології машинного навчання стають дедалі потужнішим інструментом в галузі віртуального скринінгу. Таким чином, перше завдання дисертації - **створення технології прогнозування молекулярної спорідненості.**

1.4.2 Прогнозування виходу продукту хімічної реакції.

Ефективний синтез завжди був важливою проблемою у хімії через її величезний вплив на різні галузі: від сільського господарства та фармацевтики до енергетики та автомобілебудування. Інколи теоретично можлива реакція на практиці не відбувається, а очікувані виходи продукту виявляються нижчими через неповне перетворення реагентів, побічні реакції, складну очистку продуктів тощо. Вихід цільового продукту із поправкою на такі чинники називають фактичний (практичний) вихід. Переоцінити здатність надійно прогнозувати фактичні виходи продуктів - особливо для нових реакцій - важко. Технологія прогнозування кількісного фактичного виходу продукту хімічної реакції суттєво підвищить рентабельність хімічної індустрії, оскільки надасть хімікам здатність обирати високопродуктивні реакції на етапі планування синтезу, заощаджуючи час, зусилля та реагенти для отримання бажаного продукту. Отож, другим завданням роботи стало *створення технології для вибору високопродуктивних шляхів органічного синтезу молекул-кандидатів у лікарські речовини.*

1.4.3 Генерація молекулярних структур із заданими властивостями.

На етапі дизайну нових біологічно активних речовин важливо враховувати не лише їх хімічну коректність, але і очікувані фізико-хімічні властивості. Серед фізико-хімічних властивостей важливо згадати розчинність у полярних та неполярних розчинниках (LogP та LogS), топологічну полярну площу поверхні (Topological Polar Surface Area, або TPSA) та молекулярну масу (Molecular Weight, або MW). Успіх обраної молекули-кандидата на подальших доклінічних та клінічних випробуваннях напряму залежить від правильного підбору вказаних параметрів та точності їх прогнозування. Тому третім завданням роботи стало *створення технології для дизайну молекулярних структур із заданими фізико-хімічними властивостями.*

1.4.4 Платформа для розробки нових лікарських речовин.

Однак, одних лише правильно підібраних фізико-хімічних властивостей молекули-кандидата недостатньо, оскільки для початку доклінічних та клінічних досліджень цю речовину потрібно синтезувати, очистити та виділити у лабораторії. Відомо багато випадків, коли перспективні молекули-кандидати виявлялися непридатними до синтезу або їх вихід у реакціях був надто низьким через складнощі з очисткою, виділенням, наявністю побічних продуктів, стеричних перешкод тощо. Відповідно, до описаних вище властивостей важливо додати очікувану синтетичну доступність та вихід продукту у хімічній реакції синтезу певної молекули-кандидата.

Іншим критично важливим чинником успішності молекули-кандидата на пре- та клінічних дослідження є селективність до цільового рецептору. Селективність - це висока спорідненість молекули-інгібітора до цільового рецептору та одночасно низька спорідненість до усіх інших. Чим більш селективним є інгібітор, тим менше побічних ефектів (через взаємодії з нецільовими рецепторами) буде виявлено на клінічних дослідженнях і тим безпечнішим буде створений лікарський препарат. Селективність розраховується через оцінку молекулярної афінності молекули до цільового та ряду побічних рецепторів. Якщо молекула має високу спорідненість до першого, та низьку - до других, вона вважається високоселективним інгібітором, а її цінність - як з точки зору користі так і пряма грошова вартість - зростає.

Отже, високоефективна технологія дизайну нових лікарських речовин повинна комплексно враховувати усі вищенаведені фактори успішності молекул-кандидатів. У цей спосіб запропоновані молекули матимуть найвищий коефіцієнт успіху на клінічних випробуваннях. Тому четвертим завданням роботи стало **створення платформи для дизайну молекул**, яка поєднуватиме генерацію кандидатів та комплексний контроль їхніх фізико-хімічних (синтетична доступність, молекулярна маса, ступінь розчинності, топологічна полярна площа поверхні, очікуваний вихід продукту у реакції синтезу) та біологічних (молекулярна спорідненість та селективність до рецептору) властивостей.

1.5 Висновки.

У цьому розділі було проаналізовано методи машинного навчання для прогнозування молекулярної спорідненості, фактичного виходу продукту хімічної реакції та генерації молекулярних структур. За результатами огляду матеріалу було сформовано функціональні вимоги до платформи для розробки нових лікарських речовин. Результатом розділу є розв'язання задачі дослідження №1.

2 РОЗДІЛ 2. МЕТОД ПРОГНОЗУВАННЯ МОЛЕКУЛЯРНОЇ СПОРІДНЕНОСТІ.

У цьому розділі описано новітню технологію для передбачення молекулярної спорідненості між білковими рецепторами та низькомолекулярними лігандами: метод мета-навчання на ансамблях моделей машинного навчання. Результати розділу опубліковано у працях автора [5, 3, 4, 12, 9].

2.1 Підготовка даних.

У публічному доступі є небагато наборів даних які містять експериментальні константи інгібування K_i та для класифікації активних/неактивних лігандів до різних рецепторів. K_i - це концентрація ліганду в розчині, необхідна для інгібування функції рецептора-мішені наполовину. K_i відображає, наскільки сильно молекула-інгібітор пригнічує біологічну функцію молекули-рецептора: чим нижче K_i , тим сильнішим (більш активним) є інгібітор, і навпаки. Не існує строгого порогу для константи інгібування, яка розрізняє активні та неактивні ліганди, однак K_i в 10 000 нмоль часто використовується як такий роздільник [40, 42, 71, 44, 202]. Оскільки досліджуваний метод складається з ансамблів класифікації та регресії, ми підготували два набори даних для навчання відповідних моделей. Класифікаційні та регресійні набори даних були об'єднані з трьох баз даних: BindingDB [203], DUD-E [204] і ChEMBL [70]. Ліганди були представлені за допомогою рядка SMILES, під час попередньої обробки вони були канонізовані за допомогою RDKit [205] із видаленням інформації про ізометрію молекул.

2.1.1 Підготовка класифікаційного набору даних.

Початковий розмір необробленого набору даних для класифікації становив 27497 записів. Було канонізовано SMILES, видалено дублікати та записи з помилками. Ліганд вважався неактивним, якщо K_i для нього мав значення $>10\ 000$ нмоль. Якщо для одного ліганду в різних базах зустрічалися суперечливі записи, клас зв'язування визначався за більшістю

одностаїних записів. Якщо більшість неможливо було встановити, ліганд вилучався з набору даних. Під час навчання ми використовували п'ятиразову валідацію моделей класифікації та зберігали сталий розподіл активних та неактивних лігандів у кожній навчальній і тестовій вибірці. Для видалення аномалій була розрахована молекулярна маса усіх лігандів. Видалення аномалій проводилося за методом Д. Тюки, який заснований на розрахунку міжквартильної відстані (IQR) за формулою 2.1 і наступного визначення меж м'яких викидів за формулами 2.2 та 2.3:

$$IQR = Q3 - Q1 \quad (2.1)$$

$$LIF = Q1 - 1.5 * IQR \quad (2.2)$$

$$UIF = Q13 - 1.5 * IQR \quad (2.3)$$

де LIF - нижня внутрішня межа для виявлення м'яких викидів; UIF - верхня внутрішня межа для виявлення м'яких викидів; $Q1$ - значення першого квартиля; $Q3$ - значення третього квартиля; IQR - міжквартильна відстань.

2.1.2 Підготовка регресійного набору даних.

Початковий розмір набору даних для регресії становив 4161 записів. До набору даних було включено лише записи з точними значеннями K_i . Якщо для одного ліганду було кілька різних значень K_i , ми обирали найбільше значення - як негативний сценарій. Додаткова умова при створенні вибірки даних для регресії впливає з конструкції конвеєра: регресійний ансамбль передбачає K_i , лише якщо класифікаційний стек моделей визначив активний ліганд. Тому для задачі регресії ми обирали значення K_i лише в межах «активного» діапазону та опускали зразки з високими K_i . В результаті, мітки в наборі даних для регресії знаходяться у діапазоні K_i [0 : 30 000 нМ]. Невелике перевищення діапазону K_i поза 10 000 нмоль мало на меті дати змогу регресійним моделям не обмежуватись лише обмеженою вибіркою найактивніших лігандів, а й передбачати K_i для слабких інгібіторів (10 000 - 30 000 нМ). Оскільки похибка

вимірювання K_i зростає прямопропорційно її значенню, ми працювали з десятковим логарифмом константи - $\log_{10}K_i$. Таке перетворення допомогло збалансувати внесок помилок у функцію втрат у різних діапазонах концентрації на наблизити розподіл цільових значень до нормального. Гістограма, що показує розподіл зразків із значеннями $\log_{10}K_i$, показана на рис. 5. Набір даних регресії також був розділений на п'ять частин для перехресної валідації. Так само як і для класифікаційного набору, з даних було видалено дублікати. Аномалії за методом Д. Тюки із використанням двох параметрів: молекулярної маси та значення $\log_{10}K_i$.

2.1.3 Результати підготовки даних.

Після підготовки, набір даних для класифікації складався з 26 808 лігандів, що відносились до двох класів активності по відношенню до людського тромбіну: 3565 активних (13,3%) та 23243 не активних (86,7%). Набір даних для регресії складався із 3940 унікальних лігандів тромбіну із відповідними значеннями констант інгібування (K_i). Відповідно до описаних вище правил, 3225 (81,6%) з них були активними та 726 (18,4%) - неактивними. Розподіли активних та неактивних лігандів можна бачити на рис. 2.1. Відзначимо, що набори даних для класифікації та регресії мають подібну до нормального розподілу форму та близькі середні значення молекулярних мас - 443 та 481 відповідно. Розподіл класифікаційних даних в основному локалізовано в межах від 300 до 600 а.о.м., тоді як розподіл даних для регресії демонструє більш широкий розкид - від 300 до 700. Середнє значення $\log_{10}K_i$ становило 3 та розкид від -2 до 6.

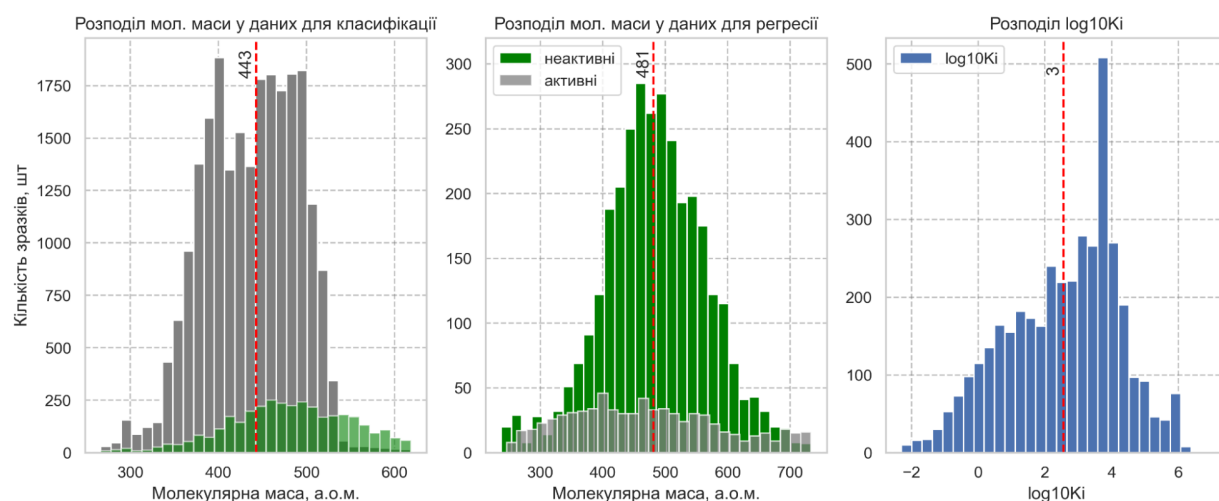


Рис. 2.1. Розподіли молекулярної маси лігандів у наборах даних для класифікації (зліва) та регресії (посередині) після видалення дублікату та аномальних точок. Активні ліганди показано зеленим, неактивні - сірим кольором. Розподіл значень $\log_{10}K_i$ у для регресії показано справа. Червоними вертикальними лініями показані середні значення величин, відкладених по горизонтальній осі координат [5].

2.1.4 Розподіл функціональних груп.

Для кращого розуміння даних, було проаналізовано частоту зустрічання спільних молекулярних фрагментів серед активних та неактивних молекул. Для цього ми скористалися методом, що визначає різноманітні функціональні групи, а також хімічні групи, утворені лише атомами вуглецю, гетероцикли, ароматичні структури та поодинокі атоми [206]. Аналіз мав на меті встановлення можливих структурних причин, що визначають міру активності ліганду, із точки зору хімічної будови речовини та розподілу даних.

У класифікаційному наборі даних розраховані частоти зустрічання молекулярних фрагментів у активних лігандах та співставлені із такими у неактивних молекулах. Результат порівняння для 20 фрагментів, що зустрічаються серед активних лігандів найбільш часто, можна бачити на рис. 2.2. Активні ліганди відзначаються більшим вмістом вторинного вуглецю (C020), бензольних кілець (c1ccccc1) та пептидних груп (NC=O). Неактивні ж молекули мають більше метильних груп (C010), етильних фрагментів (CC) та загальним вмістом атому Оксигену (O).

У регресійному наборі даних спочатку було виділено терцилі за цільовою метрикою ($\log_{10}K_i$). Концептуально, терцилі відповідали уявному класу активності лігандів: q1 - “найактивніші”, q2 - “активні”, q3 - “помірно активні”. Розраховані межі терцилів становили -2.3, 1.9, 3.6, 7. Частоти зустрічання фрагментів були розраховані для усіх молекул. Потім, 20 найчастіших фрагментів першого терцилю (“найактивніші” ліганди) були співставлені із частотами цих фрагментів у двох інших терцилях. Результат можна побачити на рис. 2.2. Легко побачити, що найактивніші ліганди вирізняються високим вмістом вторинного (C020) та третинного (C030) вуглецю, а також довшими нерозгалуженими алкільними елементами (ССС) - на противагу більшому вмісту коротших (СС) у менш активних лігандах. Цікаво, що фрагменти бензолу (c1ccccc1) сприяють зменшенню активності лігандів - їхній вміст у q2 та q3 більший, ніж у q1. Також варто відзначити, що у регресійному наборі даних прослідковується та ж тенденція, що і у класифікаційному: сполуки із вищим вмістом Оксигену (О) виявляють нижчу активність.

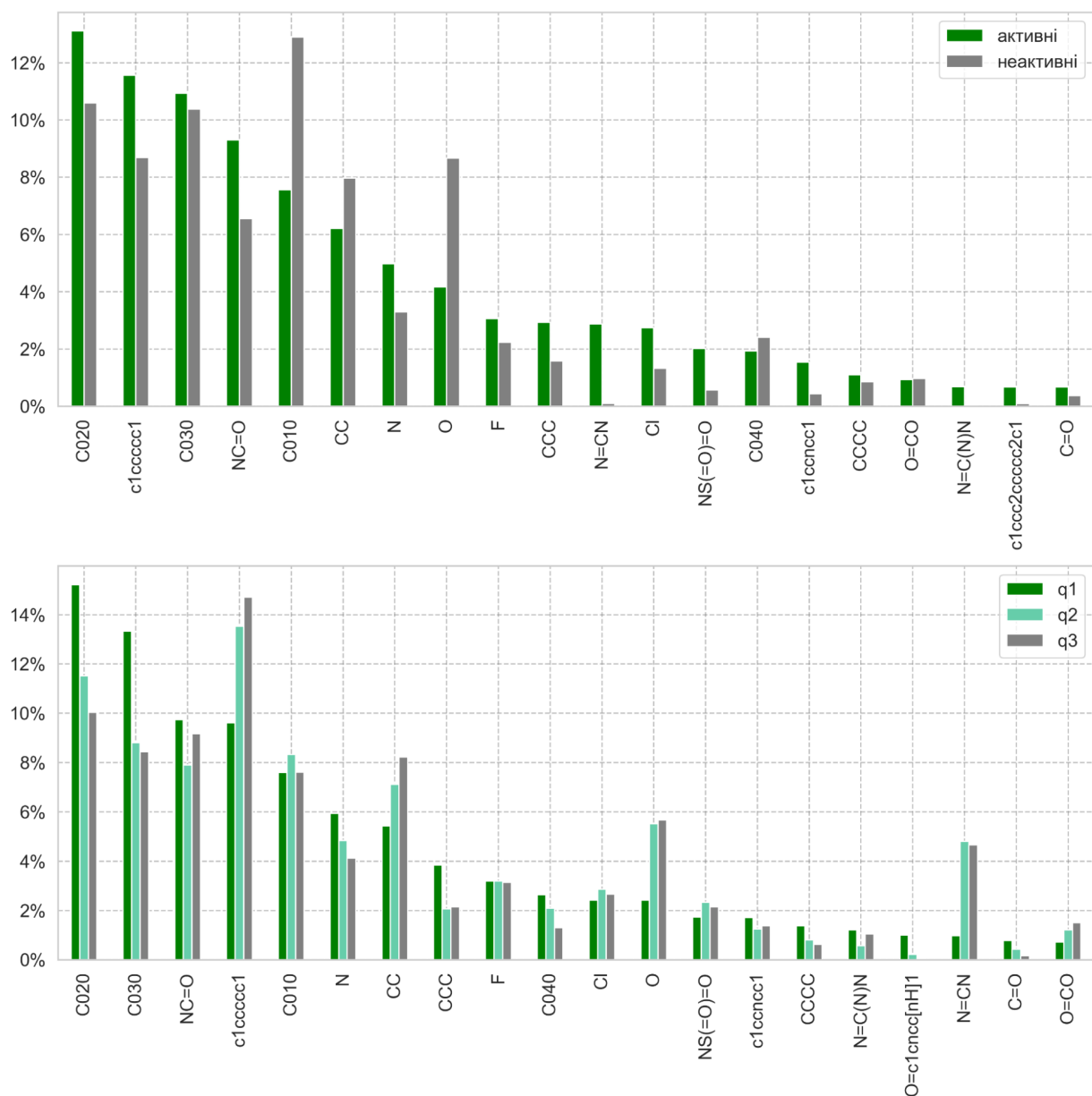


Рис. 2.2. Зверху: нормалізована частота зустрічання молекулярних фрагментів у активних та неактивних молекулах, що використовувалися для тренування класифікаційних моделей. Знизу: Нормалізована частота зустрічання молекулярних фрагментів у сполуках, що належать до трьох терцилів регресійних даних: q1 - K_i до 100 нмоль, q2 - K_i від 100 до 4000 нмоль, q3 - K_i 4000 нмоль та більше [5].

2.1.5 Інженерія ознак.

Ліганди кодувалися трьома способами. Перший спосіб – використання відбитків ECFP4 – використовується в підходах SVM, RF, СВ та нейронних мережах прямого поширення. Ці відбитки створювалися за

допомогою бібліотеки RDKit. Другий спосіб - представлення лігандів через молекулярні графи, які використовуються в підході з графовими нейронними мережами. Третій спосіб - токенизація текстових позначень SMILES при використанні BERT. Жоден із цих підходів не використовує просторові координати лігандів, що є важливою перевагою у порівнянні із підходами, що використовують координати молекул, оскільки, по-перше, дозволяє прогнозувати спорідненість для нових лігандів із невідомими просторовими структурами; по-друге, у такий спосіб із даних виключається неминуча експериментальна похибка вимірювань координат, що залежить від методу та приладу, яким проводиться вимірювання.

Перш ніж перейти до наступного розділу, варто більш детально обговорити молекулярні графи. Очікується, що визначальні взаємодії між атомами в молекулі можуть бути змодельовані за допомогою графу, і така математична модель може описати функції та властивості молекули. Щоб створити молекулярний граф, необхідно закодувати ознаки вузлів (атоми) та ребер (зв'язки). Для цієї мети ми використовували Weave atom and bond featurizer [186], який генерує дев'ять атомних параметрів та три параметри зв'язку для характеристики атомів та їх оточення. До характеристик атома відноситься тип атома ('H', 'C', 'N', 'O', 'F', 'P', 'S', 'Cl', 'Br', 'I', 'інший'), формальні та часткові заряди, хіральність, ароматичність, тип sp-гібридизації, кількість донорів та/або акцепторів водневого зв'язку та розмір кільця. До характеристик зв'язку відноситься тип зв'язку («одинарний», «подвійний», «потрійний» або «ароматичний»), довжина та належність до одного кільця. Більшість цих ознак кодуються підходом one-hot, за винятком зарядів і кількості кілець, до яких належить атом, - ці ознаки є цілими числами через їх адитивну природу.

2.1.6 Підготовка даних для редуکتивного спрощення.

Набір даних із 12351 рядків було створено шляхом поєднання лігандів тромбіну людини з трьох відкритих джерел: BindingDB [203], DUD-E [204] і ChEMBL [70]. Ліганди були представлені за допомогою рядків SMILES [49]. Під час попередньої обробки SMILES були канонізовані за допомогою RDKit [113], щоб усунути неоднозначність у представленні молекули. Дублікати було відкинуто. Викиди були видалені за допомогою

інтерквартильного діапазону (IQR) молекулярних мас лігандів. Було видалено молекули з молекулярною масою менше $Q1 - 1,5 * IQR$ і більше $Q3 + 1,5 * IQR$. Оскільки похибка вимірювання концентрації зростає пропорційно її значенню, ми використовували логарифм концентрації ліганду ($\log_{10}K_i$) як цільову змінну. Гістограма, що показує розподіл зразків за значеннями $\log_{10}K_i$, показана на рис. 2.3, де K_i виражено в наномольях.

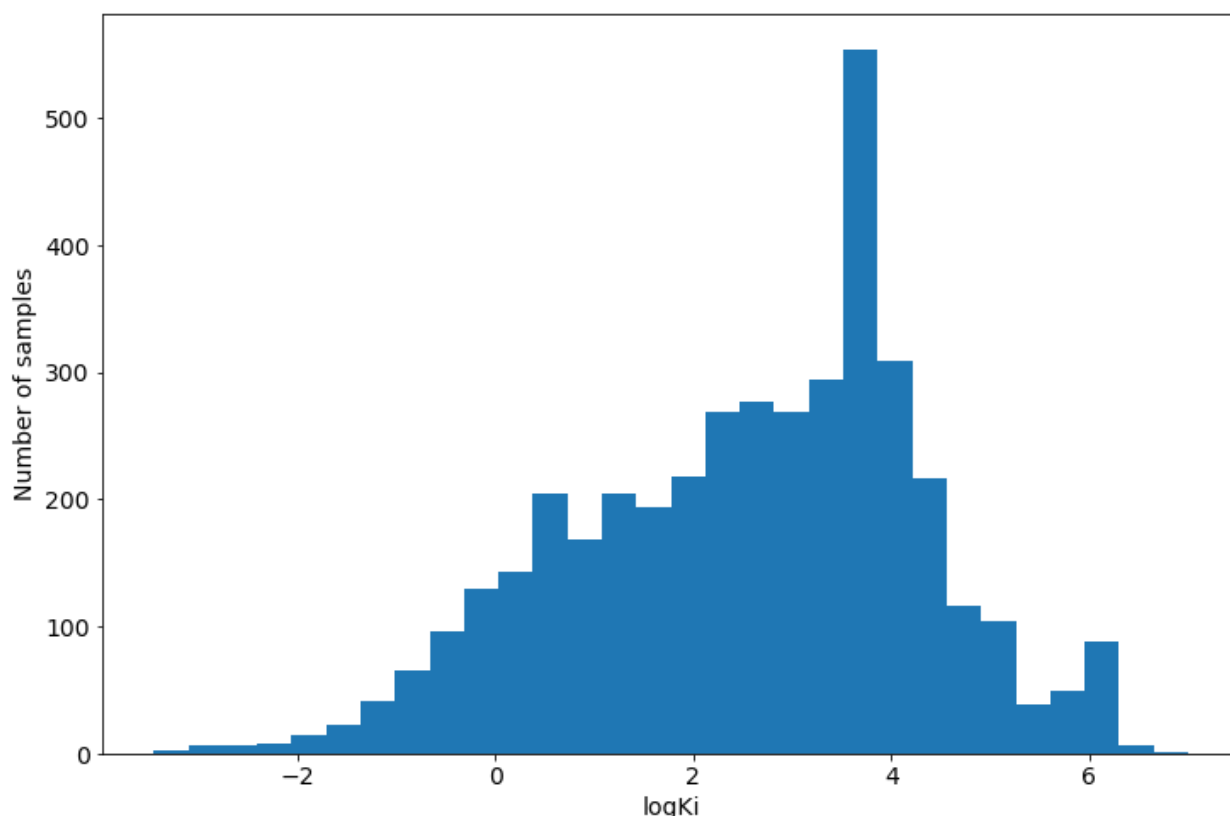


Рис. 2.3. Розподіл значень константи інгібування (K_i , нмоль/л) у формі десяткового логарифму ($\log_{10}K_i$) [5].

Ліганди, представлені рядками SMILES, були закодовані як двійкові відбитки ECFP4 [94] з радіусом 4 і довжиною 2048 біт. Набір даних було розділено на п'ять частин із співвідношенням зразків навчання до тестування 80 до 20 для перехресної перевірки.

2.2 Методологія.

Мета-навчання, яке також називають мета-стекінгом, Super Learning [207] або Stacked Regression [208, 209] — це клас алгоритмів, у

якому навчається мета-модель другого рівня для визначення оптимальної комбінації моделей першого рівня. На відміну від пакування (bagging) та бустингу (boosting), мета-стекинг полягає в тому, щоб об'єднати сильні та різноманітні групи базових моделей. Ансамбль із мета-стекингом є асимптотично оптимальною системою для навчання [207]. Визначальною особливістю Super Learning'у є використання перехресної перевірки (k-fold cross-validation) для формування передбачень «першого рівня», на яких виконується навчання моделей другого рівня - або мета-моделей, - які комбінують моделі першого рівня оптимальним чином.

У цьому підрозділі молекулярна спорідненість передбачається за допомогою поєднання шести моделей машинного першого рівня методом стекингу та наступного тренування моделей другого рівня на передбаченнях моделей першого рівня. Моделі поєднані у два послідовні ансамблі. Використовувалися наступні алгоритми: метод опорних векторів (SVM) [210], випадковий ліс (RF) [211], градієнтний бустинг в реалізації бібліотеки CatBoost (CB) [212], повнозв'язну нейронну мережу прямого поширення (FNN), графова нейронна мережа (GNN) [213], і двоспрямовані кодувальні представлення з трансформерів (BERT) [214]. Перші чотири моделі – SVM, RF, CB, FNN – використовують молекулярні відбитки ECFP4 [215] як вхідні дані. GNN – графове представлення лігандів, інтерпретуючи атоми як вузли, а зв'язки між атомами - як ребра графа [216]. Було обрано графи із параметризацією як вузлів, так і ребер з метою розширення здатності представлення оригінального формалізму GNN. На рис. 2.4 показаний спосіб утворення графового представлення на прикладі молекули хлороформу ($CHCl_3$). На відміну від згаданих вище підходів, що мають справу лише з фізико-хімічними властивостями, BERT працює безпосередньо зі стрічковими представленнями лігандів і, таким чином, усуває потребу в інженерії ознак.

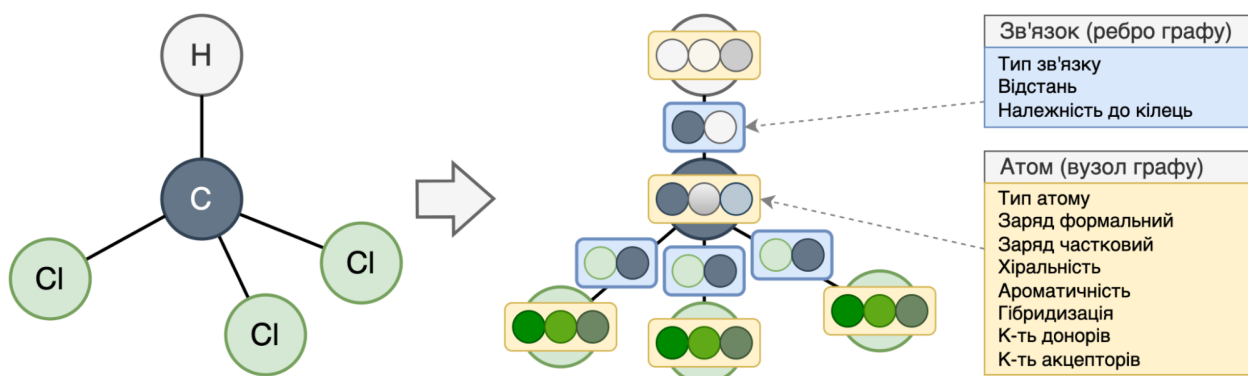


Рис. 2.4. Хлороформ як приклад ліганду, представленого у вигляді молекулярного графу із вузлами (атомами) та ребрами (хімічними зв'язками між атомами). Кожен вузол та ребро описується набором параметрів, що вказані на малюнку [5].

Зазначені моделі та їх ансамблі застосовуються для прогнозування спорідненості органічних молекул до людського тромбіну. Тромбін був обраний через доступність значної кількості публічних даних для навчання моделей, а також перспективи перевірки результатів у лабораторії високопродуктивного скринінгу. Приклад комплексу альфа-тромбіну та гірудину показано на рис. 2.5. Однорецепторна парадигма є поширеним підходом у застосуванні машинного навчання для хемінформатики [67, 68, 69, 217]. Усі моделі можуть бути перенавчені для інших рецепторів.

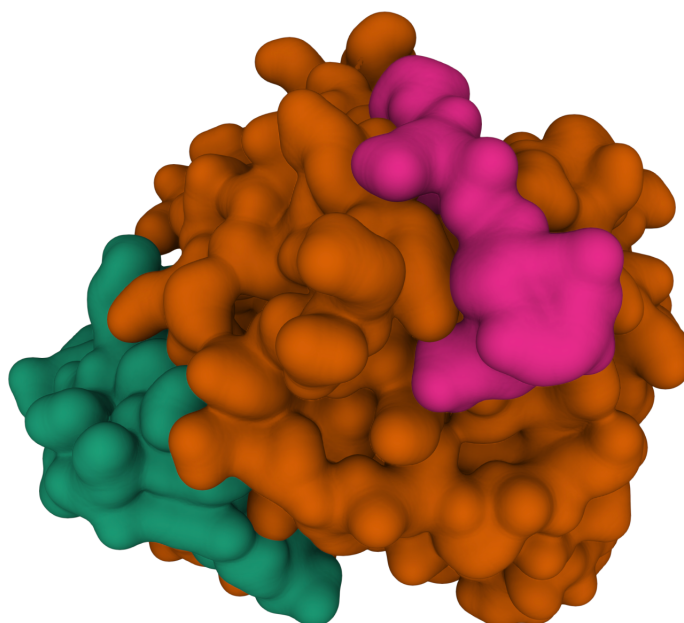


Рис. 2.5. Комплекс людського альфа-тромбіну (велика субодинаця забарвлена коричневим, мала - зеленим), лігандом - пептид гірудин (рожевий) [5].

subsubsectionСхема методу.

Принципова схема методу показана на рис. 2.6. Метод складається із двох поєднаних послідовно ансамблів: перший класифікує ліганди-кандидати на активні та неактивні; другий - прогнозує значення константи інгібування (K_i) для активних лігандів.

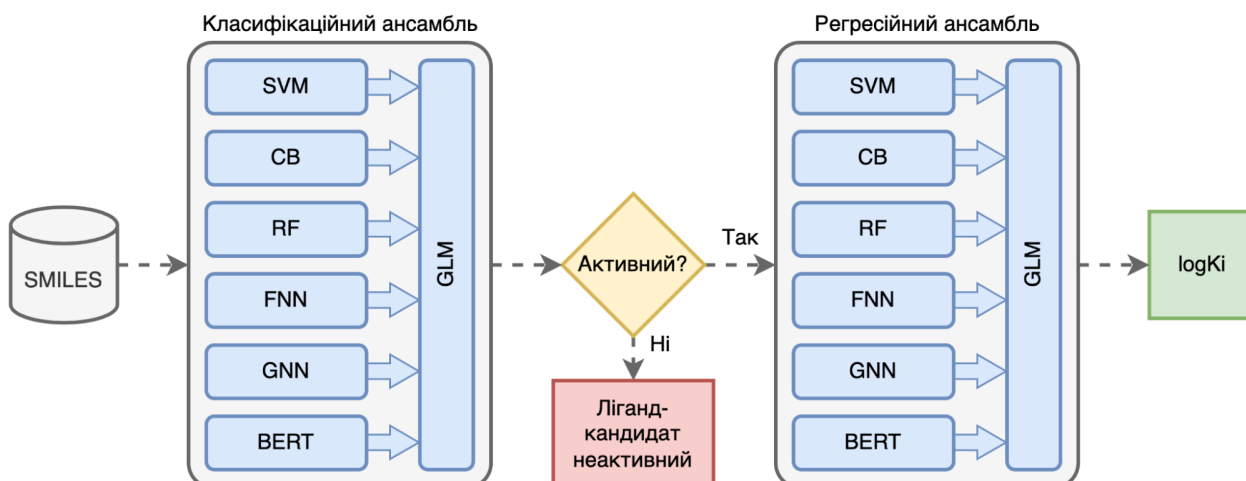


Рис. 2.6. Комплекс людський альфа-тромбін (велика субодинаця забарвлена коричневим, мала - зеленим), лігандом - пептид гірудин (рожевий) [5].

2.2.1 Мета-навчання.

Ансамблі - як класифікаційний, так і регресійний - утворюються із окремих моделей за методом мета-навчання (див. рис. 2.6. Мета-навчання відноситься до класу алгоритмів машинного навчання із вчителем (Supervised Learning). Його суть полягає у тренуванні моделей другого рівня для виявлення оптимальної комбінації моделей першого рівня та поєднання їх у ансамбль із вищою предиктивною здатністю [207]. У якості алгоритму мета-навчання нами був обраний набір лінійних моделей (англ. “Generalized Linear Model” або GLM) у реалізації h2o.ai [218, 219]. На додаток до гаусівського (нормального) розподілу, до GLM належать розподіли Пуассона, біноміальні та гамма-розподіли. Кожен з них служить різним цілям і залежно від вибору функції розподілу та цільового значення може використовуватися або для регресії, або для класифікації [220]. Були обрані гаусова регресія та біноміальна логістична регресія для відповідних задач регресії ($\log K_i$) та класифікації (активний чи неактивний). Детально схема підготовки даних та тренування мета-моделей показана на рисунку 2.7. Щоб знизити або повністю виключити вплив малоінформативних моделей, було застосовано L1 регуляризацію. Кількість перехресних валідацій - 5, функція втрат - Binary LogLoss. Застосовувалася рання зупинка через 10 епох в разі відсутності зменшення значення функцій втрат на валідаційному наборі даних. Солвер - градієнтний спуск.

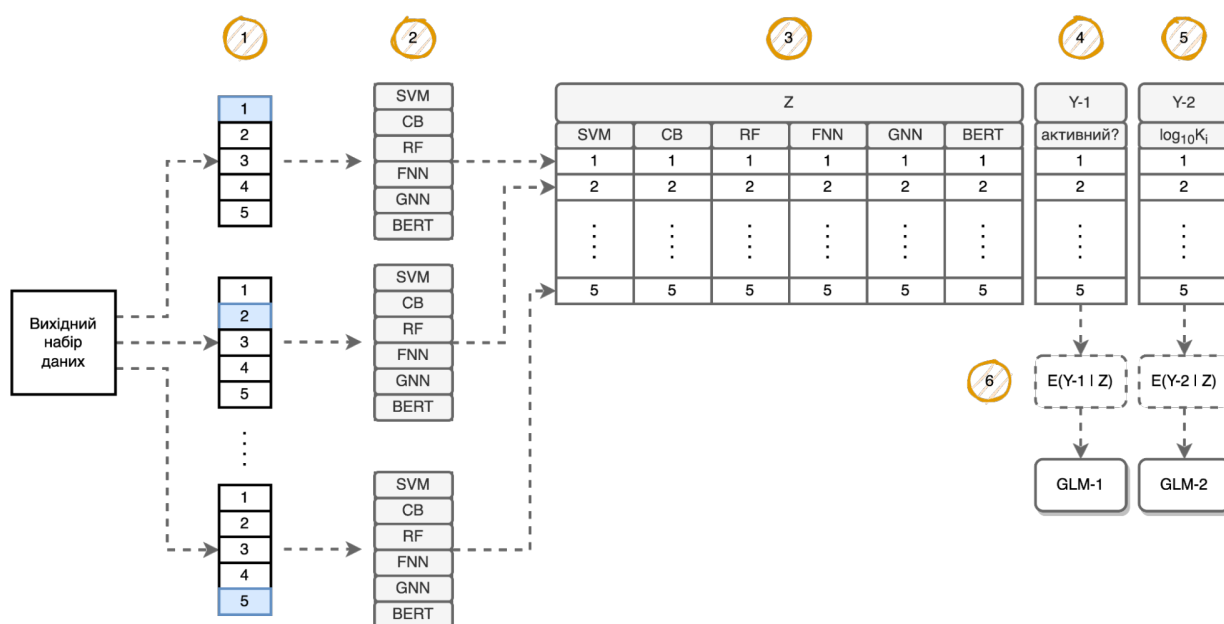


Рис. 2.7. Блок-схема навчання мета-моделі. 1 - крос-валідаційні набори даних. Синім кольором позначено блок даних для валідації. 2 - тренування шести моделей першого рівня. 3 - результати передбачень моделей першого рівня на валідаційних наборах даних на кожному розбитті (1-5). 4 - справжні мітки активності лігандів. 5 - справжні значення $\log_{10}K_i$. 6 - тренування моделей другого рівня із використанням передбачень моделей першого рівня (Z) як вхідних даних та справжніх міток ($Y-1$, $Y-2$) як цільових значень [5].

2.2.2 Метод опорних векторів.

Ми використовуємо метод опорних векторів [210] (SVM) із Гаусівською радіальною базисною функцією (RBF), яка використовується для трансформування простору ознак. Моделі SVM задіяні як у класифікаційних, так і в регресійних ансамблях. В обох випадках були використані реалізації бібліотеки Scikit-learn [221]. Завдяки використанню ядра RBF

$$k(x_1, x_2) = \exp \frac{-1}{2\gamma^2} \|x_i - x_j\| \quad (2.4)$$

на векторах ознак x_i і x_j з двійковими компонентами, отримана модель порівнювала вхідні молекули за схожістю їх відбитків ECFP4 [215] на бітовому рівні. Параметр регуляризації C моделі було обрано методом сіткового пошуку серед значень 0,01, 0,1, 0,2, 0,5, 1,0, 2,0, 5,0,

10,0 і 100,0. Пошук оптимального значення цього параметру показав, що $C = 1$ максимізує метрику точності. Під час навчання моделі було встановлено режим `class_weight = «balanced»`, який використовує значення цільових міток для автоматичного регулювання ваг, обернено пропорційних частотам класів у вхідних даних. Для регресії була використана модель Epsilon-Support Vector Regression [222], яка також реалізована в бібліотеці Scikit-learn.

2.2.3 Випадковий ліс.

Для випадкового лісу [211] (RF) також використано реалізацію від Scikit-learn. RF також використовує параметр `class_weight = «balanced»`, що автоматично регулює баланс між кількістю активних та неактивних зразків у задачі класифікації. На початковому етапі застосовано процедуру пошуку найкращих гіперпараметрів методом сітки. Під час цієї оптимізації були перевірені всі комбінації наступних параметрів: кількість дерев в ансамблі (`n_estimators`) – 200, 500, 1000, 2000, максимальна глибина дерева (`max_depth`) – 2, 5, 7, 8, 10, мінімальна кількість екземплярів, необхідних для поділу вузла (`min_samples_split`) – 1, 2, 4, 8, 10, 20. Перехресна валідація на п'яти поділах була застосована, щоб знайти комбінацію гіпер-параметрів, які дають найкращу усереднену точність. Оптимальною виявилася комбінація таких гіпер-параметрів: `n_estimators = 200`, `max_depth = 10`, `min_samples_split = 10`. Пошук гіпер-параметрів для регресії був подібним, але цього разу в рамках п'яразової перехресної валідації була мінімізована середня квадратична похибка моделі регресії. В результаті, оптимальними для регресії були наступні значення гіпер-параметрів: `n_estimators = 2000`, `max_depth = 10`, `min_samples_split = 2`.

2.2.4 Градієнтний бустинг.

У цьому параграфі описується модель градієнтного бустингу з деревами рішень CatBoost [212] (CB). Моделі були натреновані на бінарних ECFP4 [215] відбитках лігандів. Гіперпараметри відповідних класифікаційних і регресійних моделей спочатку були налаштовані шляхом пошуку по такій сітці: швидкість навчання – 0,1, 0,03, 0,01; L2

регуляризація – 1, 3, 5, 7, 9; максимальна глибина дерев рішень – 6, 8, 10. Показниками для вибору оптимальних гіпер-параметрів були точність для класифікації та MSE для регресії, обчислені на тестовій вибірці. В результаті обраний набір гіпер-параметрів для завдання класифікації включає швидкість навчання = 0,1; параметр регуляризації листків L2 = 3; максимальну глибину дерев = 10. Той самий набір гіпер-параметрів, за винятком регуляризації L2 = 1, використовується для завдання регресії. Валідаційні метрики зазвичай припиняли покращуватися після 200 епох навчання для класифікації та 500 епох для регресії. Під час навчання було застосовано алгоритм ранньої зупинки з параметром patience, встановленим на 100 епох.

2.2.5 Нейронна мережа прямого поширення.

У цьому параграфі описується підхід, заснований на формалізмі нейронних мереж прямого поширення (FNN). Подібно до наведених вище методів, вхідними даними FNN є відбитки ECFP4. Розмір вхідного шару становить 2048 нейронів. Вхідні дані надалі надходять у набір повнозв'язних шарів. Під час оптимізації гіперпараметрів було випробувано різні архітектури FNN, у яких змінювали ширину та глибину мережі, спробували рекурентні та пропускові з'єднання, такі функції активації як ReLU, Softplus та PReLU. Незважаючи на наші очікування, рекурентні та пропускові з'єднання не покращили продуктивність як для класифікації, так і для регресії. У випадку класифікації, остаточна архітектура складалася з п'яти шарів: вхідний шар розміром 2048, три прихованих шари з 512, 256 і 64 нейронами відповідно та вихідний шар із 2 нейронами. Функція активації на першому, другому та третьому рівнях — ReLU. У випадку завдання регресії архітектура складалася також з п'яти шарів: вхідний шар має розмір 2048, приховані шари – 1024, 256 і 64 нейронів відповідно, а вихідний шар – з 1 нейроном. Функція активації для прихованих шарів – Softplus. Навчання проводилося за допомогою технології стохастичного градієнтного спуску з оптимізатором Adam та розміром батчу рівним 32. Під час навчання ми також використовували планувальник швидкості навчання, який зменшував початкову швидкість навчання в 0,001 кожні 50 епох навчання в 0,9 раза. У задачі класифікації

навчена модель була оптимізована за втратою перехресної ентропії, тоді як найкраща модель була обрана відповідно до кращого показника точності в тестовій підмножині. MSE використовувався як функція оптимізації втрат у задачі регресії. Була обрано модель із найнижчим MSE на тестовій підмножині.

2.2.6 Графова нейронна мережа.

Як вже зазначалося, представлення лігандів у рамках підходу GNN будуються у вигляді молекулярних графів. У роботі використовується модель AttentiveFP GNN [223, 224] у реалізації DGL-LifeSci [225] як для завдань класифікації, так і для регресії. AttentiveFP передає повідомлення між вузлами та вивчає нелокальні ефекти завдяки механізму уваги. Механізм уваги полягає у тому, що додаткові шари нейронів навчаються на вагах вузлів. Це дозволяє атомам агрегувати ознаки стану сусідів та поширювати власні ознаки назад до сусідів. Таким чином, окремі атоми роблять внесок у вектор стану усієї молекули. Результати AttentiveFP передаються на повнозв'язний шар із двома (класифікація) або одним (регресія) нейроном. Перерахуємо деякі параметри архітектури: кількості параметрів вузлів і ребер становили 27 і 12. Векторні представлення для атомів (вузлів) та хімічних зв'язків (ребер) були згенеровані за допомогою WeaveAtomFeaturizer і WeaveEdgeFeaturizer [186]; кількість шарів в AttentiveFP – 2, розмір ознак графу – 200, кількість рекурентних кроків – 2, коефіцієнт дропауту – 0,2. Вся мережа навчалася наскрізним способом за допомогою оптимізатора Adam з розміром батчу рівним 30, коефіцієнтом регуляризації L2 - 0,0002 та затримкою ранньої зупинки рівною 40 епох. Початкова швидкість навчання 0,001 поступово зменшувалася в 0,9 разів кожні 10 епох протягом процесу навчання. У задачі класифікації ми використали фокальну функцію втрат [226], а у задачі регресії - MSE як для оптимізації гіпер-параметрів, так і для вибору найкращої моделі.

2.2.7 Двоспрямовані трансформери.

У цьому параграфі описується методологія навчання, підготовки даних та оптимізації гіпер-параметрів моделі-трансформера. Ми обрали двоспрямовану трансформерну модель BERT [214, 119], оскільки вона

добре враховує як лівий, так і правий контексти під час побудови векторних представлень, що має вирішальне значення для повного охоплення складних взаємозалежностей у хімічних структурах. Навчання BERT зазвичай складається з двох етапів. Перший етап – попереднє навчання на великому нерозміченому наборі даних з метою прогнозування випадково замаскованого символу у послідовності. Другий етап – донавчання моделі для конкретного завдання з відповідно розміченим (невеликим) набором даних.

Для завдання класифікації архітектура BERT виглядає наступним чином: 4 шари, 12 внутрішніх нейронних мереж механізму уваги, прихований розмір 768. Для кожного символу з рядка SMILES створюється ембедінг шляхом додавання позиційного та контекстного кодувань. Попередньо натреновані ваги BERT [227] були оптимізовані на приблизно 155 000 послідовностях SMILES з бази даних PubChem [228]. Максимальна довжина рядка SMILES була рівна 128. Словник створювався за допомогою алгоритму BytePair criteGage1994; його розмір становить приблизно 52 000 токенів. На етапі донавчання ми експериментували з кількома функціями втрат. Найкраща точність і precision/recall були досягнуті за допомогою зваженої фокальної функції втрат із параметрами гама 2 і альфа 0,81. Щоб уникнути перенавчання та покращити узагальнення під час навчання, у BERT було додано ймовірності дропауту 0,3 до входних шарів, шарів уваги та прихованих станів, що виводяться кожним шаром моделі. Регресійна модель BERT була меншою: 3 прихованих шари, 6 self-attention шарів, розміром 768 та розміром словника 2229 токенів. Попередньо навчені ваги для такої конфігурації також були взяті з Wolf [227], які були оптимізовані на 400 000 послідовностях SMILES з бази даних ChEMBL [70]. Навчання обох моделей проводилося протягом 8 та 14 епох відповідно, з batch size рівним 32. Початкова швидкість навчання становила $9 * 10^{-6}$ та $1 * 10^{-4}$ для класифікації та регресії відповідно та лінійно знижувалася протягом навчання. Ваги моделей оновлювалися оптимізатором Adam [229].

2.2.8 Редуктивне спрощення.

Основна концепція редукції [230, 231, 232, 233] та редуктивного спрощення [9] полягає в ітераційному визначенні та видаленні ваг FNN, які найбільше реагують на невеликі збурення. Збурення вносяться до цільової змінної. Для цього FNN було навчено у двох варіантах: 1) контрольна модель (CM) – на цільових значеннях спорідненості у незмінному вигляді та 2) редукційний експеримент (RE) – з використанням значень спорідненості з шумом. Параметри навчених мереж CM і RE порівнювали, щоб визначити ваги RE з найбільшими відносними відхиленнями від відповідних ваг CM. Ці ваги визнавалися надлишковими та видалялися. Потім оцінювалася ефективність узагальнення мережі RE. Видалення ваг вважалось успішним, якщо значення функції втрат під час перехресної перевірки покращилися. У цьому випадку проводилася наступна ітерація редукційного експерименту (спроба видалити більше ваг). В іншому випадку - якщо значення функції втрат погіршилося - завантажувалася останній успішний "спрощений" набір ваг, а процес видалення ваг припинявся.

Спершу, окремо було виділено 20% від загальної вибірки для out-of-sample тесту. Інші 80% використовувалися для тренування та тестування. Щоб уникнути підлаштування алгоритму до будь-якого конкретного поділу даних, усі моделі валідувалися методом п'ятикратної перехресної перевірки. Кожен фолд мав 80:20 тренувальних та тестових даних, фолди не перекривались. Ітераційне видалення ваг припинялося при першому погіршенні середнього значення функції втрат (RMSE) перехресної перевірки. Додатково ефективність узагальнення редукованих моделей оцінювалася на out-of-sample тестових даних (20%), як показано на рис. 2.8.

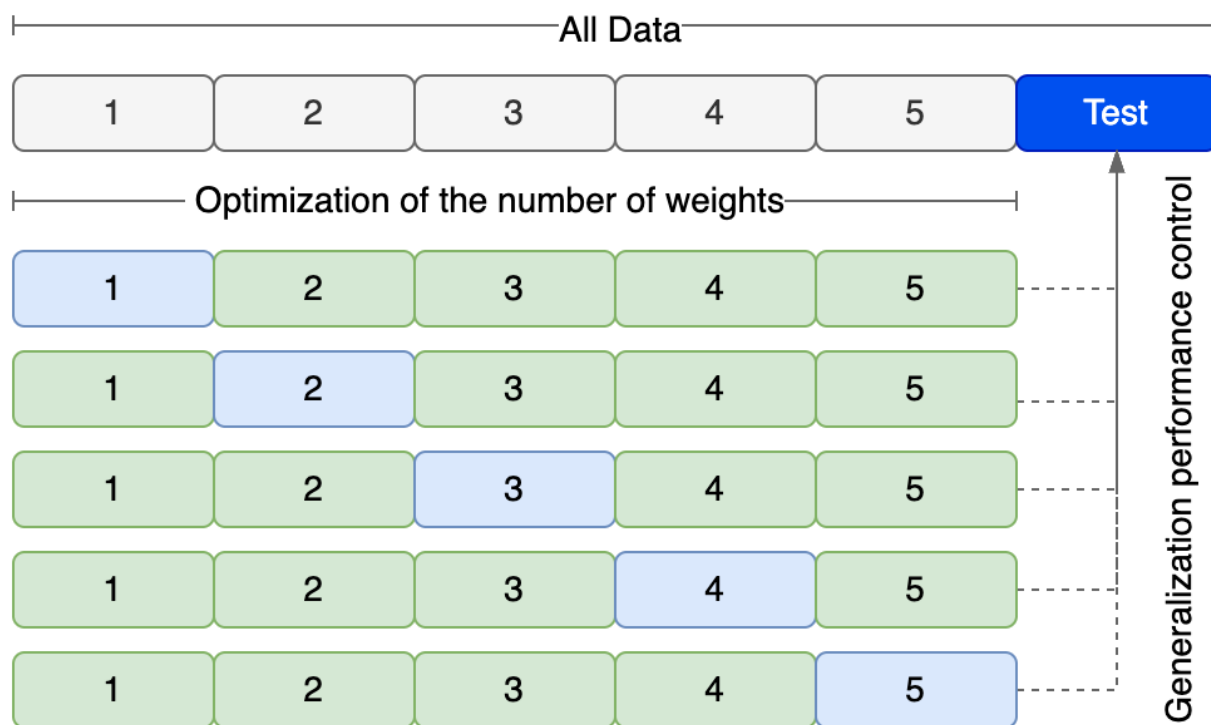


Рис. 2.8. Схема п'ятикратної перехресної перевірки для редуційного спрощення з використанням 80% даних (сірим кольором). Додатково ефективність узагальнення редукованих моделей оцінювалася на out-of-sample тестовому наборі (20%, синім кольором) [9].

Ми навчали одну контрольну модель СМ і дві експериментальні моделі RE на кожному з п'яти поділів даних. Щоб диверсифікувати збурення, одна з двох мереж RE навчалася на наборі даних, де вихідні значення молекулярної спорідненості були помножені на 0.9, а інша - на наборі даних зі значеннями спорідненості помноженими на 1,1 (див. рис. 2.9). Усі 15 нейронних мереж (3 експерименти по 3 мережі на 5-ти розділах) були ініціалізовані однаковими початковими вагами.

Нижче наводиться алгоритм редуційного спрощення:

1. Ініціалізація

- (a) Ініціалізація початкових ваг моделей СМ та RE однаковими наборами ваг.
- (b) Ініціалізація бінарної маски M такого ж розміру, що й ваги моделей. Надалі маска множитиметься поелементно на ваги СМ і RE моделей на етапах навчання та тестування.

Початкова маска M_{nil} вказує, що всі ваги активні. Пізніше ідентифіковані надлишкові ваги позначаються нулями за відповідними індексами у масці M .

2. Ітеративне тренування, кросвалідація, видалення надлишкових ваг

- (a) На початку кожної нової епохи ініціалізується вектор двох коефіцієнтів збурення $disturb = [d_1, d_2]$. d_1 вибирається з діапазону $[0,9, 1,0)$, d_2 - з $(1,0, 1,1]$ так, щоб d_1 і d_2 мали рівні абсолютні відхилення від 1,0, строго більше нуля.
- (b) Множення коефіцієнту d_1 на цільові значення $RE_{0,9}^1, RE_{0,9}^2, RE_{0,9}^3, RE_{0,9}^4$ і $RE_{0,9}^5$ наборів даних - тобто створення 5 наборів даних із зашумленими цільовими значеннями.
- (c) Множення коефіцієнту d_2 на цільові значення $RE_{1,1}^1, RE_{1,1}^2, RE_{1,1}^3, RE_{1,1}^4$ і $RE_{1,1}^5$ - тобто створення 5 наборів даних із зашумленими цільовими значеннями.
- (d) Цільові значення CM моделей залишаються незмінними для порівняння - тобто 5 наборів даних із незмінними цільовими значеннями.
- (e) Навчання та 5-кратна перехресна перевірка моделей CM і RE - тренування 15 моделей. Надлишкові ваги, ідентифіковані за попередніх епох, вимикаються поелементним множенням маски M .
- (f) Обчислення середніх відносних відхилень ваг моделей $RE_{0,9}$ ($n = 1$) і $RE_{1,1}$ ($n = 2$) (навчених на наборах даних зі збуреннями цільової змінної) від ваг CM (навчених на незміненому вихідному наборі даних) на кожному з k кросвалідаційних фолдів:

$$\delta_k = \frac{1}{n} \sum_{re=1}^n \frac{|w_{re} - w_{cm}|}{w_{cm}} \quad (2.5)$$

де δ_k - матриця середніх відносних відхилень ваг моделі RE^k від контрольних ваг CM^k у межах фолду k , $k \in 1, 2, 3, 4, 5$; w_{re} - вагові коефіцієнти n -ї моделі RE в межах фолду k , $w_{re} \in$

$RE_{0.9}^k, RE_{1.1}^k$; w_{cm} - вагові коефіцієнти моделі CM для фолду k ,
 $w_{cm} \ni CM^k$.

- (g) Обчислення середнього відносного відхилення серед усіх k фолдів:

$$\delta_{avg} = \frac{1}{k} \sum_{i=1}^k \delta_k \quad (2.6)$$

де δ_{avg} - матриця середніх відносних відхилень по усіх фолдах;

- (h) Запис поточної маски M у змінну M_{prev} .
- (i) Визначення N найбільших відхилень у матриці δ та оновлення маски M шляхом обнулення значень у відповідних індексах. Ці обнулені індекси "виключають" надлишкові ваги на усі наступних епохах. Початкове значення N становило 10%. Було протестовано 1%, 5%, 10%, 15% і 20% та виявлено, що 10% є оптимальним компромісом між швидкістю видалення ваг та покращенням продуктивності моделі для обох наборів даних.
- (j) Множення оновленої маски M на ваги моделі CM .
- (k) Оцінка якості генералізації моделі CM на out-of-sample тестовому наборі (20%) (див. синій прямокутник на рис. 2.9).
- (l) Якщо ефективність узагальнення спрощеної моделі CM стає кращою, перехід до кроку 2(a) з останнім N ; інакше, якщо $N > 0,01\%$, заміна M на M_{prev} та перехід до кроку 2(a) з $N = \frac{N}{2}$; інакше - зупинка процесу редукції ваг та повернення M_{prev} як результату.

3. Пост-процесинг ваг та архітектури мережі

- (a) Відкидання усіх "обнулених" ваг моделі CM за індексами у фінальній масці.
- (b) Редагування архітектури нейронної мережі відповідно до маски.
- (c) Збереження архітектури та ваги спрощеної моделі.

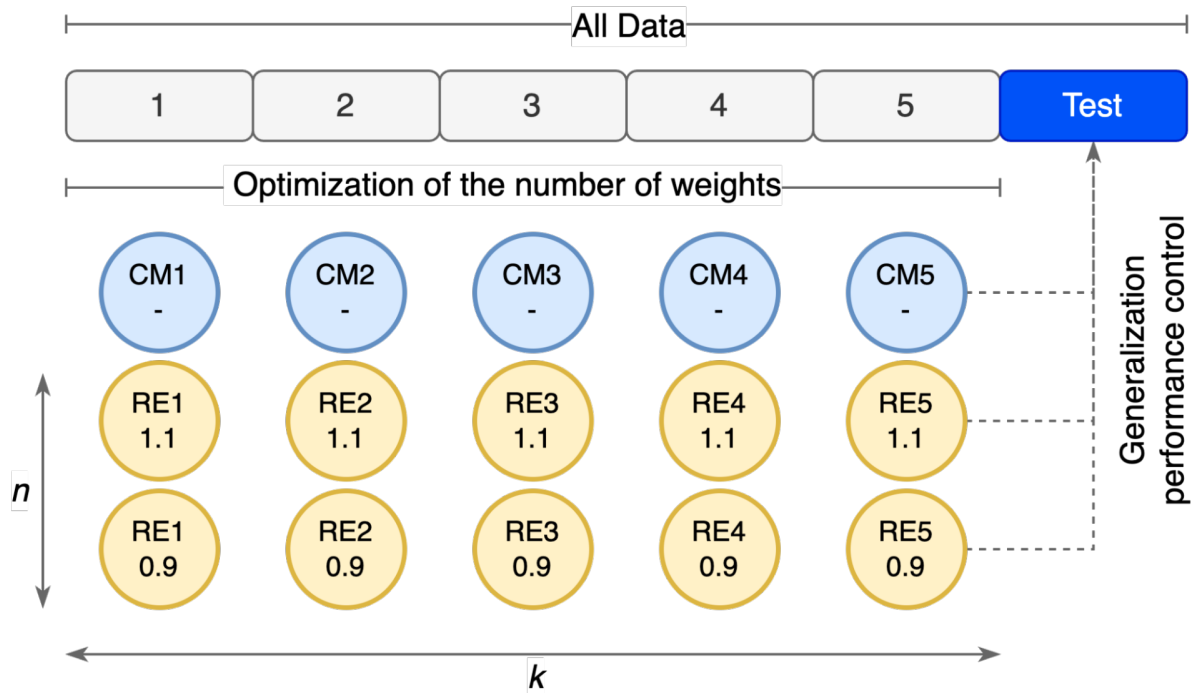


Рис. 2.9. Контрольні (CM1...CM5) та редукційні моделі (RE1...RE5), що реагують на два рівні збурень цільової змінної (0,9 і 1,1). k - індекс кратності перехресної перевірки, $k \in 1, 2, 3, 4, 5$; n - індекс групи RE, $n \in 1, 2$ [9].

2.3 Результати.

2.3.1 Поодинокі регресійні та класифікаційні моделі.

У цій частині ми аналізуємо ефективність моделей класифікації за такими метриками як accuracy, precision, recall, та AUC. Оцінка виконувалась за методом п'ятиразової перехресної валідації: кожна з моделей була натренована та провалідована п'ять разів - щоразу на унікальному розбитті вихідного набору даних на тренувальну та тестову підмножини. Тобто загалом було отримано п'ять моделей і п'ять наборів метрик. Усереднені значення класифікаційних метрик показані в Таблиці 2.1. Найраці результати за AUC серед класифікаційних моделей показали SVM (0.98), CB (0,97) та RF (0,97).

Також наведено результати тренування регресійних моделей: усереднені середньоквадратичні (MSE), абсолютні похибки (MAE) та коефіцієнти детермінації (R^2). Серед окремих регресійних моделей,

найменші похибки та найвищі R^2 продемонструвала модель SVM. Варт відзначити, що складні моделі засновані на нейронних мережах (FNN, GNN, BERT) показали гірші результати за лінійні моделі (SVM) та моделі засновані на деревах рішень (CB, RF). Це можна пояснити порівняно невеликим набором даних для класифікації та малим - для регресії.

Таблиця 2.1

Зведена таблиця показників регресійних та класифікаційних моделей розрахованих за методом п'ятиразової перехресної валідації [5].

Модель	Класифікація				Регресія		
	ACC	PRC	RCL	AUC	R^2	MAE	MSE
SVM	0.95	0.84	0.91	0.98	0.74	0.55	0.56
RF	0.93	0.80	0.82	0.97	0.66	0.65	0.71
CB	0.94	0.87	0.77	0.97	0.71	0.59	0.62
FNN	0.95	0.85	0.86	0.91	0.69	0.60	0.66
GNN	0.88	0.80	0.45	0.94	0.69	0.59	0.66
BERT	0.87	0.69	0.55	0.90	0.61	0.66	0.82
Сер.	0.92	0.808	0.727	0.945	0.68	0.61	0.67
Анс.	0.99	0.94	0.98	0.998	0.827	0.49	0.49
Δ	+7.6%	+16.3%	+34.9%	+5.6%	+21%	-19.2%	-27%

2.3.2 Стекінг поодиноких моделей та мета-навчання.

Ансамблювання моделей виявляється результативним, якщо частота та розподіл помилок між методами різняться: зменшення кількості “спільних” помилок моделей призводить до зниження кількості FP ансамблю. Щоб перевірити цю гіпотезу, потрібно дослідити частоту перекривань помилок окремих моделей, що входять до ансамблю. Це можна зробити, підрахувавши площу перетину об'єднання (IoU) хибно-негативних (FN) та хибно-позитивних (FP) результатів усіх методів. Відповідні коефіцієнти IoU представлені на рис. 2.10. Найбільший внесок у виключення хибно-позитивних помилок зробила пара FNN-GNN (перекриття FP IoU рівне 0.18); найменший внесок зробили концептуально різні підходи SVM і RF (0.72). Найбільше перекриття хибно-негативних значень спостерігається у пари SVM-FNN (0.92), тоді як найнижчий - 0,59–0,61 – для комбінацій GNN

з іншими методами. Такий аналіз виконаний лише для класифікаційних моделей, оскільки він є простішим.

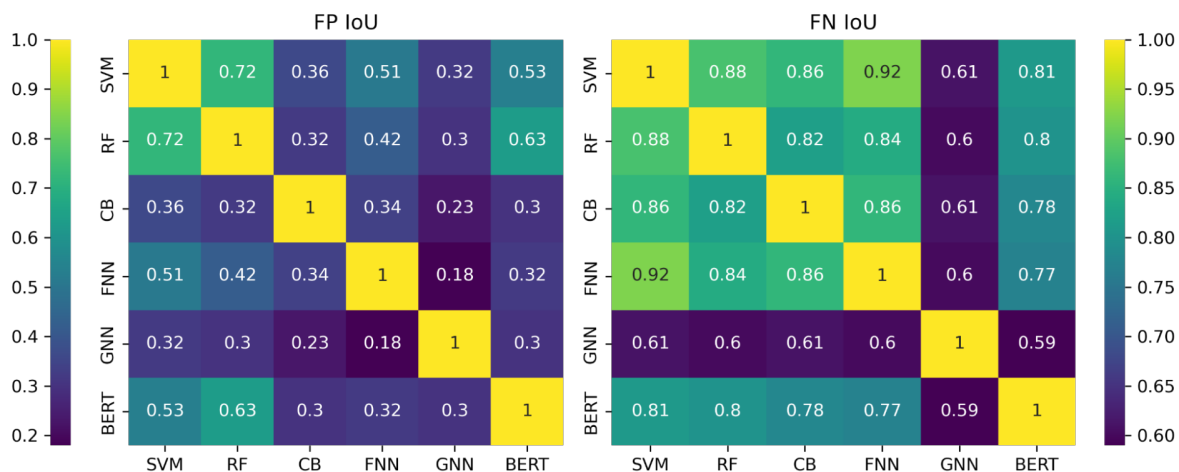


Рис. 2.10. Матриці коефіцієнтів перетину по об'єднанню (IoU) хибно-позитивних (FP IoU, зліва) та справжніх позитивних (FN IoU, справа) передбачень усіх моделей класифікаційного ансамблю [5].

Для покращення ефективності передбачень, ми поєднали окремі моделі у ансамблі методом мета-стекингу. Навчання мета-моделей проводилося на перехресних передбаченнях базових моделей (5 підвибірок), які поєднувалися у матриці розміром $N \times L$, де N - кількість рядків набору даних, L - кількість базових моделей, яка у нашому випадку становила 6. Тренування та валідація моделей другого рівня (мета-моделей) також виконувалася із 5-разовою перехресною валідацією. Таке комбінування слабших моделей у ансамблі із мета-моделями дозволило покращити відгук (Recall) класифікаційного ансамблю на 34,9% та коефіцієнт детермінації (R2) регресійного ансамблю на 21% у порівнянні із середніми значеннями окремих моделей. Усі результати можна бачити у зведеній Таблиці 2.1.

Для мета-стекингу використовувалися лінійні моделі (GLM). Коефіцієнти цих лінійних моделей розраховувалися шляхом максимізації оцінки максимальної правдоподібності (maximum likelihood estimation), отож абсолютні значення знайдених ваг моделей другого рівня можна розглядати як мірило інформаційного внеску певної моделі-учаснику ансамблю до загального результату. Абсолютні величини коефіцієнтів були нормалізовані та виражені у відсотковому вигляді. Результат можна бачити на Рис. 2.11.

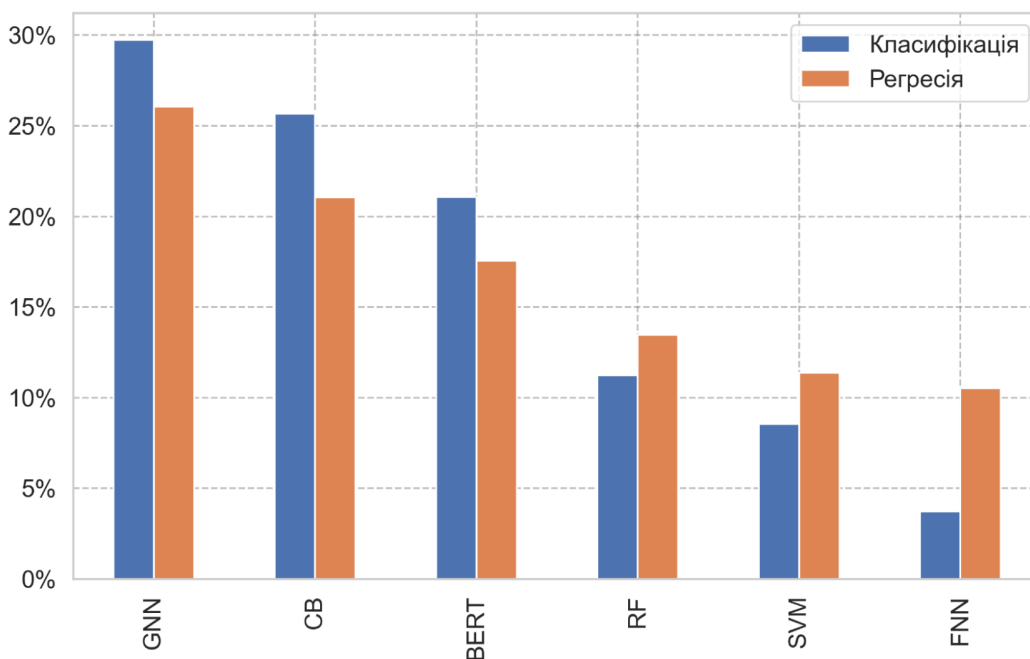


Рис. 2.11. Нормалізовані коефіцієнти класифікаційних (синім) та регресійних (помаранчевим) мета-моделей [5].

2.3.3 Редуктивне спрощення для моделей молекулярної спорідненості.

Редуктивне спрощення [230, 231, 232, 233, 9] застосовувалося до повнозв'язної нейронної мережі з одним прихованим шаром (FNN) для прогнозування молекулярної спорідненості ($\log_{10}K_i$). Початкова мережа мала 2 098 176 ваг. Для кожної ітерації редуктивного спрощення на тестовому наборі поза вибіркою (див. рис. 2.8 і рис. 2.9) розраховувалися та порівнювалися R^2 , MSE , MAE , максимальна помилка, пояснена дисперсія, реалізовані у бібліотеці [221]. Еволюція функції втрат RMSE і метрики показана на рис. 2.12.

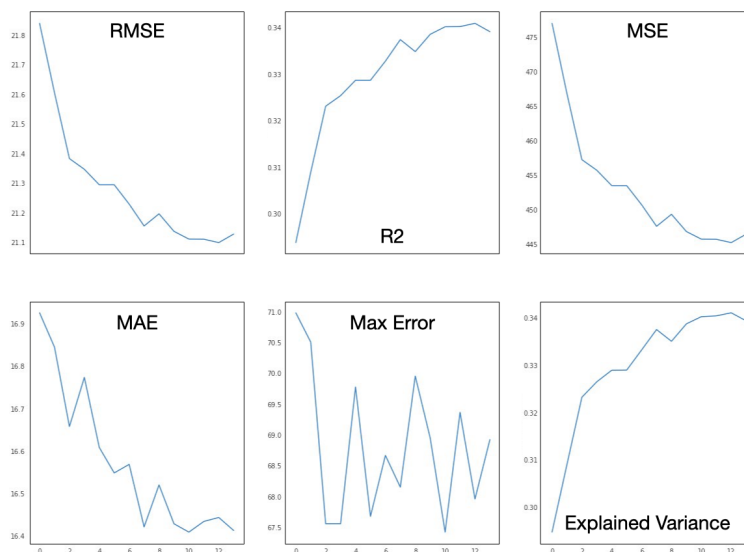


Рис. 2.12. Еволюція метрик спрощених моделей CM на out-of-sample тестовому наборі даних (20%) під час редукції ваг мережі для прогнозування молекулярної спорідненості [9].

Видалення ваг було зупинено на 14-й епосі через погіршення перехресних валідаційних втрат з N менше за 0,01%. Таким чином, оптимальною виявилася маска 13-ї (попередньої до останньої, 14-ї) епохи. У таблиці 2.2 порівнюються початкова та кінцева продуктивність моделей та кількість ваг. Легко побачити, що в цьому випадку метод зменшення не тільки дозволив видалити майже 90% ваг мережі, але й покращив функцію втрат на 5,16%. Ці результати можна порівняти з найефективнішими методами скорочення, згаданими в Розділі 1, з точки зору кількості видалених ваг, але - і це найважливіша особливість методу - спрощена мережа має підвищену здатність до узагальнення замість погіршеної - як це зазвичай відбувається у методах спрощення.

Порівняння повної та спрощеної FFN для передбачення молекулярної спорідненості [9].

Метрика	Повна модель	Спрощена модель	Зміна, %
RMSE (ф-я втрат)	0.794	0.753	-5.16
R^2	0.787	0.809	+2.80
MSE	0.63	0.568	-9.84
MAE	0.569	0.544	-4.39
Найбільша помилка	3.463	3.338	-3.61
Пояснена дисперсія	0.789	0.809	+2.53
Активні ваги	2,098,176	275,091	-86.88

2.4 Висновки.

У цьому розділі описано новий метод високопродуктивного віртуального скринінгу для передбачення молекулярної спорідненості лігандів до одного рецептора. Основою методу є використання методу мета-стекингу та різноманітних за своєю природою моделей машинного навчання: методу опорних векторів, випадкового лісу, градієнтного бустингу, повнозв'язної нейронної мережі прямого поширення, графової нейронної мережі і двоспрямованого кодувального представлення з трансформерів. Реалізовано два послідовні ансамблі: класифікаційний та регресійний. Класифікаційний ансамбль передбачає імовірність зв'язування рецептора та ліганда. Ліганди, які були класифіковані як активні, надходять до регресійного ансамблю, який передбачає спорідненість ліганду до рецептору кількісно - у вигляді константи інгібування K_i .

Запропонований метод дає змогу передбачити активність молекул-кандидатів не лише якісно (активний або неактивний), але й кількісно (значення K_i). Показано, що поєднання моделей методом мета-стекингу збільшує відгук (Recall) класифікації на **34,9%**, підвищує загальну точність та статистичну достовірність результатів (R^2) на **21%**; дозволяє виключити (у випадку класифікації) або компенсувати (у випадку регресії) помилки, допущені іншими моделями ансамблю.

Застосування редуکتивного спрощення [230, 231, 232, 233, 9] до мережі прямого поширення (FFN) першого рівня дозволило зменшити кількість її активних ваг на 86.88% та одночасно із цим покращити значення функції втрат на 5,16%, а коефіцієнт детермінації (R^2) - на 2.8% [9].

3 РОЗДІЛ 3. МЕТОД ПРОГНОЗУВАННЯ ВИХОДУ ПРОДУКТУ ХІМІЧНОЇ РЕАКЦІЇ.

У цьому розділі запропоновано нову архітектуру графової нейронної мережі спрямованої передачі повідомлень для прогнозування властивостей хімічних реакцій (Directed Message-Passing Neural Network for chemical Reaction properties, RD-MPNN). Мережа використовує структурну інформацію про учасників реакції, а також дескриптори рівня молекули та реакції для прогнозування фактичного виходу продукту хімічної реакції (Yield) у числовому (відсоток виходу продукту) та категорійному (рівень виходу продукту) представленнях. Показано, що ефективність розробленої графової мережі рівна або перевершує усі відомі на момент публікації підходи та моделі машинного навчання на двох публічних (реакції Сузукі–Міяури [104] та реакції Бухвальда–Гартвіга [103]) та одному наборі даних (Enamine [234]). Розробку технології доповнено описом даних та процедури їх підготовки. Проаналізовано результати і похибки, а також специфічні доменні причини виникнення похибок, що виходять за межі наявної у даних інформації: вплив просторових факторів на вихід продукту реакції, вплив побічних реакцій, способу та ефективності очищення кінцевого продукту. Результати опубліковано у працях автора [6, 9].

3.1 Аналіз даних.

3.1.1 Публічні набори даних.

Для порівняння розробленої у підрозділі 3.3 графової мережі із найкращими відомими на час публікації [6] методами машинного навчання для прогнозування фактичного виходу продукту, було використано два публічні набори даних – реакції Сузукі–Міяури [104] і реакції Бухвальда–Гартвіга [103]. Кожен із вказаних наборів даних містив записи реакцій одного іменного класу реакцій - відповідно до назви механізму.

Набір даних реакції Сузукі–Міяури містив 5760 записів реакцій високопродуктивного скринінгу з фактичними виходами продукту.

Записи реакцій містили інформацію про реагенти, реактанти, назви лігандів, а також їх молярні еквіваленти та молярні концентрації. Розподіл фактичного виходу продукту в наборі даних Сузукі–Міяури продемонстровано на рисунку 3.1.

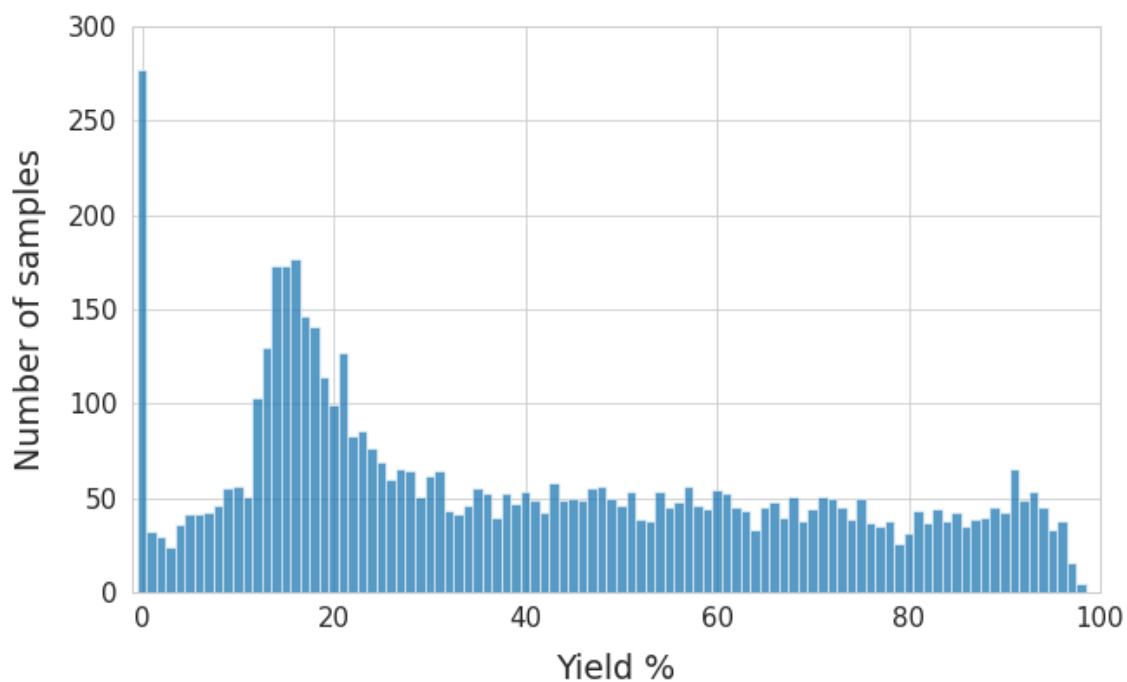


Рис. 3.1. Розподіл фактичного виходу продукту (Yield) в наборі даних Сузукі–Міяури [104, 6].

Набір даних реакції Бухвальда-Гартвіга містив 4608 каталізованих паладієм міжмолекулярних реакцій перехресного сполучення C-N, виконаних у надвисокопродуктивній установці. Записи містять реагенти, адитиви, основи та продукти, закодовані як рядки SMILES. Розподіл фактичного виходу продукту показано на рисунку 3.2.

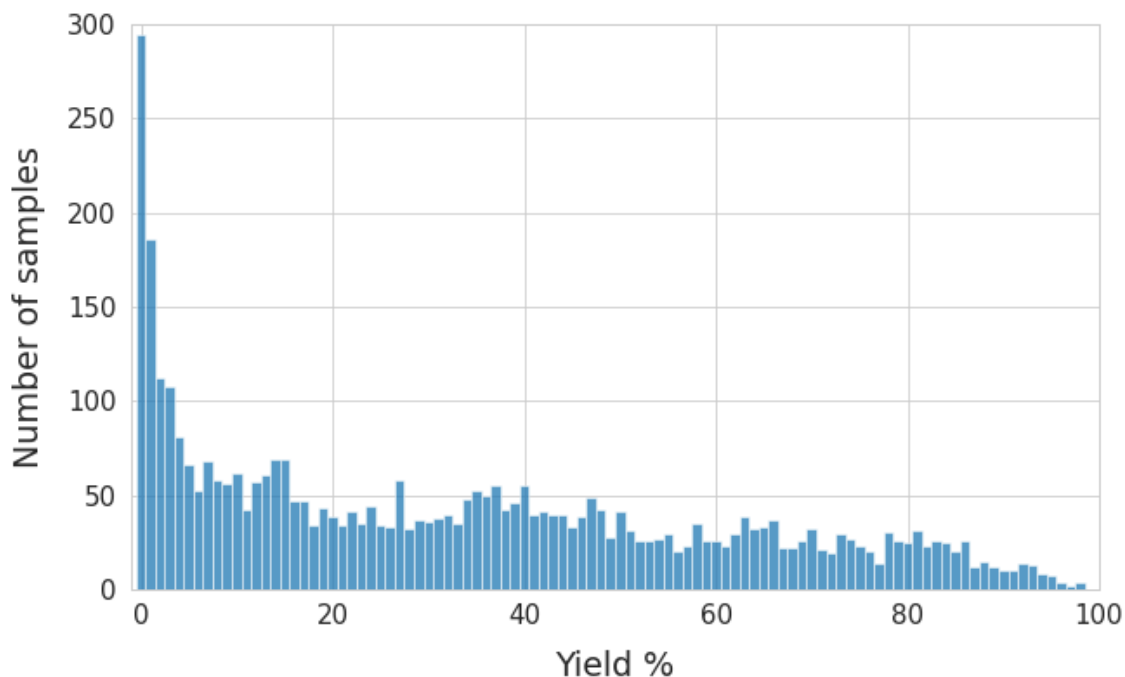


Рис. 3.2. Розподіл фактичного виходу продукту (Yield) в наборі даних реакції Бухвальда-Гартвіга [103, 6].

Варто звернути увагу на кілька спостережень, пов'язаних із вказаними вище наборами даних.

По-перше, побічні продукти були пропущені в обох наборах даних. Внаслідок, формалізм закону збереження матерії не дотримувався в записах реакцій: кількість і тип реагуючих атомів не дорівнювали кількості і типу атомів продуктів реакції. Тому, для обчислення графових ембедінгів різниці продуктів і реагентів, нам довелося створити алгоритм "дописування" незавершених хімічних реакцій.

По-друге, у $\sim 30\%$ реакцій у кожному з наборів даних не було виділено цільового продукту: фактичні виходи становили 0% . Таким чином, обидва набори даних містили значну частину непродуктивних реакцій.

3.1.2 Пропріетарний набір даних Enamine.

У цьому розділі описано пропріетарний набір реакцій Enamine [234], які часто використовуються в сучасному органічному синтезі. Початковий набір необроблених даних містив 80014 реакцій, виконаних у режимі паралельного синтезу. Після очищення даних, канонізації і видалення

повторних записів залишилося 79904 реакції. Початкові записи реакцій містили наступні колонки: SMILES та InChI-коди цільового продукту; категорійні та числові форми значень фактичного виходу продукту; унікальний ідентифікатор класу реакції (ID); SMILES та ID реагентів; SMILES каталізаторів та інших допоміжних сполук.

Було видалено стовпці з кодами InChI продукту та ID реагентів як дубльовану інформацію. Як цільові змінні було використано числові та категорійні виходи продукту. Нагадаємо, що числовий вихід реакції — це відношення кількості молей утвореного продукту до кількості молей використаних реагентів. Розкид значень числових виходів становив 0% - 100%. Категорійним представленням числового виходу є рівень виходу продукту: нульовий (цільовий продукт реакції не було виділено), низький (0.1% - 33%), середній (34% - 66%) і високий (67% - 100%). Безперервні та категорійні розподіли значень фактичного виходу продукту показані на рисунку 3.3.

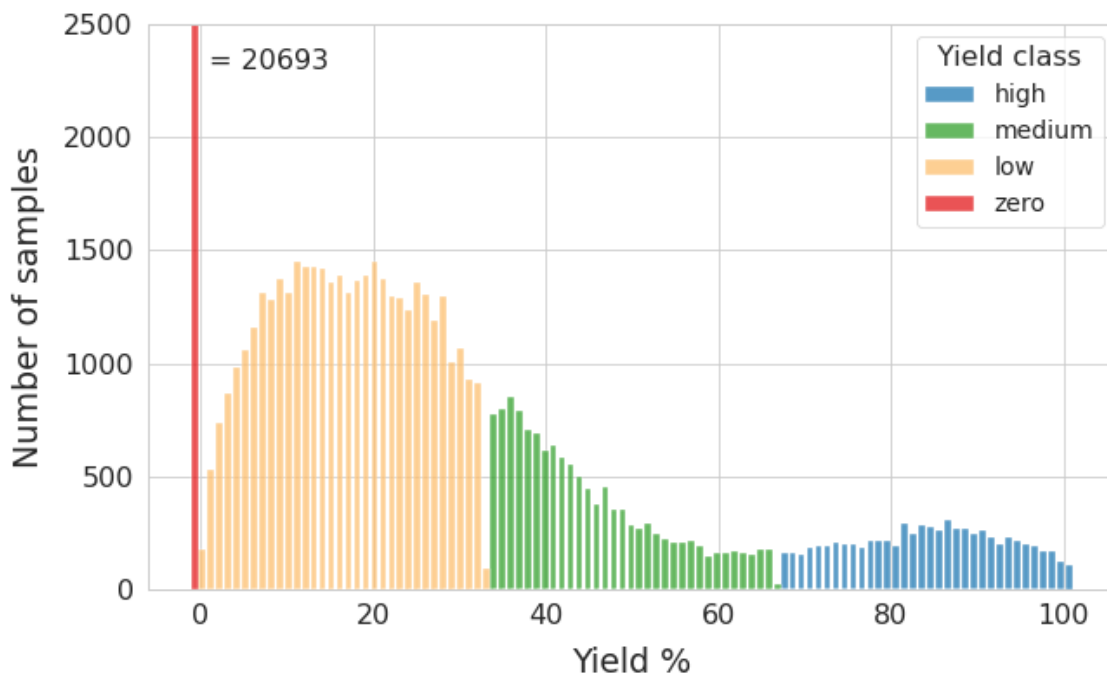


Рис. 3.3. Розподіл рівнів та числових значень фактичних виходів продуктів (Yield) пропріетарного набору даних Enamine [234]: *нульовий* (цільовий продукт реакції не було виділено) - 20693 записів або 25,9% від загальної кількості даних, *низький* (0.1% - 33%) - 39078 записів або 48,9%, *середній* (34% - 66%) - 12937 записів або 16,2%, *high* (67% - 100%) - 7196 рядків або 9% [6].

У 99% реакцій при взаємодії двох або трьох реагентів отримані унікальні продукти. Реагенти включали 27418 унікальних сполук, 80% з яких зустрічалися лише в одному класі реакцій, 14% - у двох класах реакцій і 6% - більш ніж у двох типах реакцій. Ці факти ускладнюють будь-які задачі ідентифікації на таких даних.

Розподіл класів та продуктивностей реакцій показано на рисунку 3.4. Кількість зразків у найбільшому та найменшому класах реакції відрізняється в десятки разів. Реакції середнього та високого виходу відсутні в класі реакції # 7.

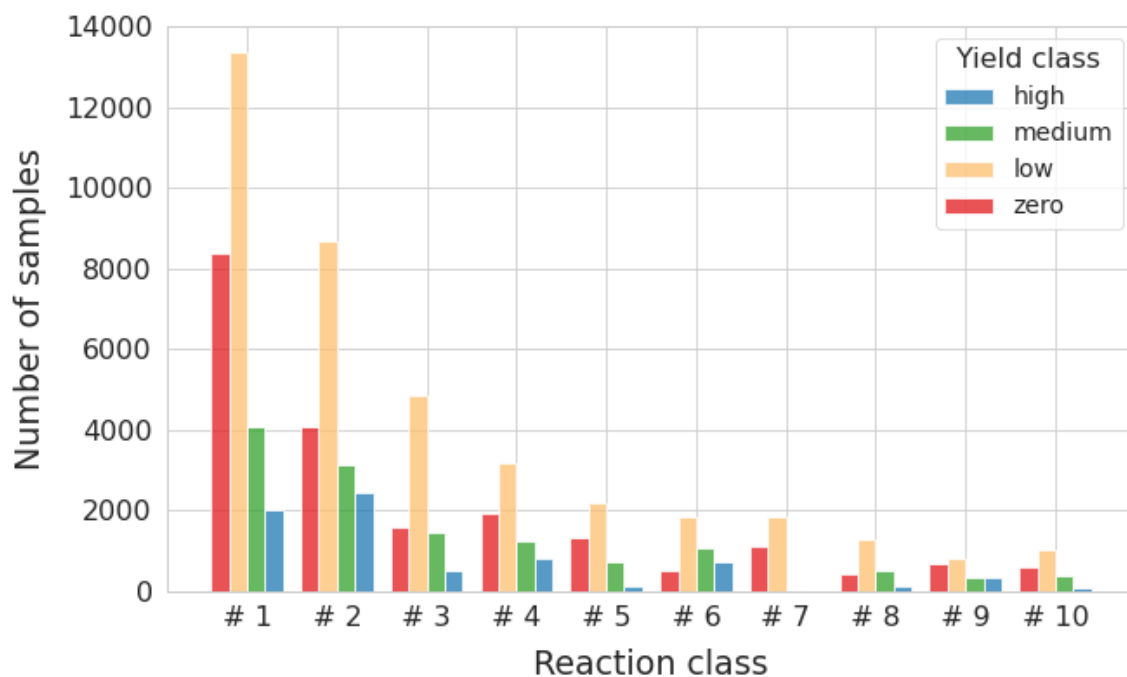


Рис. 3.4. Розподіл фактичних виходів продуктів (Yield) пропріетарного набору даних Enamine [234] за класом реакції: #1 - 27857 зразків або 34,8% від загальної кількості записів; #2 - 18343 зразків або 22,9%; #3 - 8447 зразків або 10,1%; #4 - 7209 зразків або 9%; #5 - 4342 зразків або 5,4%; #6 - 4121 зразків або 5,1%; #7 - 2965 зразків або 3,7%; #8 - 2363 зразків або 2,9%; #9 - 2204 зразків або 2,7%; #10 - 2053 зразків або 2,6% [6].

Кожна реакція належить до одного з наступних широких класів: алкілювання, гетероциклізація, ацилювання, сульфонування та сполучення. Реакції були поділені на десять класів на основі механізму, групи реагентів, продуктів і умов реакції. Схеми реакцій показано на рисунку 3.5.

Важливо вказати на критичну важливість якості даних. Хорошою демонстрацією вирішальної важливості якості даних є набір даних патентного бюро США (USPTO [121]): цей набір даних містить суперечливі записи із різними виходами для однакових реакцій. Крім того, представлені механізми занадто різноманітні і, внаслідок, розріджені. Деякі механізми представлені лише декількома записами реакцій. У цьому відношенні набір даних Enamine [234] відрізняється на краще: типи реакцій обмежені широко використовуваними, а зафіксовані виходи продуктів є відносно узгодженими та зробленими одним виробником у

одній лабораторії. Крім того, набір даних не має дисбалансу щодо реакцій з високим виходом, як це спостерігається у USPTO [121]. З іншого боку, у порівнянні з наборами реакцій одного класу, такими як Бухвальда-Гартвіга [103] і Сузуки-Міяури [104], набір даних Enamine [234] має набагато більшу варіативність реагентів, реагентів і продуктів.

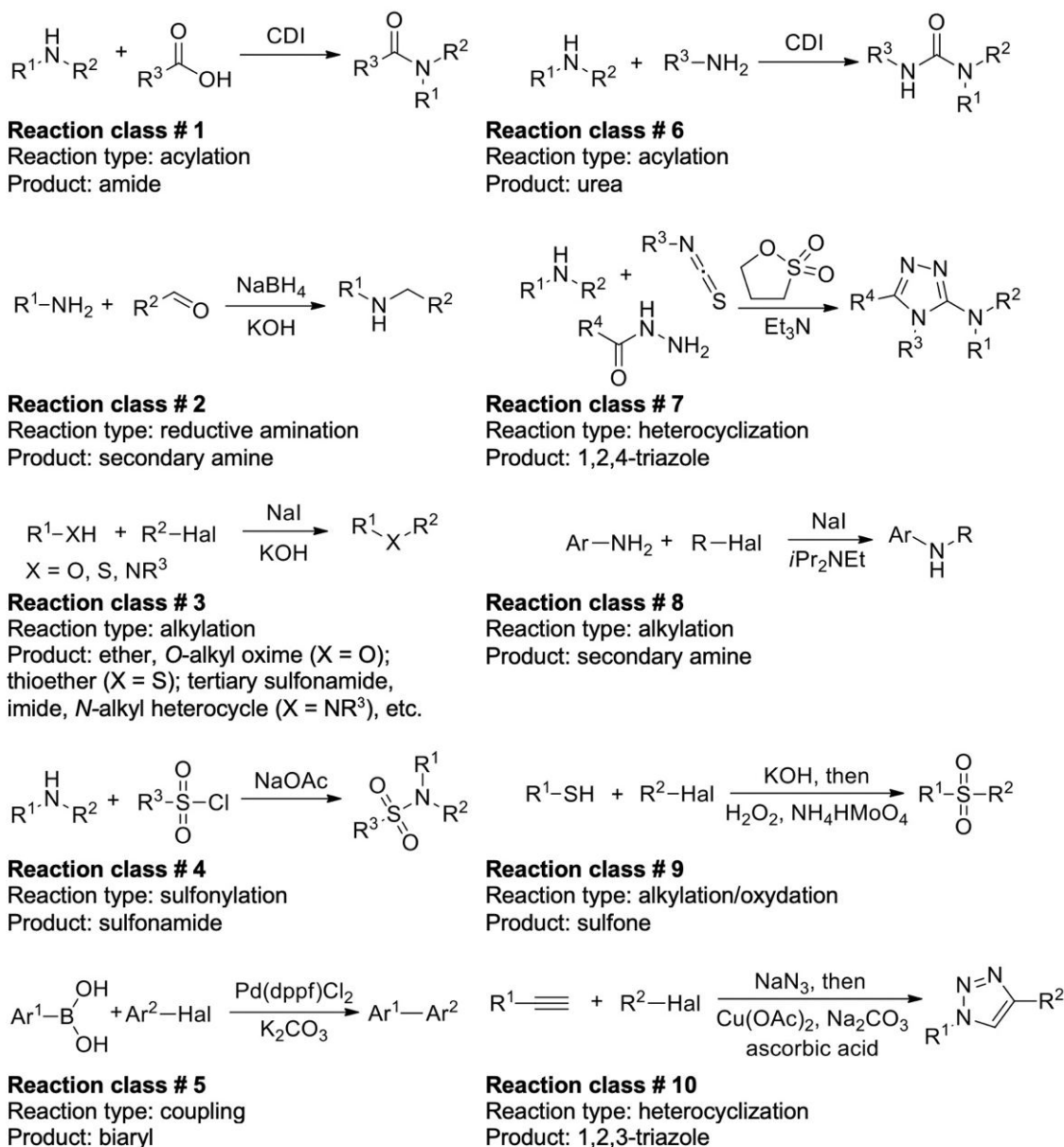


Рис. 3.5. Схеми реакцій набору даних Enamine [234, 6].

На додаток до вищезгаданого, існує багато важливих чинників органічного синтезу, які важко систематично вимірювати та записувати. Так, вихід продукту і навіть перебіг тієї самої реакції залежать від умов навколишнього середовища, таких як вологість, температура повітря, тиск, а також "людських" факторів, таких як метод, час і спосіб очищення,

чистота реагентів та каталізаторів, рівень кваліфікації хіміка тощо. Ці фактори вносять неминучу помилку в будь-яку спробу прогнозувати фактичний вихід продукту хімічної реакції.

3.2 Підготовка даних.

3.2.1 Підготовка даних для прогнозування виходу продукту хімічної реакції.

Для порівняння ефективності створеної графової нейронної мережі (RD-MPNN) було використано загальнодоступні набори даних з одним класом реакцій. Ми відтворили результати групи Schwaller [118] на наборах даних Бухвальда-Гартвіга [103] і Сузукі-Міяури [104] і порівняли їх з результатами RD-MPNN на цих же наборах даних. Усі етапи підготовки даних були виконані відповідно до описаних Schwaller та ін. [118] в усіх експериментах з одним класом реакції.

Пропріетарний набір даних Enamine [234] містив записи десятих класів реакцій. Способи підготовки даних Enamine були тісно пов'язані з природою моделей та описані нижче. Також було прийнято кілька загальних рішень щодо попередньої обробки даних.

По-перше, потрібно було вирішити, чи включати каталізатори, компоненти та препаративні сполуки у вхідні ознаки моделей чи видалити їх, припустивши, що ID класу реакції неявно кодує цю інформацію. У випадку RD-MPNN [6] і CatBoost [212] ми вибрали перший варіант, керуючись дещо вищою точністю моделей, підтвердженою порівняльними випробуваннями з використанням вказаних ознак та без них. Для BERT [214] ми обмежили текстовий рядок реакцій лише реагентами, продуктами та ідентифікаторами класів реакції. Причиною було вкрай незначне покращення точності моделі на більш повних даних одночасно зі значним збільшенням витрат на обчислення. Інша причина полягала в тому, що включення всіх учасників реакції (каталізаторів, адитивів тощо) збільшило довжину об'єднаного рядку реакції до понад 256 токенів максимальної довжини послідовності. Внаслідок, рядки для більшості реакцій потрібно було заповнювати великою кількістю службових символів-"заглушок", що негативно впливало на якість роботи

моделей.

По-друге, проблема незбалансованої класифікації (прогнозування рівня виходу продукту) вимагала спеціальної підготовки даних. Було протестовано наступні стратегії: дублювання рядків для недостатньо представлених класів фактичного виходу продукту; аугментація даних; зважування екземплярів класів; спеціальні функції втрат. Аугментації включали перегрупування реагентів і реактантів, а також запис канонічних SMILES учасників реакції, починаючи з іншого атома в молекулі. У ряді робіт повідомляється, BERT отримує приріст продуктивності завдяки аугментаціям [235]. Тому аугментацію було застосовано для підготовки набору даних для тренування BERT, оскільки ця модель використовує рядки тексту безпосередньо. Аугментації не застосовувалися для інших моделей, оскільки будь-які перестановки символів SMILES у рядку молекули призводять до тих самих відбитків пальців, ембедингів та/або графів. У рамках підходу із введенням вагових коефіцієнтів для записів менш представлених класів, значення функції втрат, розраховане на таких зразках, множилося на коефіцієнти, обернено пропорційні кількості зразків для кожної категорії фактичного виходу продукту. У рамках підходу зі спеціальними функціями втрат, було застосовано Focal Loss [236], реалізований у [237]. Для вибору оптимальних гіперпараметрів, було протестовано значення параметру фокусування (гама) від 2 до 4. За результатами випробувань, усі моделі досягли найкращих результатів за стратегії дублювання зразків менш представлених класів реакцій.

Для перехресної валідації набір даних Enamine було поділено на п'ять частин зі співвідношенням навчальних зразків до тестових 80:20. Рядки у фолдах не перекривалися. Кожен тестовий набір (20%) містив збалансовану кількість рядків для кожної категорії фактичного виходу продукту та для кожного класу реакцій. Також було збільшено кількість рядків у недостатньо представлених категоріях продуктивності та класах реакції (10 шт.) у тренувальних наборах. Для експериментів регресії ми дотримувалися тієї ж техніки збільшення кількості даних, що і для класифікації, але у межах 10% груп кожної категорії значень фактичного виходу продукту для кожного фолду регресії.

3.2.2 Підготовка даних для редуکتивного спрощення.

Набір даних для редуکتивного спрощення складався із 80 014 хімічних реакцій. Кожен рядок відносився до одного із наступних класів реакцій: алкілювання, гетероциклізація, ацилювання, сульфанилювання та сполучення. Реакції були розділені на десять класів на основі механізму, групи реагентів, продуктів і умов реакції. Поля початкових даних включали коди SMILES та InChI цільового продукту; відсоток виходу хімічної реакції; ідентифікатор класу реакції; SMILES та ідентифікатори реагентів; каталізатори та присадки у вигляді рядків SMILES. Було видалено стовпці з кодами InChI та ідентифікаторами реагентів як повторну інформацію. Було також видалено реакції, у яких цільовий продукт не був отриманий (0% вихід). Кількість рядків після підготовки склала 59 291. Значення виходу продукту у відсотках використовувалося як цільова змінна. Розкид значень цільової змінної становив від 0,1% до 100%. Розподіл цільової змінної показано на рис. 3.6.

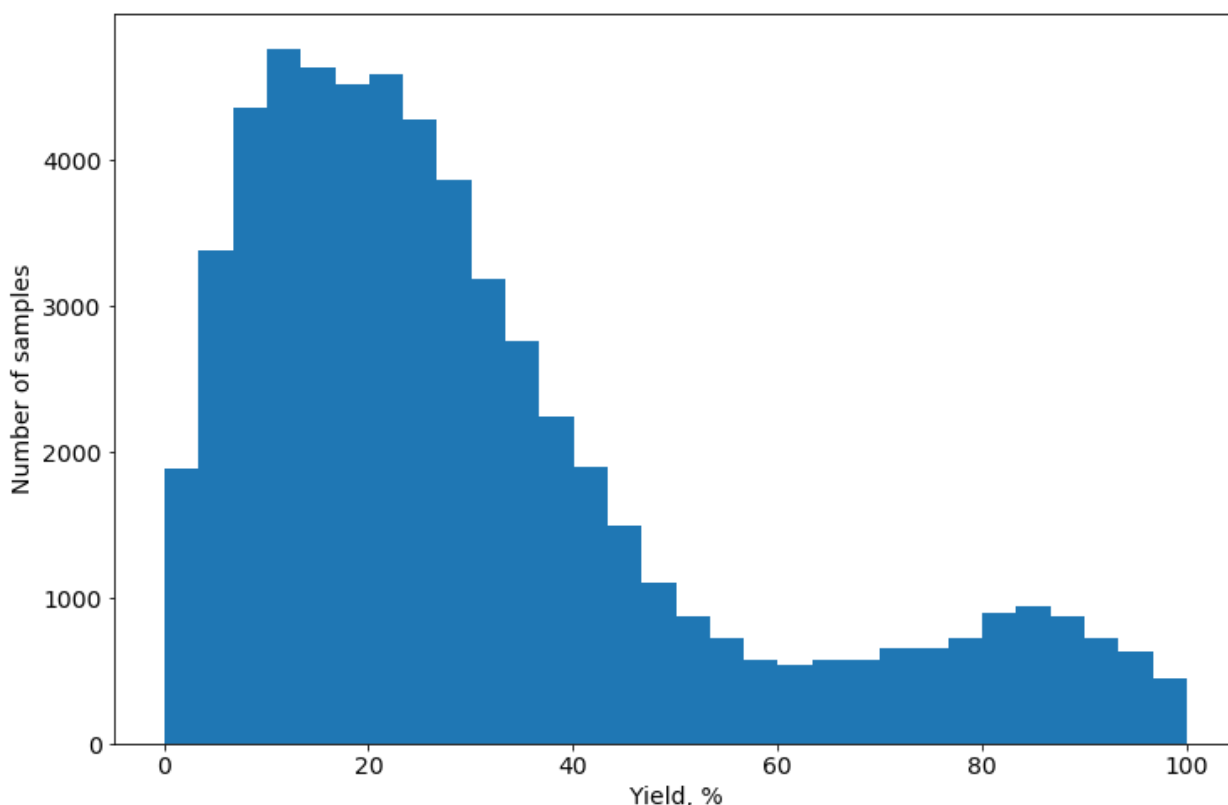


Рис. 3.6. Розподіл значень виходу хімічної реакції [9].

SMILES реагентів і продуктів реакції були закодовані у бінарні відбитки ECFP4, категорійні ознаки (клас реакції, каталізатор і добавки) - як

one-hot вектори. Набір даних було розділено на п'ять фолдів 80:20 для перехресної перевірки.

3.3 Методологія.

Якісні (реакція відбувається чи ні) та кількісні (кількість отриманого продукту) прогнози властивостей хімічних реакцій визначаються різними патернами даних. Тому методологія прогнозу виходу продукту хімічної реакції була розділена на два відповідні етапи: (1) - прогнозування перебігу хімічної реакції (реакція відбувається чи ні) та (2) - прогнозування кількості виходу продукту. Етап (1) вирішувався за допомогою бінарної класифікації. Для цього усі дані були розділені на два великі класи: усі реакції з нульовим виходом, та усі реакції з ненульовим виходом продукту. Кількісний етап (2) включав лише ненульові зразки з першого етапу. Кількісний вихід продукту прогнозувався у відсотках (задача регресії) та категоріях (низький, середній, високий) - тернарна класифікація.

Для вибору найбільш ефективного способу прогнозування кількісного виходу продукту, було порівняно кілька моделей із відповідно підготовленими даними для кожного етапу методології прогнозу виходу продукту хімічної реакції. Моделі включали лінійну та логістичну регресію, опорні векторні машини (SVM) [238], CatBoost [212], представлення двонаправленого кодувальника-трансформера (BERT) [119] та RD-MPNN. Набір протестованих моделей був однаковим для кожного етапу методології, за винятком того, що логістичну регресію було замінено на лінійну регресію у випадку прогнозування відсотку виходу продукту. Навчання та перехресна валідація лінійної/логістичної регресії, SVM і CatBoost виконувалось на ембедінгах SMILESVec [239]. CatBoost - на конкатенованих відбитках пальців розширеного підключення (ECFP) [94]. BERT використовував комбінований SMILES [118] усіх учасників як рядок тексту, без попередньої підготовки, а RD-MPNN [6] – реагенти та продукти, представлені у вигляді графів. Усі зафіксовані результати є усередненими в циклі п'ятиразової перехресної перевірки без перекриття фолдів.

3.3.1 Лінійні моделі.

Ми почали з лінійних моделей як базових для порівняння ефективності інших підходів. Обраними лінійними класифікаторами були логістична регресія та SVM із ядром радіальної базисної функції, реалізованим у sklearn [240]. Ми використовували лінійну регресію для прогнозування відсотку виходу продукту та логістичну регресію для бінарної (продукт отримано чи ні) та тернарної (низький, середній, високий) класифікацій рівня виходу продукту на наборі даних Enamine [234].

Текстові рядки SMILES були закодовані у ембедінги SMILESVec [239]. Це дозволило описати реакцію як конкатенований вектор ембедінгів учасників і продуктів. Вектори ознак для відсутніх учасників були заповнені нулями для уніфікації розмірності вхідних даних.

Підхід SMILESVec [239] описує техніку векторизації з використанням текстового рядка SMILES молекули. SMILES розбивається на підрядки по 8 хімічних символів. Згенеровані підрядки передаються у попередньо навчену модель Word2Vec [241]. Під час тренування, модель Word2Vec вивчає векторне представлення кожного підрядка на основі його сусідів. Відповідна модель Word2Vec була попередньо навчена на величезному корпусі SMILES із баз даних ChEMBL [242, 243] і PubChem [244, 245]. Отриманий ембедінг молекули є усередненим вектором представлених підпоследовностей SMILES. У цьому розділі реакції містили максимум 11 молекул. Отже, довжина конкатенованого Word2Vec вектора реакції становила 1100.

Усі лінійні моделі були навчені з параметрами за замовчуванням. Пошук оптимальних гіперпараметрів не проводився, оскільки лінійні моделі слугували базовими моделями для порівняння.

3.3.2 Моделі на основі градієнтного бустингу.

CatBoost було обрано як зразкову реалізацію градієнтних ансамблів дерев рішень для роботи з категорійними даними. Автор використав CatBoostRegressor [212] для прогнозування відсотку виходу та CatBoostClassifier [212] для бінарної та тернарної класифікацій рівня виходу продукту. CatBoost застосовувався як порівняльний підхід лише до набору

даних Enamine, зокрема до трьох його підмножин: бінарної, тернарної класифікації (CatBoostClassifier) і регресії (CatBoostRegressor).

Ансамблі дерев було навчено та перевірено на ембедінгах SMILESVec [239] та бінарних відбитків Morgan у форматі ECFP [94], описаних у підрозділі 3.2. Обидві методикі описують реакцію як конкатенований бінарний вектор представлень реагентів і продуктів. Вектори ознак для учасників, які не реагують, заповнювались нулями.

ECFP кодують наявність певних функціональних хімічних підструктур і стереохімію молекули. Ця модифікація бінарних відбитків Моргана була розроблена для моделювання структури-властивості [94]. Відбиток молекули генерується рекурсивно на основі властивостей і зв'язності сусідніх атомів. У цьому дослідженні обраний розмір відбитків пальців був 2048, а радіус сприйняття дорівнював 2. Максимальна кількість молекул-учасників реакції у даних була 11. Таким чином, відбиток конкатенованої реакції мав розмір 22 528.

Пошук оптимальних гіперпараметрів було проведено за допомогою Tree-Structured Parzen Estimator, реалізованого в пакеті *hyperopt* [246]. Пошук гіперпараметрів включав глибину дерева, кількість ітерацій і швидкість навчання. Продуктивність моделі оцінювали на тестовій підмножині першого фолду з використанням середньої квадратичної помилки (MSE) для регресії та точності (Accuracy) для бінарної та тернарної класифікацій. Оптимальними виявилися наступні значення: для бінарної класифікації: глибина – 10, кількість ітерацій – 470, швидкість навчання – 0,116; для тернарної класифікації: глибина становила 7, кількість ітерацій – 590, швидкість навчання – 0,143; для регресії: глибина 12, кількість ітерацій 450, швидкість навчання 0,084.

3.3.3 Моделі-трансформери.

BERT — це модель-трансформер, яка використовує токенізовані рядки у текстовому вигляді та не потребує окремої підготовки ознак.

Для порівняльних досліджень на публічних наборах даних з одним класом реакцій ми дотримувалися всіх етапів підготовки даних, навчання та оцінки моделі як описано у роботі групи Schwaller [118]. У цьому випадку ми взяли всі гіперпараметри «як є».

Для експериментів із набором даних Enamine [234] ми перевикористали та дотреноували (fine-tuning) ваги модель BERT [118] для прогнозування значень фактичного виходу продукту [247]. Ми скоротили максимальну довжину послідовності до 256 токенів, оскільки більший розмір був не потрібен. Ми також адаптували параметр швидкості навчання (learning rate) та відсоток "виключення" навчання ваг прихованого шару (dropout) до нашого завдання - 0,001 і 0,2 відповідно, і збільшили розмір батчу з 16 до 32 зразків для стабілізації навчання. Інші гіперпараметри BERT залишилися незмінними.

3.3.4 Графові нейронні мережі.

Message Passing Neural Network (MPNN) [248] - попередник D-MPNN, що належить до сімейства графових нейронних мереж і складається з трьох диференційованих функцій, придатних до навчання: передачі повідомлення (message), оновлення вузла (node update) та зчитування (readout) [248]. Рекурентна функція передачі повідомлення має заданий радіус (фактично, кількість "передач повідомлення" - зазвичай 2 або 3) та включає оновлення ваг ребер і вершин. Під час фази передачі повідомлення приховані стани в кожному вузлі на графі оновлюються на основі повідомлень сусідів. На останньому етапі зчитування обчислюється вектор ознак для всього графу за допомогою функції зчитування.

Directed Message Passing Neural Network (D-MPNN) [249] підтримує два зворотні повідомлення щодо зв'язку між сусідніми атомами A і B: одне "надсилається" від атома A до атома B, інше – навпаки, від атома B до атома A. Отже, замість агрегування інформації від сусідніх атомів, D-MPNN комбінує інформацію з сусідніх хімічних зв'язків [249]. Кожне "повідомлення" від хімічного зв'язку оновлюється на основі всіх вхідних повідомлень від ребер (зв'язків). Завдяки зосередженню на зв'язках і розрізненню між двома напрямками "повідомлень" від зв'язків, D-MPNN має більший контроль над потоком молекулярної інформації. D-MPNN створює більш інформативні молекулярні представлення ніж MPNN [250, 251] і виключає цикли під час передачі повідомлень, які дублюють інформацію для оновлення представлення вузла.

RD-MPNN [6] використовує розширений набір молекулярних

дескрипторів і характеристик хімічних реакцій, заповнює відсутні елементи хімічних реакцій і створює атомні карти реагентів-продуктів для некартованих схем реакцій. RD-MPNN легко адаптувати для прогнозування інших властивостей та умов хімічних реакції. Моделі можна застосовувати для оптимізації типу реагентів, каталізаторів і розчинників, а також концентрації, швидкості додавання, часу, температури або полярності розчинника; прогнозування тепла згоряння, енергії активації, ефективності перетворення та багатьох інших завдань [252, 253, 254, 255, 256].

Робочий процес RD-MPNN включає підготовчий і основний етапи.

Підготовчі етапи включали створення словників атомів (atom mapping) для кожної реакції. Atom mapping були відсутні для наборів даних використаних у роботі, тому автор використав набір інструментів Indigo [257], щоб отримати словники атомів для них. Варто зазначити, що побічні продукти не були включені до записів реакцій в усіх наборах даних. Отже, інколи побудова словнику відповідності атомів реагентів до атомів продукту була неможливою. Тому автор реалізував доповнення для логіки підготовки даних RD-MPNN. Патч автоматично додає відсутні атоми як прості побічні продукти, такі як вода, аміак або окремі елементи. Таким чином, нейронна мережа може вивчати перегрупування атомів у реакціях з відсутньою інформацією про побічні продукти, утворені з цих атомів.

Основні етапи наступні. По-перше, модель зчитує графи реагентів та продуктів та генерує відповідні ембедінги. По-друге, ембедінги вирівнюються та віднімаються відповідно до відображення атомів реакції. Таким чином видаляються атоми, які не беруть участі в хімічному перетворенні та лишається лише інформація, що стосується суті хімічного перетворення: перегрупування атомів у реагентах та продуктах. По-третє, readout шар перетворює отриманий ембедінг різниці в кодування реакції. По-четверте, кодування реакції збагачується додатковими ознаками: ідентифікатором класу реакції та попередньо обчисленими молекулярними дескрипторами реагентів, каталізатора та продукту. Таким чином додається інформація, до якої модель не має доступу під час

навчання лише на графах учасників реакції. Нормалізовані молекулярні дескриптори RDKit2D, як-от молекулярна маса, кількість валентних електронів, максимальний і мінімальний частковий заряд, кількість амідних зв'язків, ароматичні карбо- та гетероцикли тощо, були згенеровані за допомогою пакету DescriptaStorus [258]. Нарешті, розширений відбиток реакції передається до повнозв'язного шару, а згенеровані активації на виході з нього порівнюються зі справжніми значеннями виходу продукту (Yield). Схема RD-MPNN для прогнозування значень фактичного виходу продукту показана на рисунку 3.7.

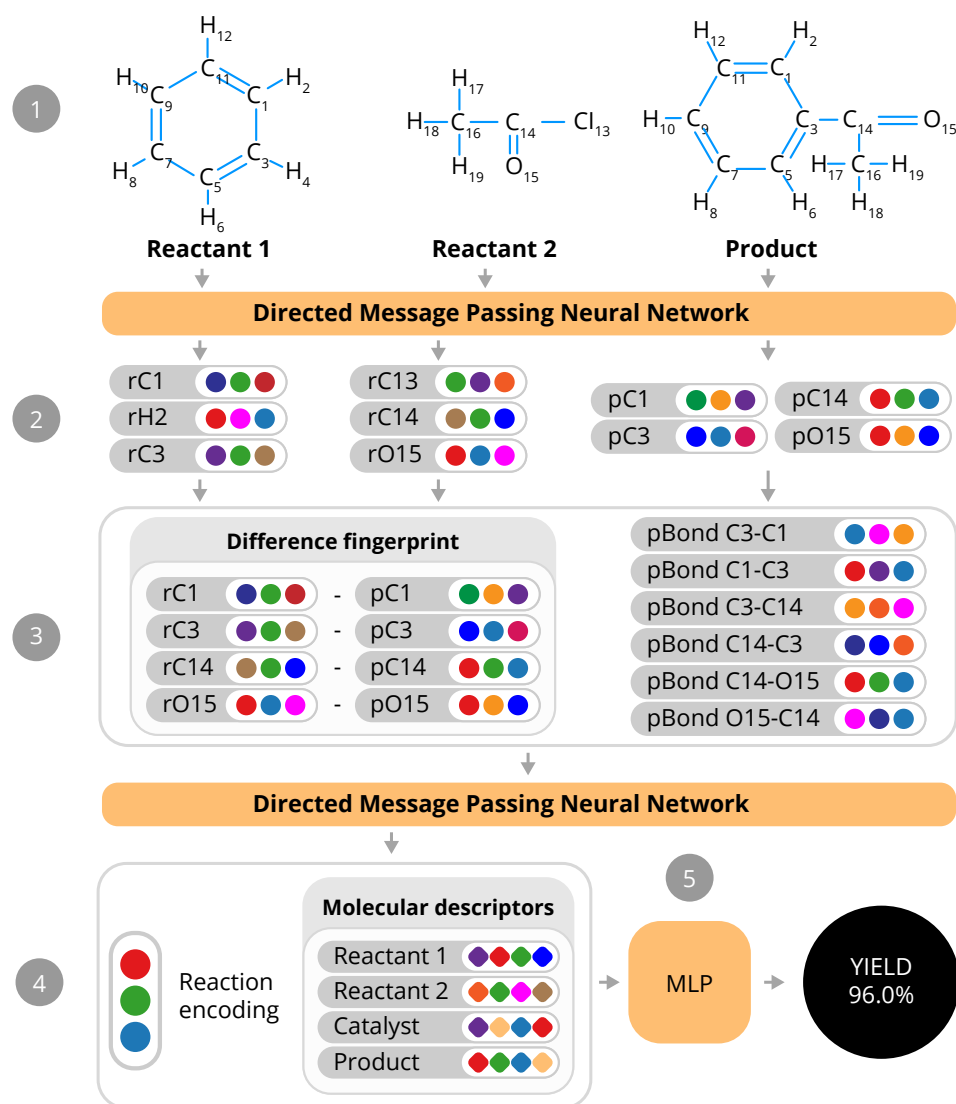


Рис. 3.7. Архітектура RD-MPNN. Підготовчі етапи робочого циклу RD-MPNN включають додавання відсутніх атомів до незавершених реакцій, картографування атомів реагентів і продуктів та генерування графів із рядків SMILES для реагентів і продуктів. Графи передаються до RD-MPNN (1). Згенеровані ембедінги реагентів та продуктів вирівнюються та віднімаються (2). В результаті віднімання отримують ембедінги різниці атомів і зв'язків між продуктами і реагентами. Потім readout шар RD-MPNN перетворює ці відбитки у кодування хімічної трансформації (3). Далі кодування поєднується з реакційними та молекулярними дескрипторами (4) і надсилається на остаточний повнозв'язний шар RD-MPNN (5) [6].

Працюючи з набором даних Enamine, ми виконали оптимізацію гіперпараметрів для RD-MPNN методом пошуку по сітці. Перевірені

розміри прихованого шару обиралися від 256 до 4096 із кроком у ступінь двійки. Визначений оптимальний розмір прихованого шару становив 1024. Оптимізувався також розмір повнозв'язного прихованого шару після шару readout у діапазоні від 256 до 2048 із встановленим найкращим розміром 1024. Оптимальна глибина різниці становила 1. Ми також встановили LeakyReLU як функцію активації, dropout 0,1, розмір батчу 64, кількість епох – 50 із ранньою зупинкою та зниженням коефіцієнту навчання від $1e^{-3}$ до $1e^{-5}$.

У порівняльних експериментах на наборах даних з одним механізмом реакції Сузукі–Міяури [104] і реакції Бухвальда–Гартвіга [103] автор ретельно дотримувався усіх етапів підготовки даних, навчання моделі та перевірки, як описано в [118]. Гіперпараметри RD-MPNN були оптимізовані за допомогою байєсівського підходу, реалізованого в пакеті *hyperopt* [246]. Для реакцій Сузукі–Міяури оптимальними значеннями були наступні: глибина MPNN – 5, глибина різниці – 0, кількість остаточних повнозв'язаних шарів – 2 з прихованим розміром 800, коефіцієнт дропауту – 0.05, розмір пакету – 256. Для реакцій Бухвальда–Хартвіга глибина MPNN становила 3, глибина різниці – 1, кількість повнозв'язаних шарів – 3 із прихованим розміром 1000, без дропауту та з розмір батчу 64.

3.4 Результати.

3.4.1 Прогнозування фактичного виходу продукту для одного класу реакцій.

Для порівняльного аналізу RD-MPNN із кращими результатами та моделями, автор використав набори даних Сузукі–Міяури [104] і Бухвальда–Гартвіга [103]. Гіперпараметри та підготовку даних для RD-MPNN було налаштовано відповідно до методології навчання та перевірки Schwaller та ін. [118] на наборах даних Сузукі–Міяури та Бухвальда–Гартвіга. В обох випадках показники RD-MPNN були усереднені за десятикратним випадковим розподілом трейну до тесту 70:30, подібно до оригінального підходу у роботі Schwaller та ін. [118].

Графова нейронна мережа RD-MPNN досягла нижчого RMSE (10,35) і

вищого R^2 (0,86), ніж кращі референсні моделі Granda та ін. [109] (FCNN) і Schwaller та ін. [118] (Yield-BERT) на наборі даних реакцій Сузукі-Міяури. Зведені результати цього експерименту представлено в таблиці 3.1.

Таблиця 3.1

Зведені результати порівняльного аналізу на наборі даних Сузукі-Міяури [104]. Усі метрики усереднені за десятиразовою перехресною валідацією із відношенням тренувальних до тестувальних даних 70:30 [6].

Метод	RMSE	R^2
FCNN [109]	11.0	-
Yield-BERT [118]	12.07 \pm 0.45	0.81 \pm 0.01
RD-MPNN	10.35 \pm0.43	0.86 \pm0.009

Референсні методи у порівняльних експериментах з використанням даних Бухвальда-Хартвіга включали Ahneman та ін. [103] (RF-DFT) і Schwaller та ін. [118] (Yield-BERT або Y-BERT). На додаток до порівняння моделей оцінених на десятикратних випадкових розподілах 70:30, цей експеримент було доповнено оцінкою роботи моделей на даних поза вибіркою (out-of-sample). Автор перевіряв ефективність RD-MPNN на чотирьох фолдах поза вибіркою. Для підготовки out-of-sample даних, було виділено унікальні ізоксазоліві адитиви у чотири окремі тестові підмножини, а тренувальні підмножини збиралися із решти зразків, у яких вказані адитиви були відсутні. Було навчено і оцінено чотири моделі RD-MPNN на вказаних out-of-sample розбиттях. Результати RD-MPNN на випадкових розподілах 70:30 були на одному рівні з BERT [118]. Валідація RD-MPNN на out-of-sample розбиттях показала нижчу здатність до узагальнення під час випробувань поза вибіркою. Однак варто зазначити, що у даних експериментах молекулярні графи були єдиною інформацією про реакції, яка була доступна для RD-MPNN. Таким чином, результати тестування поза зразком узгоджуються з результатами, отриманими за допомогою інших підходів на основі молекулярних дескрипторів, і ще раз доводять важливість розширення інформації про хімічні реакції для покращення можливостей моделей машинного навчання до узагальнення. Контрольні показники набору даних Бухвальда-Хартвіга наведено в

таблиці 3.2.

Через спеціальне призначення (отримується один і той самий продукт в усіх реакціях), дані Бухвальда-Хартвіга мають низьку різноманітність реактантів і високу різноманітність реагентів. Реактанти - це молекули, атоми яких перегруповуються і утворюють молекулу(и) цільового продукту. Реагенти — це сполуки, які додають до системи, щоб ініціювати, захистити чи пришвидшити хімічну реакцію, але їхні атоми не містяться в продуктах. Прикладами реагентів є адитиви, каталізатори та захисні групи. Це спостереження може допомогти краще зрозуміти отримані результати RD-MPNN на цьому базовому наборі даних. Перевагою RD-MPNN є розширений набір ознак хімічної реакції за рахунок дескрипторів реактантів та продуктів. У цьому експерименті ця перевага нівелюється обмеженням варіативності реактантів та продуктів.

Таблиця 3.2

Зведені результати R^2 референсних експериментів на наборі даних Бухвальда-Хартвіга. Значення R^2 усереднені методом десятиразової перехресної валідації на випадкових розподілах даних 70:30. Випробування поза вибіркою (1-4) оцінюють здатність моделі узагальнювати вивчене на невідомі з тренувальних даних класи ізоксазолових добавок [6].

Метод	Випадкові 70:30	Випробування поза вибіркою				Середнє
	70/30	1	2	3	4	1-4
RF-DFT [103]	0.92	0.80	0.77	0.64	0.54	0.69
Yield-BERT [118]	0.95 ± 0.005	0.84	0.84	0.75	0.49	0.73
RD-MPNN	0.93 ± 0.009	0.63	0.77	0.56	0.48	0.61

3.4.2 Прогнозування фактичного виходу продукту для багатьох класів реакцій.

Як було вказано раніше, методологія роботи мала три підзавдання: бінарна та тернарна класифікації – для категоріальних; регресія - для безперервних значень виходу продукту хімічної реакції. Усі результати у цьому параграфі були отримані на наборі даних Enamine [234], що містить записи десяти типів реакцій. Усі значення метрик були усереднені для п'яти неповторюваних 80:20 поділів навчальних до тестових даних.

У рамках бінарної класифікації було оцінено точність (accuracy), F1 (F1 score) і площу під кривою робочих характеристик приймача (ROC-AUC) для референсних підходів та RD-MPNN [6]. У рамках бінарної класифікації, RD-MPNN досягла точності перехресної перевірки 70%. Цей результат перевищує максимальну відому точність у 65% для набору даних Reaxys [112], але вказані результати важко порівняти через різні розміри наборів даних і різну кількість представлених механізмів реакції. Експерименти з тернарною класифікацією дали найкращу точність перехресної перевірки 51%. Отриманий найкращий коефіцієнт детермінації (R^2) для прогнозів значень фактичного виходу продукту становив 0,18, що можна порівняти з оцінками USPTO R^2 - від 0,095 до 0,388 [118]. Усі результати зведені в таблицю 3.3.

Таблиця 3.3

Показники десятиразової перехресної перевірки моделей для набору даних Enamine [234, 6].

Метод	Бінарна			Тернарна	Регресія	
	Точність	F1	AUC	Точність	MAE	R^2
Logistic Regression	0.58	0.59	0.58	0.41	23.2	0.1
SVM SMILESVec	0.64	0.66	0.64	0.47	23.01	0.0
CatBoost ECFP	0.62	0.62	0.62	0.49	25.6	0.17
CatBoost SMILESVec	0.62	0.62	0.62	0.46	25.4	0.18
BERT	0.67	0.71	0.73	0.46	21.82	0.12
RD-MPNN	0.7	0.74	0.78	0.51	20.86	0.16

RD-MPNN демонструє найкращу продуктивність серед розглянутих моделей завдяки розширеній інформації про учасників і продукти реакції завдяки поєднанню молекулярних дескрипторів і особливостей реакції з графовими представленнями учасників та продуктів реакції. RD-MPNN вивчає найнадійнішу межу прийняття рішень для бінарної класифікації (див. таб. 3.3), що виражається як найвище значення ROC-AUC.

Аналіз помилок тернарної класифікації на рисунку 3.8 показує, що RD-MPNN перевершує інші моделі в розпізнаванні груп низького/середнього/високого виходу продукту реакцій.

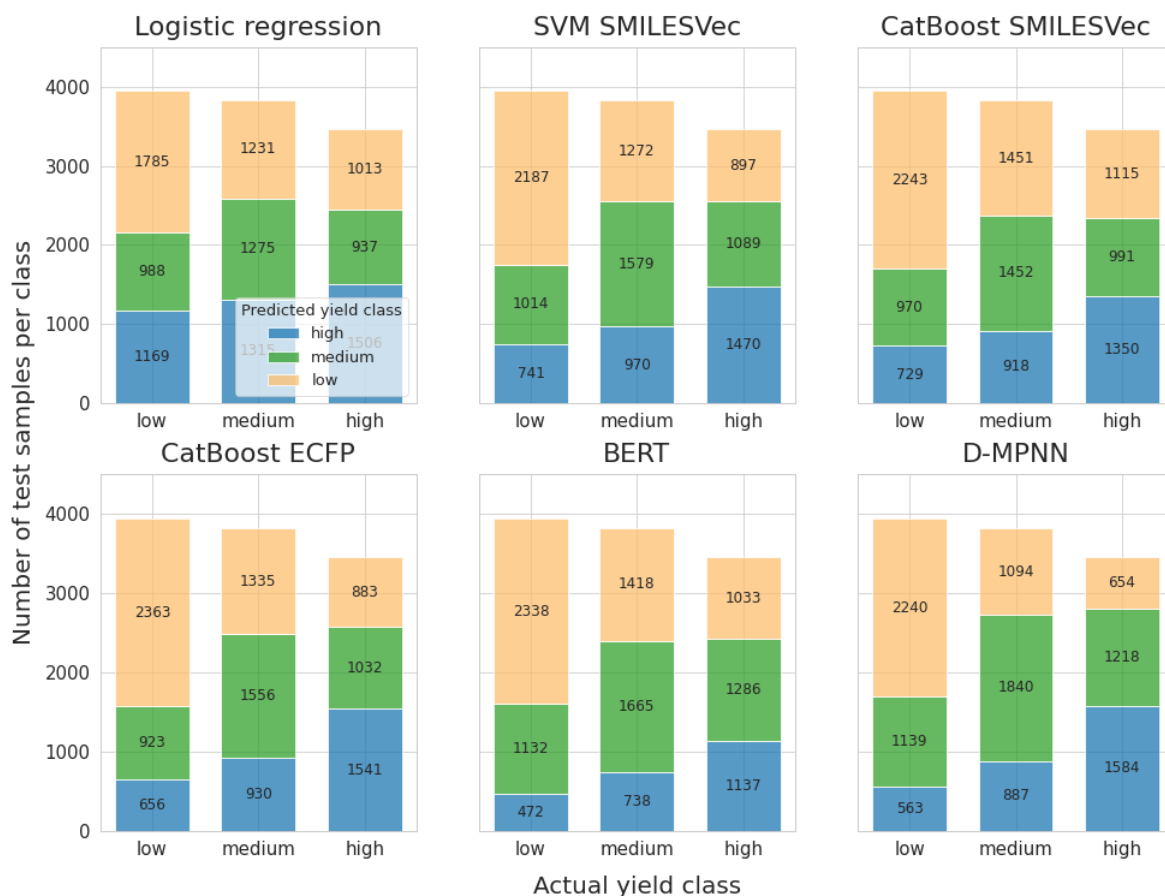


Рис. 3.8. Розподіл помилок тернарної класифікації за класом реакції [6].

Для глибшого аналізу способу розпізнавання різних класів реакцій у прогнозуванні виходу, на рисунку 3.9 показано точності за кожним із класів реакції для задачі бінарної класифікації виходу продукту. Добре видно, що різниця в точності між найкращим і найгіршими прогнозами виходу для окремих типів реакції становила приблизно 12%, незалежно від моделі.

Найкращі результати були отримані для гетероциклізації - класи # 7 і # 10. Примітно, що # 7 — це єдиний тип реакції, який включає три реагенти та має лише два класи виходу — «низький» і «нульовий» (див. рис. 3.4). Крім того, всі моделі показали низьку ефективність для синтезу сечовини - ацилювання, клас реакції # 6. Одним із можливих пояснень є те, що клас # 1 також є ацилюванням, і він містить у шість разів більше записів, ніж # 6. Таким чином, моделі перевивчають тип # 1 і передають його представлення формально подібним # 6 реакціям. Відповідно, для класу реакцій # 6 точність погіршилася.

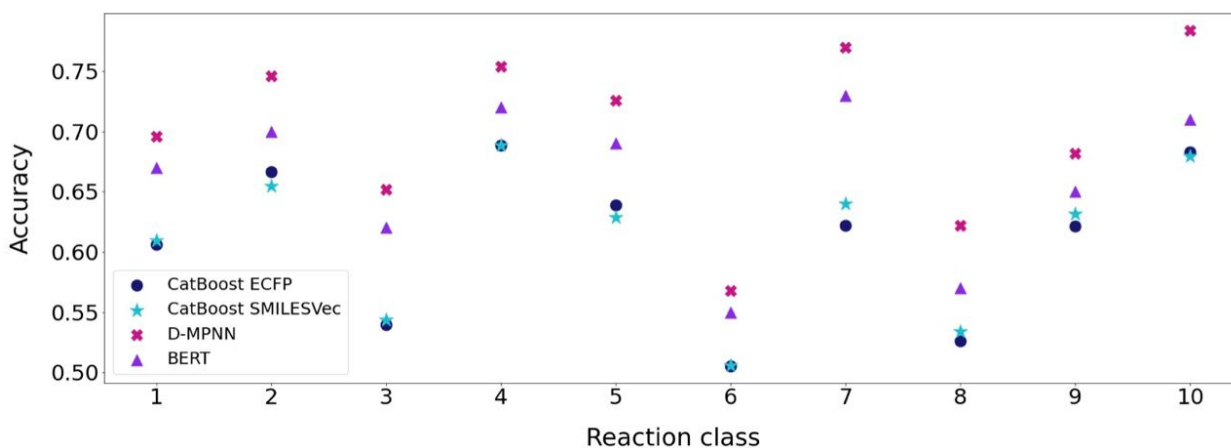


Рис. 3.9. Точність прогнозування виходу продукту за класом реакції в бінарній класифікації. Типи реакцій розташовані в порядку спадання за кількістю зразків [6].

3.4.3 Аналіз помилок прогнозування фактичного виходу продукту.

Легко помітити, що кількість зразків на тип реакції не вплинула на продуктивність моделей за цим типом реакцій. Це вказує на те, що обмеження точності походять від інших факторів - не від кількості даних. Цікаво, що нижча точність бінарної класифікації найефективнішої моделі RD-MPNN для різних класів реакцій корелює з підвищеною частотою хибнопозитивних результатів і навпаки. Це важливо для типу реакції # 6 - відповідні значення для хибнопозитивних і негативних рівнів становлять 42% і 1,6%, відповідно. Подібна ситуація спостерігається і для інших механізмів із низькою ефективністю — наприклад, # 3 і # 8, з частотою хибних позитивних/негативних результатів 30% / 5% і 31% / 7%, відповідно. Порівняйте ці значення з 10% / 12% і 13% / 8% для класів реакції # 7 і # 10 відповідно.

Може бути кілька загальних причин спостережуваних явищ.

По-перше, усі дані включали лише спрощені хімічні структури в нотації SMILES. Однак, кількісні та якісні результати органічного синтезу не визначаються лише хімічною будовою учасників. Синтез є складним явищем, на яке впливають численні параметри, окрім хімічної структури учасників і продуктів. Ці параметри включають ефективність ізоляції та очищення. Останнє визначається не тільки хімічною будовою

продукту, але і його фізико-хімічними властивостями, а також вмістом і природою забруднюючих речовин. Навіть так званий «людський фактор», тобто допущені помилки експерименту чисто випадкового характеру, що призводять до збоїв синтезу, обмежують ефективність прогностичних моделей, оскільки порушують наявні патерни у даних випадковими стохастичними збуреннями.

По-друге, такі умови як вологість, температура, барометричний тиск та інші чинники зовнішнього середовища, можуть непередбачувано вплинути на вихід синтезу. Відсутні молярні еквіваленти обмежували розуміння кількісного співвідношення реагентів і утворених продуктів у наборі даних Enamine [234].

Була здійснена спроба знайти деякі з вищезгаданих ефектів, застосовуючи структурні та фізико-хімічні фільтри. Однак статистично значущих відхилень у хибних позитивних та хибних негативних показниках не виявлено для гідрофільних сполук (розрахований $\text{LogP} < 0$), сполук з основними центрами, гідроксигрупами, NH-вмісними азолами чи іншими донорами водневих зв'язків (ці властивості часто ускладнюють виділення/очищення продуктів). Деяко нижча точність прогнозування спостерігалася для сполук без ароматичних кілець (64%; хибнопозитивні/негативні показники: 28% / 8%). Це може бути пов'язано зі складнішим хроматографічним очищенням продуктів через недостатність або відсутність УФ-хромофорів.

Тому було проведено більш ретельний аналіз експериментальних результатів реакцій, які були визначені як хибні позитивні у бінарних прогнозах RD-MPNN і мали найвищі значення сигмоїди. Дійсно, серед продуктів цих реакцій були виявлені сполуки, які

- були отримані з високим виходом згідно зі спектральними даними для неочищеного продукту, але були втрачені під час хроматографічного очищення: Рисунок 3.10.

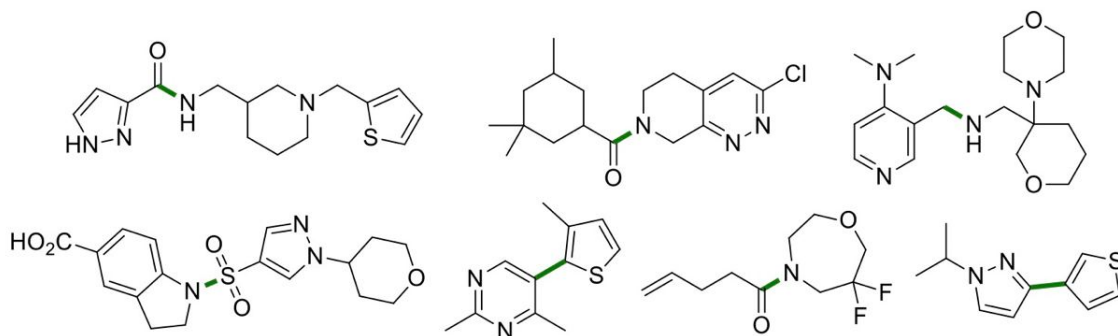


Рис. 3.10. Продукти, які не вдалося виділити під час хроматографічного очищення [6].

- не були отримані, ймовірно, через стеричний фактор: рис. 3.11.

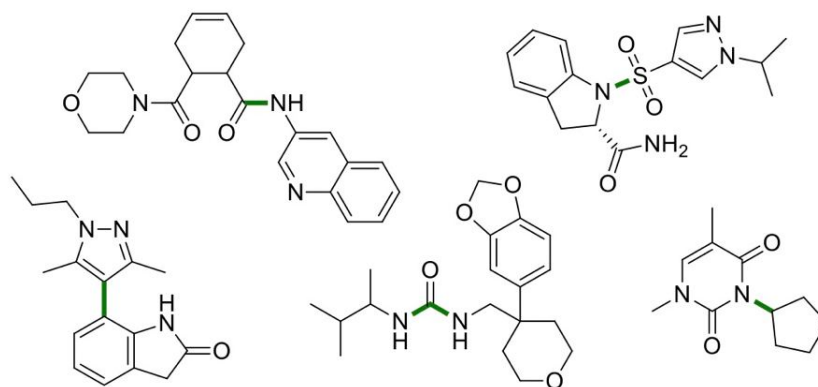


Рис. 3.11. Продукти, які не були отримані, ймовірно, через стеричний фактор [6].

- не були отримані, ймовірно, через побічні реакції на додаткову функціональну групу (показано червоним кольором): рис. 3.12.

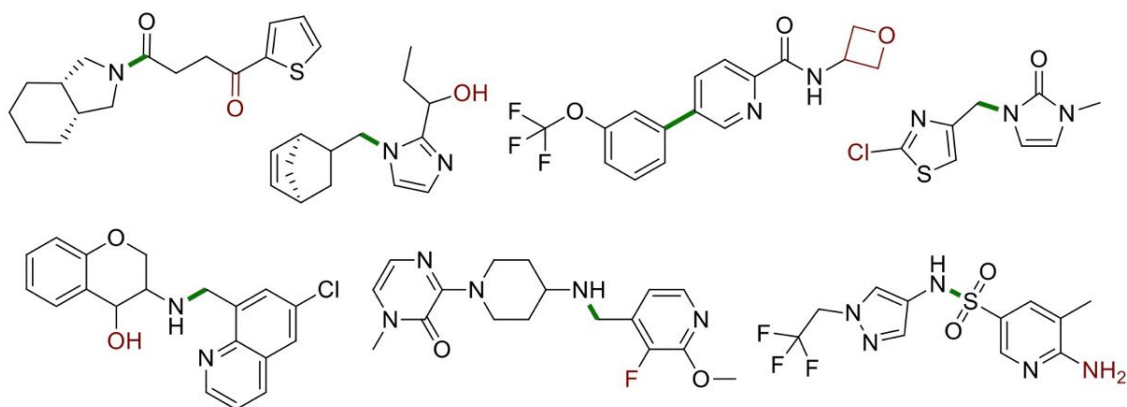


Рис. 3.12. Продукти, які не були отримані, ймовірно, через побічні реакції на додаткову функціональну групу [6].

- не були виділені після першого експерименту, але виділені при повторному експерименті: рис. 3.13.

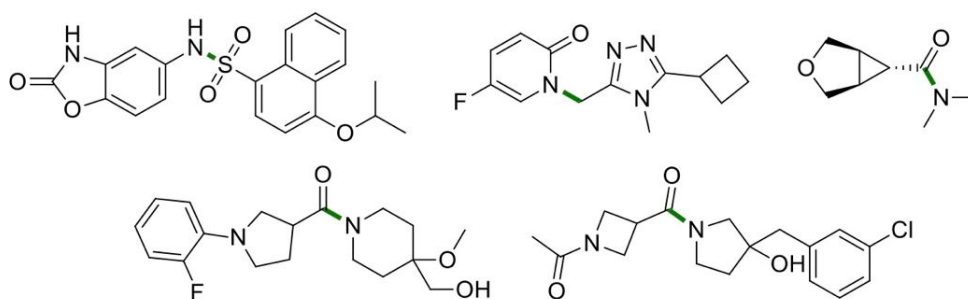


Рис. 3.13. Продукти, які не були виділені після першого експерименту, але були виділені при його повторенні [6].

- були утворені, але була допущена помилка¹ в інтерпретації даних спектроскопії: Рис 3.14.

¹У робочому процесі паралельного синтезу Епаміне початковий результат реакції визначається на основі спектральних даних (LC-MS) людиною-оператором, який приймає рішення «так/ні» для початку етапу очищення. Були перевірені спектри серії прикладів, коли реакція мала відбуватися, але записи дій операторів показали, що це не так. Внаслідок, знайдено два приклади, де спектральні дані були невірно інтерпретовані оператором.

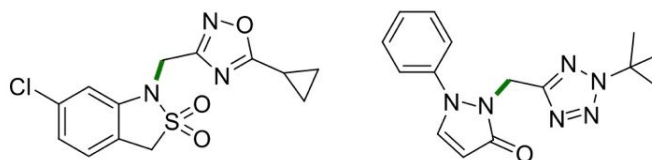


Рис. 3.14. Продукти були утворені, але була допущена помилка в інтерпретації спектральних даних [6].

3.4.4 Прогнозування виходу продукту для класу реакції поза тренувальною вибіркою.

Для подальшого дослідження узагальнюючих здатностей RD-MPNN на нові класи реакцій (відсутні у тренувальних даних), мережу ітеративно навчали на зразках дев'яти типів реакцій, залишаючи один, для тестування. Щоб оцінити здатність моделі узагальнювати хімічні знання щодо різних реагентів, ми перемістили загальні реагенти для тестування та навчання до валідаційної підмножини. Результати наведено в таблиці 3.4.

Таблиця 3.4

Здатності RD-MPNN до узагальнення на нові реагенти та класи реакцій на наборі даних Enamine [234]. Для кожного типу реакції порівнювалася точність RD-MPNN на змішаних тестових даних (валідація) та на зразках одного невідомого з тренувальних даних класу (тест). У нижньому рядку таблиці наведені нормалізовані розміри підмножин у відсотках [6].

	Точність на клас реакції:									
	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9	# 10
Валідація	0.70	0.70	0.71	0.70	0.72	0.73	0.71	0.72	0.72	0.71
Тест	0.52	0.41	0.65	0.44	0.66	0.68	0.63	0.63	0.61	0.40
К-ть даних,%	34.8	22.9	10.1	9	5.4	5.1	3.7	2.9	2.7	2.6

Здатності RD-MPNN до узагальнення на новий клас реакцій є відносно хорошим у більш широких групах реакцій, таких як алкілювання (механізми # 3, # 8 та # 9) та ацилювання (механізми # 1 та # 6). Однак «унікальні» механізми реакції (# 2, # 4 і # 10) розпізнаються гірше. Крім того, гетероциклізація (# 7) була винятковою. Оскільки одним із

критичних етапів у цьому процесі є утворення тіосечовини [259], можна очікувати деяку схожість типів реакцій # 7 і # 6 — і, як результат, краще узагальнення цих механізмів.

Загалом наведені вище результати демонструють, що завдання прогнозування виходу продукту для широкого діапазону типів реакцій залишається складним навіть для найдосконаліших методів машинного навчання. Як підкреслювалося раніше [112], це вказує на те, що інформація, включена в загальні записи реакцій, не містить усіх факторів, які визначають фактичний вихід продукту хімічної реакції.

3.4.5 Зменшення розмірності відбитків реакцій і аналіз проєкцій

Як було описано раніше, RD-MPNN конструює реакційний відбиток із різниці ембедінгів атомів, які беруть участь у хімічному перетворенні. Було перевірено чи вивчила модель доцільні представлення реакцій. З цією метою було згенеровано ембедінги реакцій, використовуючи перший тестовий фолд набору даних Enamine [234], та зменшено їх розмір із початкових 1024 до 2, використовуючи t-розподілене стохастичне вбудовування сусідів (t-SNE) [260].

За результатами оптимізації гіперпараметрів методом пошуку по сітці зі значеннями від 10 до 500 було обрано гіперпараметр методу t-SNE perplexity рівний 126. Також було встановлено early exaggeration 40, керуючись найкращим візуальним результатом у кількох варіантах значень цього гіперпараметру. Інші параметри було залишено за замовчуванням. Результати показано на рисунку 3.15. Кольори та відповідні номери на легенді справа стосуються справжніх класів реакції.

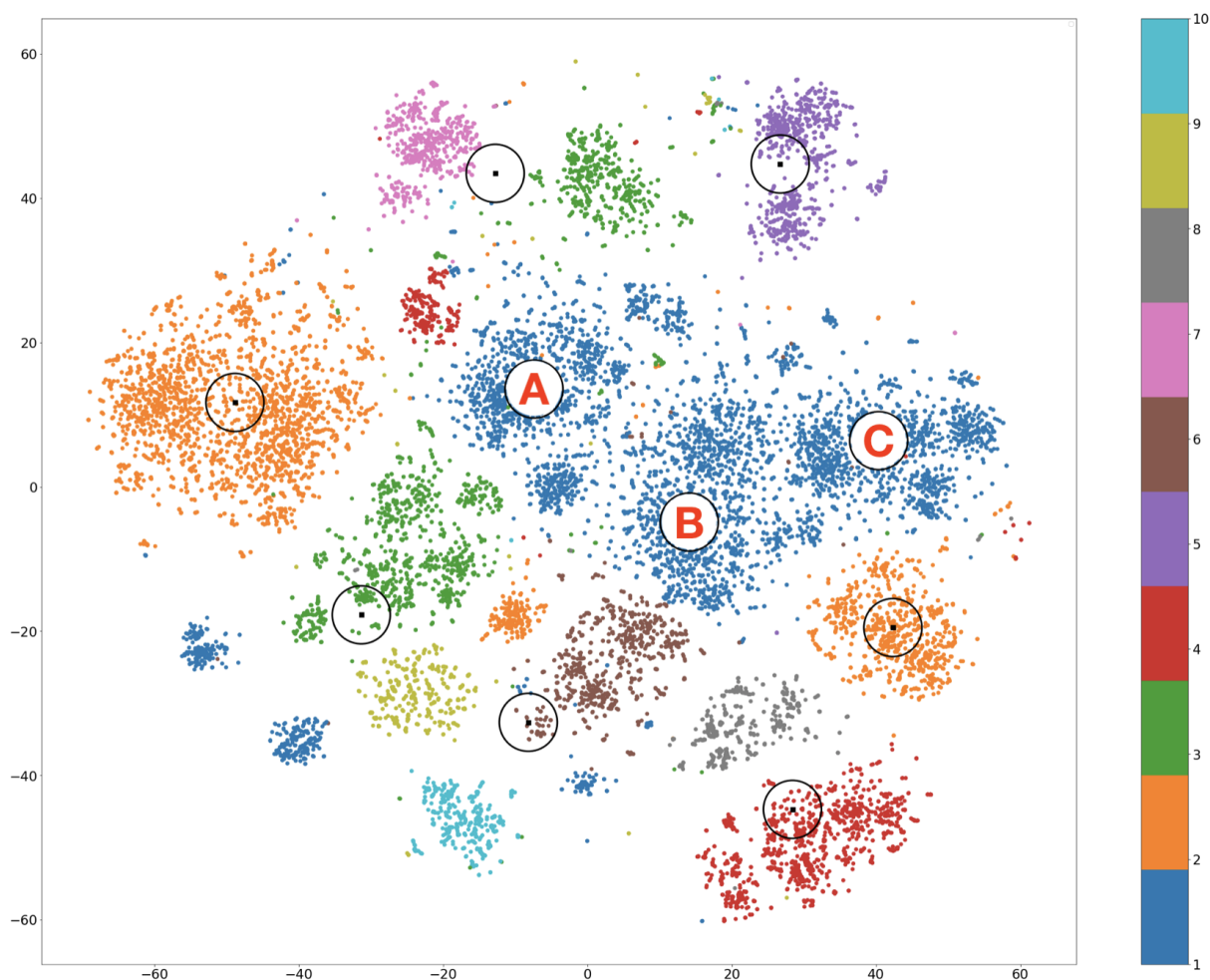


Рис. 3.15. Справжні класи реакцій позначені кольорами на 2D t-SNE ембедінгах активацій передостаннього шару RD-MPNN [6].

Далі було застосовано k-середні (k-means) для кластеризації отриманих двовимірних ембедінгів t-SNE. Кількість центроїдів становила 10 - відповідно до кількості класів реакцій у наборі даних Enamine. Класи реакцій не були закодовані у вхідних даних (рядки SMILES) до шарів графів RD-MPNN (розширені ознаки подаються у нейронну мережу після шару для якого візуалізовані ембедінги). Тому метою аналізу було перевірити чи навчилася модель розпізнавати класи реакцій і чи були вивчені моделі даних хімічно обґрунтованими. Результати показано на рисунку 3.16. Зверніть увагу, що кольори кластерів на останніх двох малюнках не збігаються. Кластери та кольори на рисунку 3.16 були розподілені за k-середніми, тоді як кластери на малюнку 3.15 походять від справжніх ідентифікаторів реакцій.

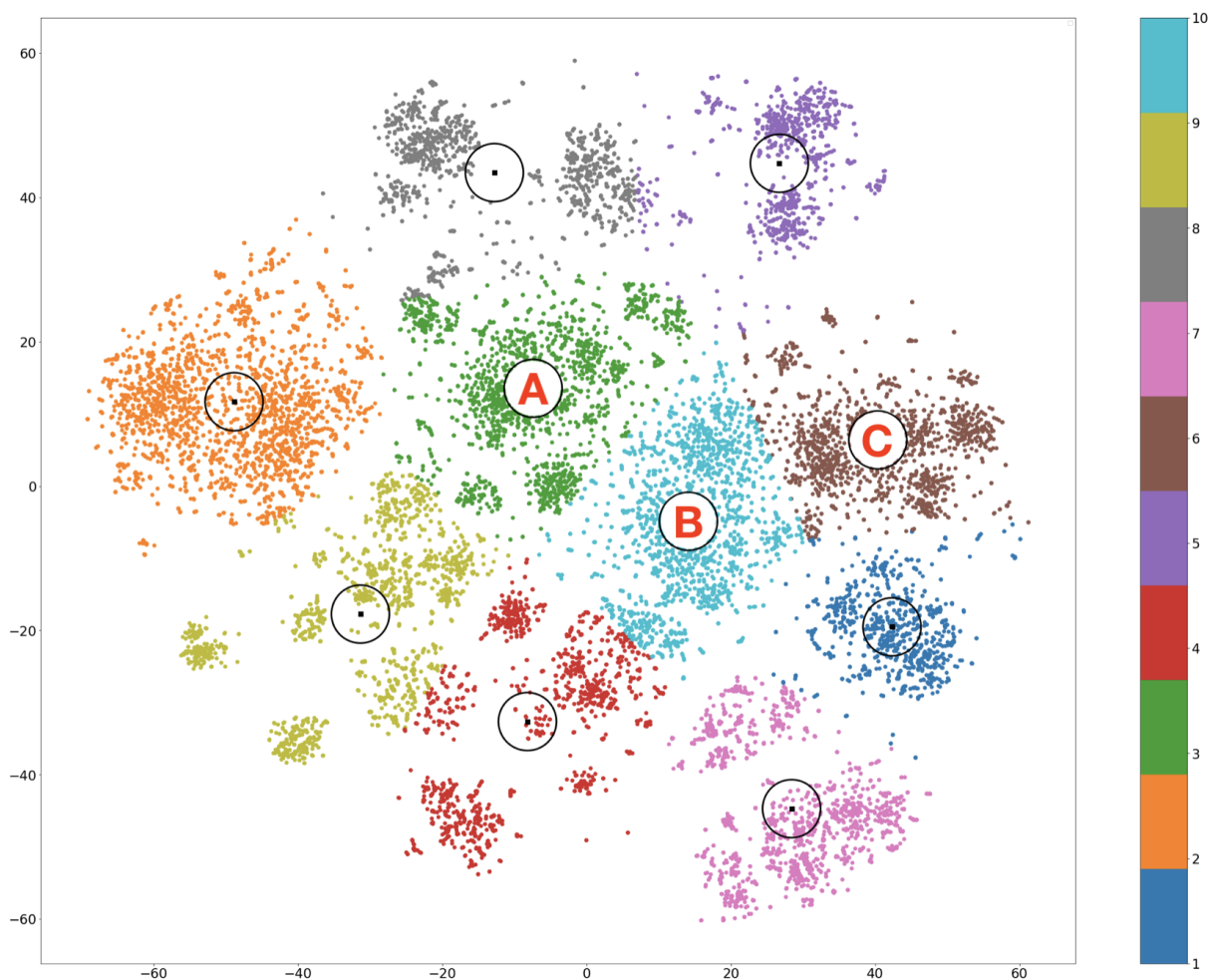


Рис. 3.16. Прогнозовані класи реакцій позначені кольорами на 2D t-SNE ембедінгах активацій передостаннього шару RD-MPNN [6].

Розглядаючи структуру кластерів на малюнку 3.16, можна побачити, що більшість зразків одного класу реакцій були вірно кластеризовані разом за допомогою k -середніх. Тому більшість вивчених кластерів відповідають справжнім кластерам. Винятком є найбільший кластер типу реакції # 1, розділений на три чітко спостережувані конгломерати (A, B і C). Щоб проаналізувати відмінності між конгломератами кластеру класу реакції #1, було відібрано точки на основі їх відстані до центроїдів. Керуючись здоровим глуздом, автори вибрали 16% найбільш близьких до центроїдів вибірок. Аргументація полягала в тому, що менша кількість записів не буде репрезентативною, тоді як більша кількість буде занадто розпливчастою і її важко буде переглянути вручну, щоб зробити будь-які висновки.

Аналіз 2600–2900 відповідних реакцій показав, що їх значні фракції (53%, 52% і 62% відповідно) мають спільні структури. Додаткова

ілюстрація хімічно обґрунтованих закономірностей, отриманих моделлю, показана на рисунку 3.17. Ця кількість спільних підструктур показує, чому зразки класу реакції # 1 були віднесені до однієї групи (див. рис. 3.15). Решта різномірних підструктур (47%, 48% і 38% відповідно) можуть пояснити, чому 1-й клас був поділений на 3 групи у вивчених представленнях реакцій.

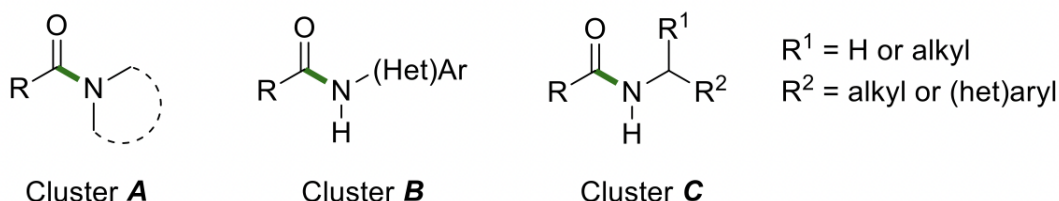


Рис. 3.17. Загальні підструктури навколо центроїдів 3 підгруп всередині класу реакції #1 [6].

3.4.6 Редуктивне спрощення моделей оцінки виходу продукту хімічної реакції.

Метод редуктивного спрощення [230, 231, 232, 233, 9], описаний у підрозділі 2.2.8, був застосований для спрощення повнозв'язних нейронних мереж з одним прихованим шаром для прогнозування виходу продукту хімічної реакції. Початкова мережа мала 16 799 744 параметри. Метрики та функція втрат були такими ж, як і в підрозділі 2.3.3. Еволюція функції втрат і метрик валідації на тестовому наборі даних поза основною вибіркою показані на рис. 3.18.

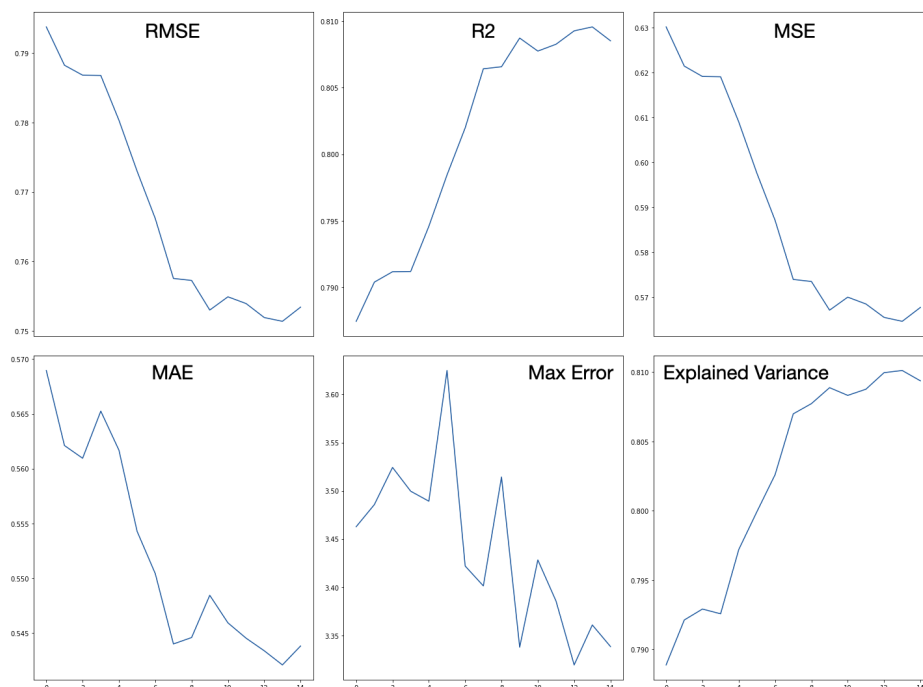


Рис. 3.18. Еволюція метрик спрощених моделей CM на out-of-sample тестовому наборі даних (20%) під час редукції ваг мережі для прогнозування відсотку виходу продукту хімічної реакції [9].

Редуктивне спрощення було зупинено після 13-ї епохи через погіршення тестових втрат з N менше за 0,01%. Фінальна маска була повернута до маски 12-ї епохи. У таблиці 3.5 порівнюються початкова і кінцева генералізація моделей і кількість ваг.

Таблиця 3.5

Порівняння повної та спрощеної FFN для прогнозування виходу продукту хімічної реакції [9].

Метрика	Повна модель	Спрощена модель	Зміна, %
RMSE (ф-я втрат)	21.83843	21.12756	-3.26
R^2	0.29373	0.33907	+15.43
MSE	477.0122	446.3936	-6.42
MAE	16.92516	16.41234	-3.03
Найбільша помилка	70.97618	68.92058	-2.9
Пояснена дисперсія	0.29471	0.33930	+15.3
Активні ваги	16,799,744	11,891,065	-29.21

Легко помітити, що спрощена мережа (70.79% залишкових ваг) має

кращу здатність до узагальнення у порівнянні із початковою, повною мережею (R^2 +15,4%, RMSE -3,26%) - замість погіршеної, як це зазвичай спостерігається у класичних методах спрощення (див. Розділ 1).

3.5 Висновки.

У цьому розділі розроблено новітню графову нейронну мережу для прогнозування властивостей хімічних реакцій (RD-MPNN). Модель використовує графи молекул та хімічні дескриптори. RD-MPNN показала найкращі результати серед оцінюваних моделей на одному пропрієтарному та двох публічних наборах даних, продемонструвавши вдале поєднання фізико-хімічних властивостей молекул і реакцій та графових представлень учасників реакції.

Спрощення повнозв'язної мережі оцінки відсотку виходу продукту хімічних реакцій дозволило покращити значення функції втрат (RMSE) на 3.26% та коефіцієнту детермінації (R^2) на 15.43% із 70.79% залишкових ваг.

4 РОЗДІЛ 4. МЕТОД РОЗРОБКИ ЛІКАРСЬКИХ МОЛЕКУЛ ІЗ ЗАДАНИМИ ВЛАСТИВОСТЯМИ.

У цьому розділі описано новий метод, у якому кілька глибоких нейронних мереж поєднуються для розробки лікарських речовин із заданими властивостями. Створення молекулярних структур доповнюється виправленням помилок хімічної будови рекурентною нейронною мережею з механізмом уваги. Для створених структур проведено аналіз таких фізико-хімічних властивостей. Стабільність та передбачену розчинність ($\log S$) згенерованих молекул підтверджено методами молекулярної динаміки. Результати розділу опубліковано у працях автора [8, 7].

4.1 Опис та підготовка даних.

4.1.1 Спосіб представлення молекул.

Існують різні представлення молекул, які дозволяють кодувати просторову структуру за допомогою компактних однорядкових позначень. Найпопулярнішими серед них є SMILES [49]. SMILES містить всю необхідну інформацію для обчислення необхідних метрик (донори Н-зв'язку, акцептори, молекулярна маса тощо), за винятком ліпофільності та розчинності у воді. Рядок SMILES не може бути поданий в нейронну мережу у вихідній формі і має бути виражений у числовому вигляді. Категоріальні дані (наприклад, символи SMILES) зручно представляти за допомогою так званого one-hot кодування, яке у випадку SMILES є матрицею (N на M) із 0 і 1 у комірках. N – це кількість унікальних елементів SMILES (наприклад, C, c, =, @, O, дужок тощо), тоді як M позначає символи рядка SMILES. Один з таких прикладів проілюстрований на рис. 4.1, де показано one-hot кодування пропіонового альдегіду. Ми розглядаємо рядки SMILES не довше 60 елементів, застосовуючи заповнення нулями для коротших. Також було обмежено розмір словника елементами, які зустрічаються в наборах даних для тренування та оцінки (58 унікальних символів).

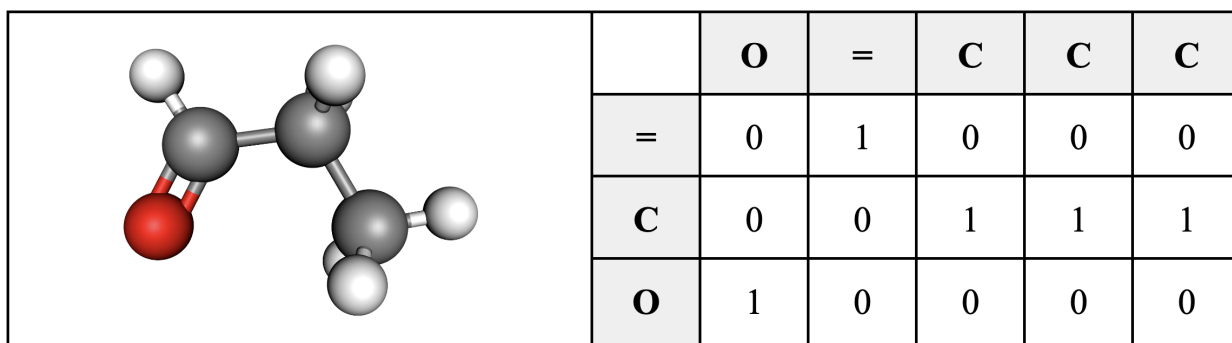


Рис. 4.1. Просторова конфігурація пропіонового альдегіду, SMILES-представлення та відповідна one-hot матриця. Вуглеці (C) на тривимірному зображенні показані сірим, кисень (O) – червоним, водень (H) – білим [7].

4.1.2 Підготовка даних для моделі-генератора молекулярних структур.

Для попереднього тренування автокодувальника було використано 500 000 випадково обраних SMILES із публічної бази даних ChEMBL [242]. Для навчання моделі реконструкції цих молекулярних представлень, на вхід і на вихід подавалося по два однакові рядки SMILES.

Як приклад фізико-хімічної властивості для контролю, було обрано розчинність у воді (logS). Для дотренування автокодувальника із додатковим модулем для передбачення розчинності було зібрано та об'єднано дані із серії відкритих наборів, опублікованих Huuskonen [261], Hou [262], Delaney [263] та Mitchell [264]. Крім того, набір даних про розчинність було розширено шляхом приведення рядків SMILES до канонічної форми, що в сумі дало 4300 рядків із значеннями розчинності у воді для рядків SMILES не довше 60 елементів. Усі розчинності представлені у вигляді логарифмічної розчинності – десятковий логарифм максимальної концентрації розчиненої речовини у воді, вираженої у моль/л.

Важливо мати на увазі, що загальнодоступні набори даних часто об'єднують дані, отримані в різних лабораторіях за різними методиками. Це може сильно погіршити якість даних [265, 266].

4.1.3 Підготовка даних для моделі виправлення помилок.

Для тренування моделі для виправлення помилок хімічної будови було зібрано 300 000 пар помилкових SMILES та відповідних їм “правильних” представлень молекул. Для збільшення кількості та різноманітності даних для тренування знешумлювального автокодувальнику [267], до вихідних рядків SMILES було внесено помилки - випадкові заміни, видалення та вставки символів зі словника SMILES за визначеним розподілом. Модель повинна була навчитися перетворювати такі пошкоджені представлення у правильний вихідний рядок. Такий підхід змусив би модель вивчити більше можливих помилок та навчитися їх виправляти. У вищеописаний спосіб було додатково підготовлено 200 000 пар SMILES та додано до даних із помилками автокодувальнику. Таким чином, фінальний набір даних для навчання автокодувальнику становив 500 000 пар рядків.

4.2 Методологія.

В основі нашої методології лежить ідея апроксимації безперервного розподілу для представлення малих органічних молекул. Для створення такого відображення ми використовуємо автокодувальник (AE), що складається із двох поєднаних підмереж – кодувальника (2) і декодувальника (4), як показано на рисунку 4.2. Латентний векторний простір є центральним шаром нейронів автокодувальника. Його ваги вивчаються у процесі реконструкції валідних молекулярних структур у форматі SMILES (1), що подаються на вхід і на вихід автокодувальника. Після тренування, ці ваги апроксимують розподіл вхідних SMILES (1) та дозволяють відбирати вектори-представлення, які надалі декодуються (2) у нові молекулярні структури-кандидати (9).

Оскільки публічно доступні та розмічені набори даних мають обмежений розмір, окремі моделі для реконструкції структур, виправлення помилок і передбачення властивостей спочатку тренувалися на окремих (більших) наборах даних із однією цільовою змінною (pretraining). Пізніше, усі моделі були поєднані у ансамбль і дотреновані (fine-tuning) на наборі даних (меншому), який містив усі мітки (LogP, SMILES, помилки).

4.2.1 Архітектура та тренування моделі-генератора молекулярних структур.

Архітектура автокодувальника на рис. 4.2 була створена за результатами серії експериментів із пошуком по сітці гіперпараметрів. Змінювалася кількість шарів (з 4 до 12), їх типи (повнозв'язні і згорткові), кількість нейронів у повнозв'язних шарах (від 2 до 3500) та кількість Conv шарів (від 2 до 10) з різною кількістю і формою фільтрів. Додавання Conv шарів покращило точність автокодувальника. Експерименти з функціями активації (сигмоїдна, ReLU, PReLU) показали перевагу ReLU, за винятком сигмоїдної активації на останньому шарі нейронів. Фінальна архітектура автокодувальника була наступною:

1. **Кодувальник** - перша частина автокодувальника - складається із чотирьох згорткових шарів (кількість каналів, висота, ширина): (1, 58, 60), (60, 1, 58), (87, 1, 40), (116, 1, 30), (120, 1, 29), за якими слідує згортковий шар (512 нейронів).
2. **Декодувальник** - друга частина автокодувальника - призначений для декодування прихованих представлень назад до оригінальних рядків SMILES. Архітектурно він є дзеркальним відображенням кодувальника та складається з повнозв'язного і чотирьох згорткових шарів.
3. **Регресор** приймає на вхід активації останнього повнозв'язного шару кодувальника (латентне векторне представлення) та навчається, зіставляючи їх з набором даних про розчинність у воді (logS). Регресор складається з чотирьох повнозв'язних блоків зі зворотніми зв'язками (Residual): (512, 256), (256, 128), (128, 64), (64, 32) і чотирьох звичайних повнозв'язних шарів: (8), (4), (2), (1). Шари прогнозувальника поступово зменшують свій розмір, і останній виводить одне число, яке розглядається як передбачення розчинності.

Дискретні рядки SMILES фармакологічно активних молекул закодовувалися у безперервний латентний простір [268, 269] за допомогою повнозв'язного автокодувальника. Для тренування АЕ один і той же рядок SMILES подавався на вхід і на вихід автокодувальника. Одночасно,

розчинність у воді у вигляді $\log S$ подавалася на вихід регресора (7) (див. Рис. 4.2). Для розрахунку помилки, передбачена розчинність ($\log S$) молекули-кандидата перевіряється на відповідність дійсним значенням. Композитна помилка моделі складається із помилки реконструкції та помилки передбачення розчинності.

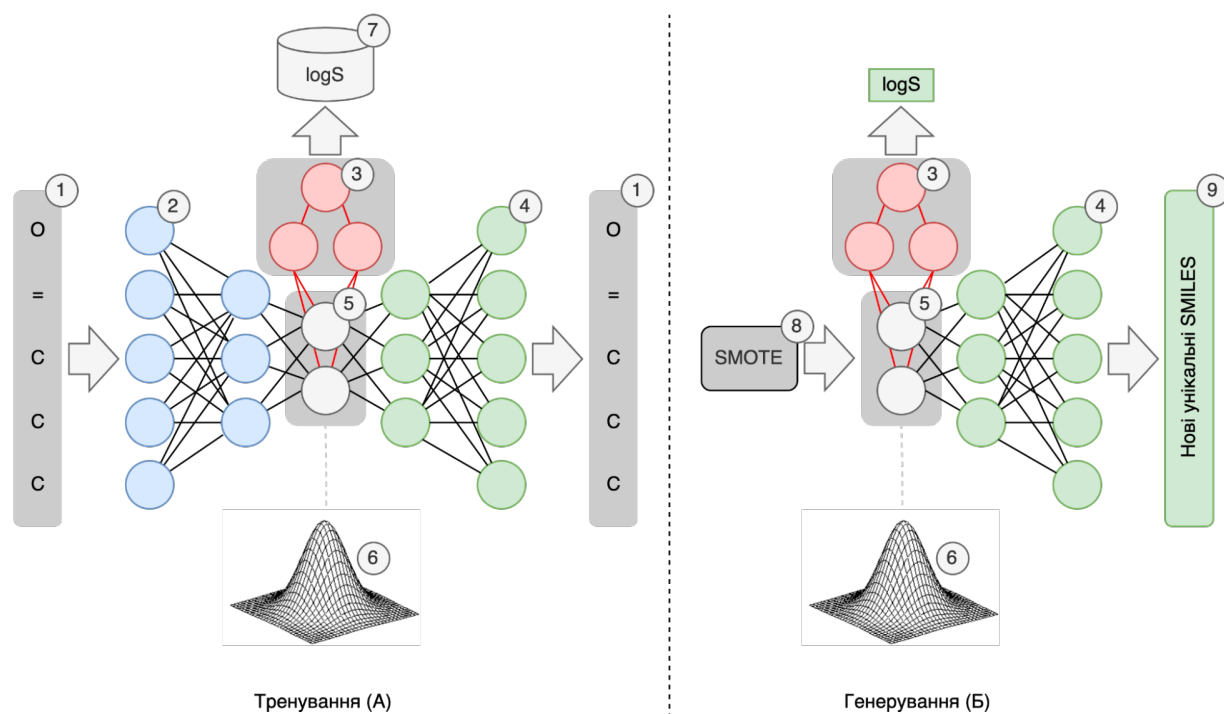


Рис. 4.2. Концептуальна архітектура мережі складається з автокодувальника (2), центрального шару (5) та кодувальника (4). Додатковим шаром нейронної мережі виступає регресор (3), який виконує функцію контролю фізико-хімічних властивостей створених молекул-кандидатів (9). В даному випадку такою характеристикою є логарифм розчинності у воді - $\log S$ (7). В результаті тренування (А) автокодувальника, ваги центрального шару (6) апроксимують розподіл вхідних даних (6). Після цього, натреновані центральний шар (5), регресор (3) та декодувальник (4) використовуються для створення (Б) нових молекулярних структур (9). SMOTE (8) виступає алгоритмом вибору початкових векторів з латентного простору (6) [7].

Оптимізатор Adam [229] використовувався для навчання автокодувальника та прогнозувальника. Під час навчання змінювався коефіцієнт швидкості навчання в межах від 10^{-3} до 10^{-5} . Застосовувалися

наступні функції втрат: для автокодувальника - бінарна крос-ентропія, для регресора - середньоквадратичне відхилення.

4.2.2 Модель для виправлення хімічних помилок.

Створені генератором SMILES із помилками складають від 30% до 99% від усіх створених [195]. Помилки виникають через розрідженість та неоднорідність векторного простору вивченого автокодувальником, а також специфіку граматики SMILES: один неправильний символ може призвести до зовсім іншої молекули, в той час як одна помилкова ймовірність на виході з кінцевого шару декодувальника не вплине на функцію втрат суттєво. За своєю суттю ця проблема аналогічна перевірці орфографії в обробці природної мови. Отож, було розроблено нейронну мережу, яка виправлятиме синтаксичні помилки в рядках SMILES і використовуватиметься як постобробка для результатів автокодувальника.

Виправлення помилок у SMILES є проблемою навчання від послідовності до послідовності, яка вирішується за допомогою рекурентних моделей із механізмом уваги [270, 271]. Кодувальник та декодувальник seq2seq моделі виготовлено із копійок ALSTM (Attention-based Long Short-Term Memory) [272] із розміром прихованого шару нейронів 512. Кодувальник перетворює вхідний рядок SMILES (X) у послідовність прихованих станів (h_1, h_2, \dots, h_n) , а декодувальник генерує по одному символу SMILES у цільовому рядку SMILES \hat{Y} . Формально, модель вивчає переходи $a : X \Rightarrow F^{512}$, $b : F^{512} \Rightarrow \hat{Y}$ так, що $a, b = \operatorname{argmin}(Y - b(a(X)))^2$. Вибір кожного наступного символу y^t обумовлений попереднім символом y^{t-1} і вектором контексту c_t . Вектором контексту обчислюється як зважена сума прихованих станів кодера:

$$c_t = \sum_{i=1}^{|X|} a_{ti} h_i$$

ваги яких визначаються за допомогою механізму уваги: $a_{ti} = \operatorname{softmax}(e_{ti})$, $e_t = A(\hat{Y}_{t-1}, S_{t-1})$, де A – нейронна мережа прямого поширення з одного повнозв'язного шару (див. рис. 4.3, елемент 1), а S^{t-1} – попередній прихований стан декодувальника. SMILES у формі ембедінгів [273] подаються до спеціального шару нейронів

(рис. 4.3, елементи 3, 8). Модель тренується шляхом мінімізації від'ємної логарифмічної ймовірності між згенерованим рядком SMILES \hat{Y} і цільовим (правильним) рядком SMILES Y [274].

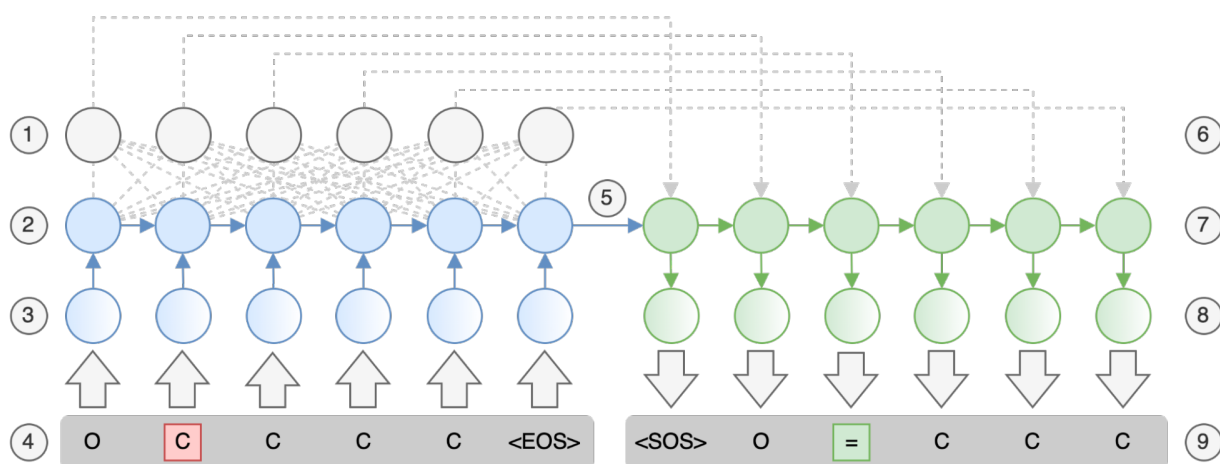


Рис. 4.3. Концептуальна архітектура моделі для виправлення помилок у SMILES. Для тренування, SMILES із помилками (4) подаються на вхід, а відповідні “коректні” SMILES (9) - на вихід. Кодувальник (2,3) позначено синім. Декодувальник (7,8) позначено зеленим. Кодувальник та декодувальник обмінюються контекстом (5), посилені шарами вбудування (3,8) та механізмом уваги (1) [7].

4.2.3 Відбір нових точок з латентного простору.

Архітектура генератора, описана у підрозділі 4.2.1 дозволяє апроксимувати дискретні SMILES у неперервний розподіл, надаючи необмежені можливості вибору довільних векторів у ньому. Це може призвести як до правильних хімічних структур, так і до неправильних або заскладних для синтезу у лабораторії. Зважаючи на це, замість випадкового вибору прихованих векторів, ми використали підхід SMOTE (Synthetic Minority Oversampling Technique) [275], взявши його реалізацію з бібліотеки Imblearn [276]. SMOTE працює наступним чином: обирає пару вихідних зразків, розташованих поблизу у підпросторі ознак, інтерполіує їх та генерує випадкові точки вздовж лінії між обраними зразками.

4.2.4 Молекулярна динаміка для перевірки очікуваної розчинності.

Для перевірки прогнозованого значення $\log S$ для згенерованих молекул-кандидатів була проведена серія симуляцій за методом молекулярної динаміки. Надалі повідомляються результати для однієї із згенерованих сполук - COCC(O)C(CC)CO . Вибір досліджуваної сполуки мотивований двома аргументами. По-перше, його хімічна структура відносно проста, тому можна очікувати, що типові силові поля взаємодії описують його належним чином. По-друге, помірною розчинністю цієї сполуки дозволяє моделювати обидва діапазони концентрацій (нижче і вище межі розчинності) без значного навантаження на розрахункові ресурси.

Ми приготували п'ять сумішей молекул води та молекул обраної сполуки різних концентрацій. Вода була представлена в рамках моделі SPC/E [277], тоді як молекули розчиненої речовини були побудовані в рамках силового поля OPLS-AA [278]. Параметри Леннарда-Джонса для різних сайтів були розраховані за допомогою правил змішування Лоренца-Бертелло. Усім частинкам дозволяли вільно рухатися через кубічну елементарну комірку із застосуванням періодичних граничних умов. Розміри комірок $L_x = L_y = L_z$ змінювалися в діапазоні 67–93 Å в залежності від складу розчину.

Молекулярно-динамічне моделювання проводилося з використанням пакету DL_POLY [279]. У всіх симуляціях ми використовували ансамбль NPT з тиском 1 бар і температурою 298 К, контрольованими баростатом і термостатом Nose-Hoover у реалізації Melchionna [280]. Кулонівські взаємодії на великій відстані розглядалися в рамках техніки Евальда з гладкою сіткою частинок, а для взаємодій на короткій дії було введено граничну відстань 9 Å. Рівняння руху були інтегровані у межах стандартної схеми перестрибування (leapfrog) із кроком 0,002 ps.

4.3 Результати.

4.3.1 Вибір розміру набору даних для попереднього тренування автокодувальника.

Для того щоб встановити залежність збіжності автокодувальника від кількості даних, було підготовлено набори даних з 10000, 50000, 100000, 200000, 500000 і 1000000 рядків SMILES. Розмір тестового набору даних із 20 000 об'єктів залишився незмінним у рамках цього експерименту. Зведений графік із залежністю величини похибки від розміру набору даних показано на рис. 4.4.

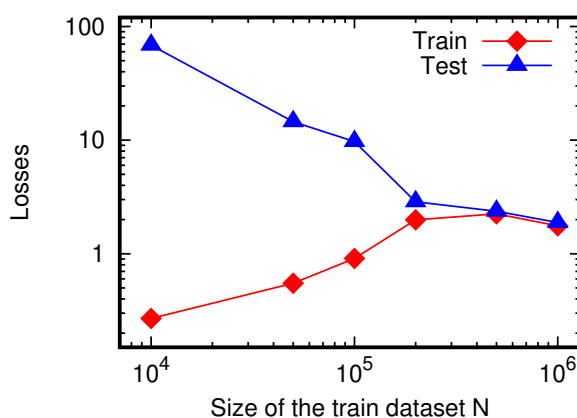


Рис. 4.4. Залежність величини помилки автокодувальника від кількості даних [8].

Із наведеного вище графіку видно, що для наборів із 500 тис. та 1 млн рядків різниця у результуючій похибці невелика. Отож, для подальшого дотренування на даних із розчинністю було вирішено використати модель автокодувальника, попередньо натреновану на 500 тис. SMILES.

4.3.2 Дотренування автокодувальника із додатковим модулем прогнозування розчинності.

Під час донавчання автокодувальнику із модулем прогнозування розчинності використовувалася процедура ранньої зупинки, щоб запобігти перенавчанню. Якість передбачення розчинності показано на рис. 4.5 - показано проекцію експериментальних та передбачених значень розчинності на тренувальній (зліва, $R^2 = 0.97$) та тестувальній (справа,

$R^2 = 0.84$) вибірках. Ідеальні прогнозовані значення $\log S$ відповідають функції $y = x$ (показано червоними лініями).

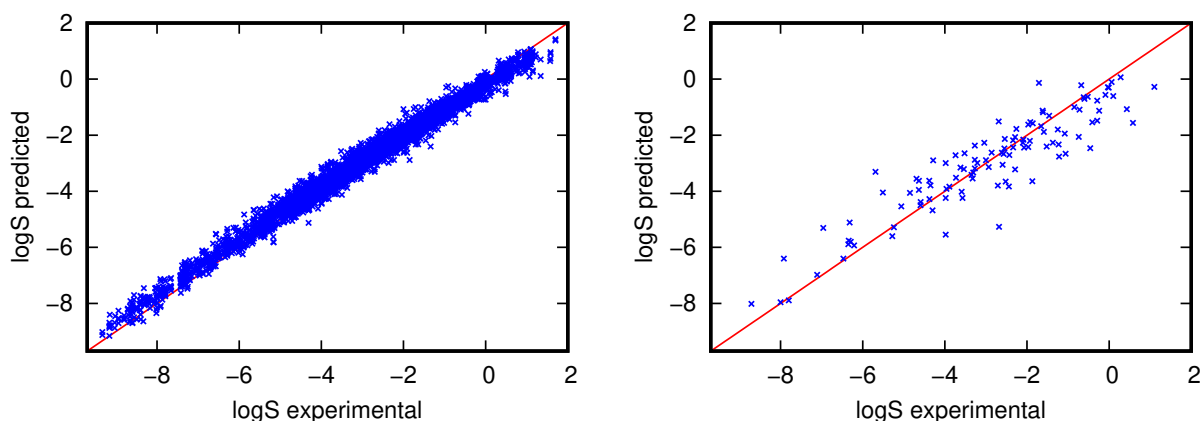


Рис. 4.5. Проекція передбачених $\log S$ на експериментальні на тренувальному (ліворуч) та тестовому (праворуч) наборі даних. Червона лінія $y = x$ позначає «ідеальну реконструкцію» для порівняння [8].

Різниця між прогнозованими та експериментальними значеннями $\log S$ не перевищує одиниці (див. правий графік на рис. 4.5), що в звичайних одиницях концентрації відповідає похибці одного порядку. Частково це пов'язано з відносно невеликим набором даних про розчинність, який містить лише 4300 значень. Тим не менш, навіть приблизна оцінка є корисною для класифікації розчинності сполуки. Отриманий результат ($R^2 = 0,84$) є кращим за інші спроби вирішити цю задачу [169, 266], проте пряме порівняння підходів доцільне лише за однакових наборів даних.

4.3.3 Тренування та оцінка моделей для виправлення помилок хімічної будови.

Моделі виправлення помилок оцінювалися на трьох типах вхідних даних (усі взяті з раніше невидимих моделями тестових наборів розміром ≈ 15000 пар SMILES):

1. згенеровані SMILES - щоб оцінити ефективність реконструкції генератору;
2. згенеровані рядки SMILES із помилками моделі-генератору - щоб оцінити ефективність моделі для виправлення помилок генератору;

3. рядки SMILES із випадковим шумом — щоб перевірити здатність моделі-коректора виправляти випадкові помилки.

Якість роботи моделі для виправлення помилок оцінювалися за трьома критеріями:

1. Здатність реконструювати SMILES без помилок: скільки вихідних рядків SMILES збігаються із вхідними при реконструкції.
2. Здатність виправляти SMILES із помилками автокодувальника: скільки виправлених рядків SMILES збігаються із оригінальними непошкодженими рядками після виправлення помилок.
3. Здатність виправляти SMILES із випадковими помилками: скільки виправлених рядків SMILES є хімічно коректними (перевірено бібліотекою RDKit [113]).

Метрики моделей узагальнено в таблиці 4.1.

Таблиця 4.1

Порівняння ефективності моделей для виправлення помилок [8].

Завдання	Випадкові помилки		Помилки генератору	
	однакові	хім.коректні	однакові	хім.коректні
Реконструкція	87%	93%	37%	58%
Випадкові помилки	66%	83%	16%	36%
Помилки ген-ру	14%	44%	43%	68%

4.3.4 Аналіз створених молекулярних структур.

За допомогою дотренованого генератору та алгоритму SMOTE було згенеровано 95446 нових молекул-кандидатів. З них, 60,3% (57556 шт.) виявилися хімічно правильними, а 39,7% (37890 рядків) - помилковими, найчастіше через химерні ароматичні системи або неправильні валентності атомів. Хімічно неправильні SMILES були виправлені моделлю корекції помилок. 12040 (31,8%) рядків були успішно виправлені. Майже усі виправлені стрічки SMILES були унікальними (99,8% або 12024 з 12040). З 57556 правильних молекул, згенерованих АЕ, лише 10,2% (5836 шт.)

виявилися унікальними, інші 89,8% (51720) були ідентичними до молекул у початковому наборі даних. Отож, загалом було створено 17860 унікальних нових молекул — 5836 з АЕ і 12024 з моделі виправлення помилок.

Застосування двох моделей виправлення помилок (навчених на помилках генератору та на випадкових помилках) збільшило на **20%** кількість хімічно коректних SMILES. Лише 10,2%(5836) згенерованих SMILES були унікальними. **67%**(12040) унікальних SMILES склали виправлені структури. Отож, моделі виправлення помилок не просто перетворюють усі некоректні рядки SMILES у коректні, які вони бачили під час навчання, а створюють нові, раніше невідомі молекули.

Аналіз каркасів. Каркас (від англ. scaffold – рiштування, каркас) – це частина молекули, що залишається після видалення некілецевих замісників, а для молекул без кілець – це найдовший вуглецевий ланцюг. Набір із 5836 новостворених структур містив 3945 унікальних каркасів. Для порівняння, вхідний набір даних із 189936 молекул, використаних для процедури відбору SMOTE, містив 58229 каркасів. Перекриття між згенерованими та вхідними наборами даних становило 2558 каркасів (64,8% згенерованих каркасів); перекриття між згенерованими та виправленими наборами становило 742 каркаси (8,8% від виправлених або 18,8% згенерованих каркасів); перекриття між виправленими та вхідними наборами даних становило 2594 каркаси (30,8% виправлених каркасів). Ці числа показують, що досліджуваний ансамбль генерує нові молекули та виправляє помилкові в межах подібного розподілу, не копіюючи при цьому існуючі підструктури повністю.

Порівняння структурних особливостей. Було розраховано наступні структурні характеристики для трьох наборів даних (тренувального, згенеровані та виправлені молекули): кількість кілець у молекулі, наявність спірокілець, гетероциклів, деяких біогенних елементів і галогенів. Розподіл створених SMILES подібний до розподілу вихідного набору даних, але не повторює його повністю. Виправлені SMILES мають наступні відмінності від вихідних та згенерованих: більша кількість структур з одним і двома кільцями та менша кількість структури з

декількома циклами та гетероциклами. Описані структурні особливості зведено у таблиці 4.2.

Таблиця 4.2

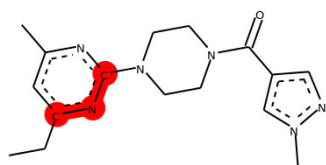
Порівняння особливостей хімічної будови вихідних, створених та виправлених молекул [8].

Ознака	Тренувальний,%	Створений,%	Виправлений,%
немає кілець	1.66	2.65	2.06
1 кільце	7.74	7.88	9.04
2 кільця	22.35	19.93	24.56
3 кільця	32.33	31.6	31.75
4 кільця	28.08	29.49	26.27
> 4 кілець	7.84	8.45	6.31
спіро кільця	2.01	1.99	1.94
гетероцикли	80.4	77.9	73.62
немає N,O,S	0.48	0.98	1.0
є N	94.06	92.37	91.74
є O	93.25	92.3	91.51
є S	39.25	37.56	35.95
є галоген	38.57	37.32	39.26

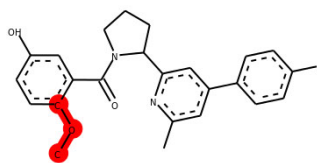
Порівняння функціональних груп. Ми використали алгоритм ідентифікації функціональних груп в органічних молекулах [281] для пошуку та аналізу підструктур у молекулах. Виявлені спільні підструктури були відсортовані за частотою зустрічань від найбільш до найменш частих і представлені як SMARTS (SMILES Arbitrary Target Specification) [282].

На рис. 4.6 показано найпоширеніші спільні підструктури у тренувальному, згенерованому та виправленому наборах даних. Підструктури виділено червоним кольором і представлено як частини згенерованих рядків SMILES. Наведені приклади впорядковано за зменшенням частоти зустрічань вираженої у відсотках. Результати свідчать що молекули у тренувальному, згенерованому та виправленому наборах даних мають подібні розподіли функціональних груп. Причиною

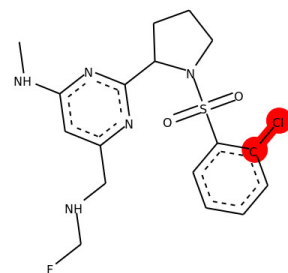
є їх походження зі спільного хімічного (латентного) простору.



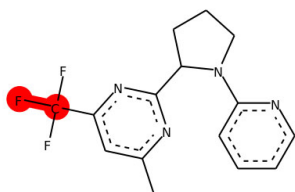
cnc 13.2% 12.9% 11.9%



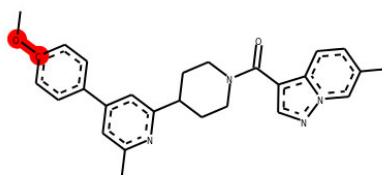
cOC 9.4% 7.0% 8.1%



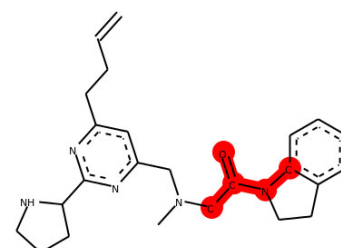
cCl 5.4% 5.4% 5.7%



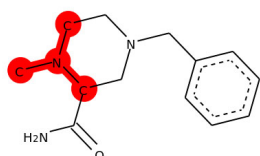
CF 3.6% 3.9% 3.5%



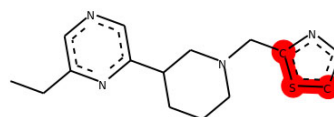
cO 1.5% 3.3% 2.0%



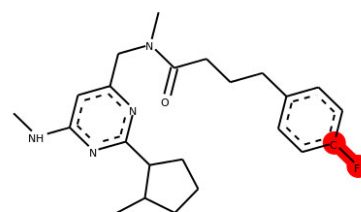
cNC(C)=O 3.3% 3.3% 3.0%



CN(C)C 2.9% 2.8% 2.8%



csc 3.2% 2.7% 2.8%



cF 2.7% 2.5% 2.6%

Рис. 4.6. Спільні підструктури в наборах даних. Формат легенди: код підструктури SMARTS, відсоток входжень у вихідному наборі даних SMILES, відсоток входжень у новому згенерованому наборі даних SMILES, відсоток входжень у виправленому наборі даних SMILES. Підструктури виділені червоним кольором і представлені як частини нових створених рядків SMILES [8].

Порівняння молекулярних дескрипторів. Властивості згенерованих, виправлених і вихідних сполук порівнювали за допомогою таких дескрипторів як молекулярна маса, $\log P$ [283] і топологічна площа поверхні). Топологічна полярна площа поверхні (TPSA) молекули визначається як сума поверхні всіх полярних атомів, включаючи приєднані до них атоми водню. $\log P$ є найбільш часто використовуваним показником ліпофільності. Це коефіцієнт розподілу розчиненої речовини між водною та ліпофільною фазами, зазвичай октанолом і водою. Результати показано на рис. 4.7. Можна побачити, що три порівнювані набори даних демонструють схожі - але не повністю однакові - розподіли.

Оцінка придатності до синтезу у лабораторії. Оцінка придатності до синтезу (SAS) - це число від 1 (легко синтезувати) до 10 (практично неможливо синтезувати). Ця метрика була розрахована відповідно до методології P.Ertl [284]. Цей метод використовує історичні синтетичні знання, отримані шляхом аналізу інформації про мільйони вже синтезованих хімічних речовин, і враховує складність хімічної структури. Синтетична доступність використовується для ранжування молекул, отриманих теоретичними методами, при переході до органічного синтезу у лабораторії. Отримані розподіли представлені на рис. 4.7, нижній правий графік. Середні значення SAS для вихідних, нових генерованих і виправлених молекул становлять 2,5, 2,6 і 2,5 - це вказує, що вони є відносно простими для синтезу.

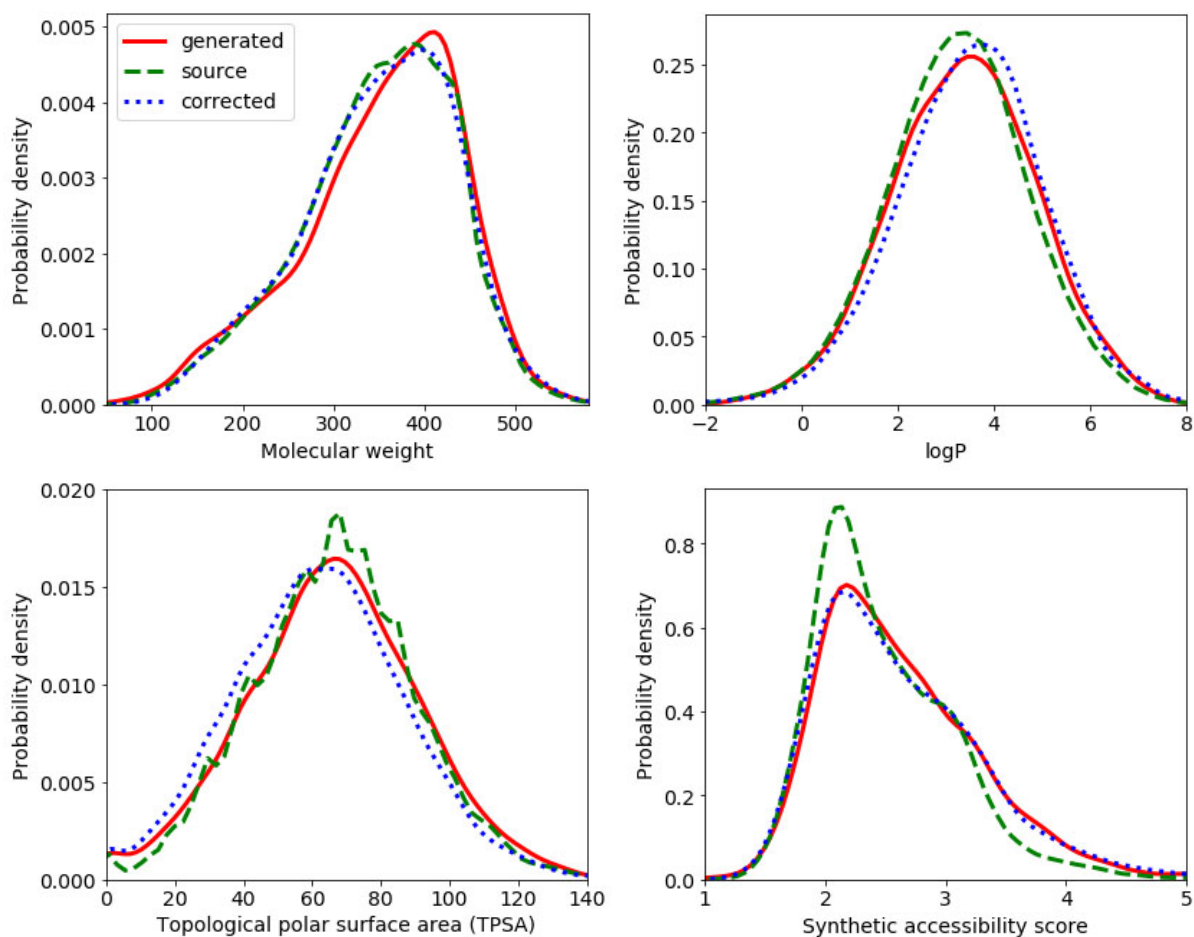


Рис. 4.7. Порівняння густин ймовірності молекулярних дескрипторів (молекулярна маса, logP, TPSA) і оцінки синтетичної доступності для вихідного набору даних, згенерованих і виправлених молекул [8].

4.3.5 Оцінка прогнозованої розчинності.

Для перевірки передбачених значень розчинності ($\log S$) була проведена серія комп'ютерних симуляцій. Розчинність у комп'ютерному моделюванні вимірюється через обчислення вільних енергій за допомогою термодинамічного інтегрування або усередненням термодинамічних траєкторій переходів між станами [285, 286, 287]). Такий підхід вимагає інтенсивного моделювання великої кількості проміжних станів та залучення значних розрахункових ресурсів. Тому, для спрощення завдання та зменшення розрахункового навантаження, ми зосередилися замість кількісного на якісному завданні - перевірити чи виявляє певна композиція розчинник-розчинена речовина будь-які тенденції поділу щодо прогнозованої межі розчинності.

Як приклад ми вибрали одну зі згенерованих сполук: COCC(O)C(CC)CO (показано на рис. 4.8). Його передбачена розчинність $\log S$ становить $-0,26$, що відповідає концентрації $\approx 0,55$ моль/л.

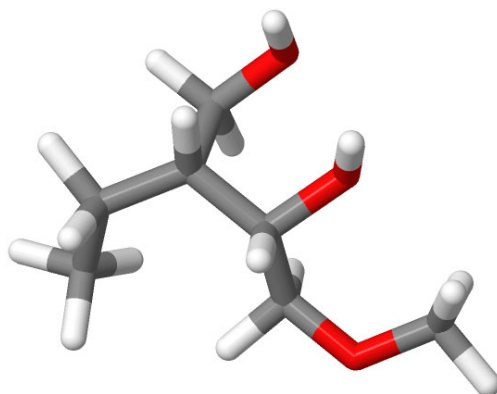


Рис. 4.8. Структура молекули COCC(O)C(CC)CO . Вуглеці позначено сірим кольором, кисень – червоним, водні – білим [8].

Для цього було створено п'ять повноатомних сумішей, що складаються з молекул розчинника H_2O і молекул розчиненої речовини $\text{C}_7\text{H}_{16}\text{O}_3$. Змодельовані суміші відрізнялися за співвідношенням кількості молекул розчинник-розчинена речовина. Концентрації речовин зібрані у таблиці 4.3. Для композицій з концентрацією (нижче межі розчинності $\approx 0,55$ моль/л) речовини не повинні розділятися, на відміну від режиму «розділення» для випадків з концентраціями вище межі розчинності. Композиції з концентраціями, близькими до порогового значення $\log S$ ($\approx 0,55$ моль/л), можуть демонструвати переходи станів.

Таблиця 4.3

Склад та концентрація модельних систем [8].

Кількість молекул		Концентрація
H_2O	$\text{C}_7\text{H}_{16}\text{O}_3$	[моль/л]
20000	10	0.03
10000	20	0.11
10000	100	0.5
Межа розчинності		0.55
10000	400	1.7
10000	2000	4.2

Результати симуляцій описаних вище систем за методом молекулярної динаміки показані на рис. 4.9. Для спрощення візуального сприйняття, результати представлені як елементарні комірки моделювання із розділеними молекулами розчинника (H_2O , верхній ряд) і розчиненої речовини ($C_7H_{16}O_3$, нижній ряд).

При концентрації 0,11 моль/л молекули розчинника (верхня ліва комірка, рис. 4.9) і розчиненої речовини (нижня ліва комірка, рис. 4.9) хаотично розподіляються по простору симуляції та не агрегуються. Концентрації 0,03 моль/л та менші не показані, оскільки вони нагадують випадок 0,11 моль/л.

У другому стовпчику зліва рис. 4.9 показано систему з концентрацією розчиненої речовини 0,5 моль/л. Ця концентрація трохи нижча передбачуваної межі розчинності 0,55 моль/л. У верхній комірці знаходяться молекули H_2O , у нижній — $C_7H_{16}O_3$. Можна помітити, що розчинена речовина добре розподіляється по коробці і тільки починає агрегуватися.

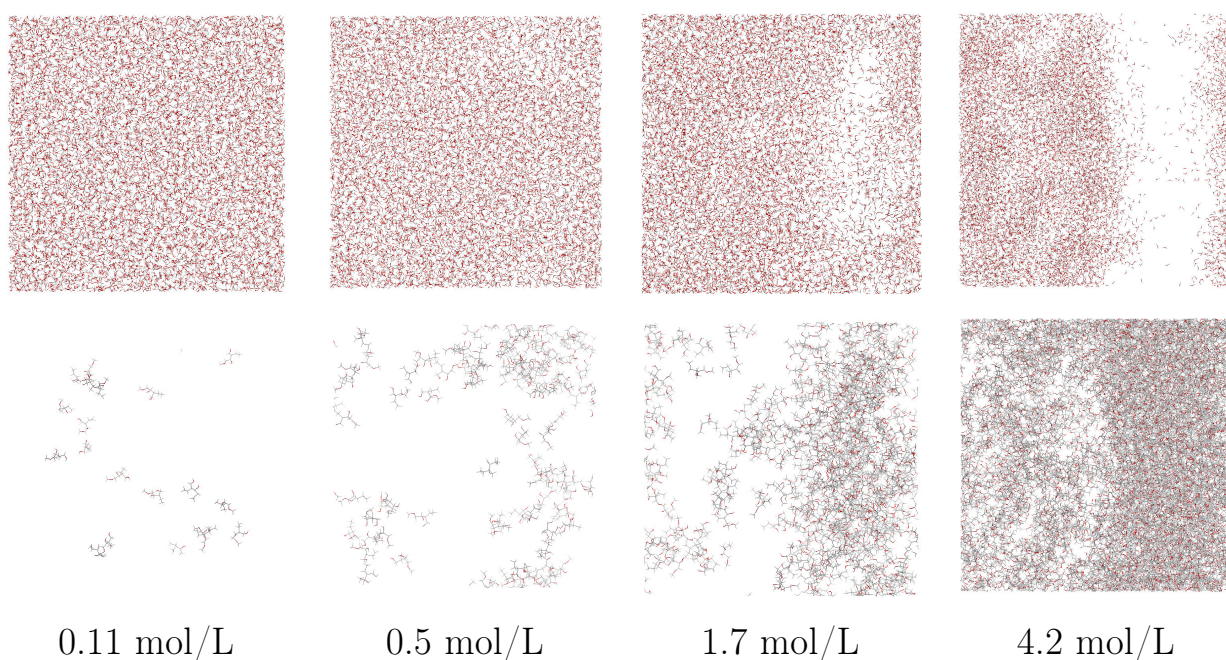


Рис. 4.9. Фрейми симуляцій з молекулами води H_2O (верхній ряд) та молекулами $C_7H_{16}O_3$ (нижній ряд). Вуглеці позначено сірим кольором, Оксигени – червоним, Водні – білим. Концентрації розчинених речовин позначені під кожною колонкою [8].

У третьому стовпчику зображено систему з концентрацією розчиненої речовини 1,7 моль/л. Ця концентрація є вищою за передбачену межу розчинності та чітко демонструє тенденцію поділу, яка узгоджується з нашою оцінкою розчинності.

Випадок найвищої концентрації розчиненої речовини (4,2 моль/л) показано в крайньому правому стовпчику рис. 4.9. Чітко видно області, не зайняті H_2O , а також області, не зайняті $C_7H_{16}O_3$. Фактичний склад значно перевищує межу розчинності, тому поділ добре проявляється.

4.4 Висновки.

У цьому розділі описано поєднання автокодувальника із прогнозувальником розчинності ($\log S$) та рекурентною нейронною мережею із механізмом уваги для розробки нових унікальних лікарських речовин. Такий конвеєр нейронних мереж дозволяє створювати нові лікарські речовини майже миттєво, прогнозувати їхні властивості без проведення лабораторних випробувань та досліджувати схожість на ліки.

Першу модель ансамблю (автокодувальник) було натреновано на сотнях тисяч фармакологічно активних молекул у форматі SMILES. Регресійна модель, вбудована у попередньо натренований автокодувальник, тренувалася разом із автокодувальником на SMILES із відомою розчинністю у воді ($\log S$). Декодувальник із натренованого AE та алгоритм SMOTE використовувалися надалі для генерування нових хімічних структур. Значна частина (близько 40%) згенерованих структур містила помилки. Помилкові SMILES було об'єднано зі SMILES, у які навмисне були внесені випадкові помилки. На отриманому наборі даних з 500 тис. пар натреновано модель для виправлення помилок хімічної будови - другу модель ансамблю. Кількість виправлених помилок цією моделлю склала 68% - для помилок автоенкодера та 83% - для випадкових помилок. Аналіз структурної подібності еталонних і створених структур показує їхню подібність із одночасним збереженням унікальності останніх.

Методом молекулярної динаміки у силовому полі OPLS-AA було встановлено, що згенерована молекула-кандидат ($C_7H_{16}O_3$) є фізико-хімічно стабільною, а її передбачена розчинність у воді ($\approx 0,55$ моль/л) відповідає симуляційній межі розчинності.

5 РОЗДІЛ 5. АРХІТЕКТУРНА ДІАГРАМА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ РОЗРОБКИ ЛІКАРСЬКИХ РЕЧОВИН.

У цьому розділі розроблено діаграму інформаційної системи для створення лікарських речовин із бажаними біологічними та фізико-хімічними властивостями. Інформаційна система поєднує технології описані у попередніх розділах: генерацію молекул-кандидатів та комплексний контроль їх синтетичної доступності, молекулярної маси, розчинності, топологічної полярної площі поверхні, очікуваного виходу продукту у реакції синтезу та молекулярної спорідненості до обраного рецептору. Таким чином, інформаційна система є ефективним інструментом розробки молекул-кандидатів із високою імовірністю затвердження контролюючими органами.

5.1 Концептуальна архітектурна діаграма інформаційної системи для розробки лікарських речовин.

Концептуальну архітектурну діаграму інформаційної системи для розробки лікарських речовин та опис її компонентів показано на рис. 5.1. Елементи на рисунку розташовані у логічній послідовності процесу створення молекулярних структур та передачі їх на синтез до лабораторії. **1** - вибірка SMILES залежно від очікуваних властивостей ліків. Це може бути множина відомих інгібіторів цільового рецептора. Цей набір даних кодується енкودером генератору у вектори латентного простору; **2** - модуль SMOTE [275] використовується для відбору нових точок із закодованих дискретних векторів латентного простору та передає їх у декодувальник **3**; **3** - декодування вбудовань SMOTE із одночасним контролем розчинності **4**. **5** - молекулярні структури із помилками хімічної будови надходять у модуль виправлення помилок. Результатом є хімічно коректні молекули-кандидати **6**, які надходять на фільтри подібності до лікарських речовин. Спершу визначається очікувана спорідненість до

обраного рецептору **7**. Відфільтровуються не-інгібітори - молекули з K_i більше 10 000 нмоль/л. Для інгібіторів ($K_i < 10\ 000$ нмоль/л) виконується оцінка токсичності, рівня проникності через гемато-енцефалічний бар'єр (BBBP) та інші показники схожості на ліки [288] **8**. Відповідно до вказаних показників, молекули-кандидати ранжуються від менш до більш успішних. Для найбільш успішних інгібіторів виконується підготовка до синтезу. У рамках підготовки до синтезу у лабораторії спершу визначається фактична можливість виготовлення обраної молекули за допомогою ChemSpace API [289] **11**. Обираються лише молекули які можна синтезувати. Далі, оскільки вартість виготовлення напряму залежить від складності синтезу та кількісного виходу продукту реакції, оцінюється синтетична доступність кожної молекули (SAS) [284] **10** та прогнозується фактичний вихід продукту у реакціях синтезу **9**. Обираються кандидати із найнижчою складністю синтезу та найвищим очікуваним відсотком виходу продукту. Замовляється синтез цих молекул-кандидатів для початку доклінічних досліджень.

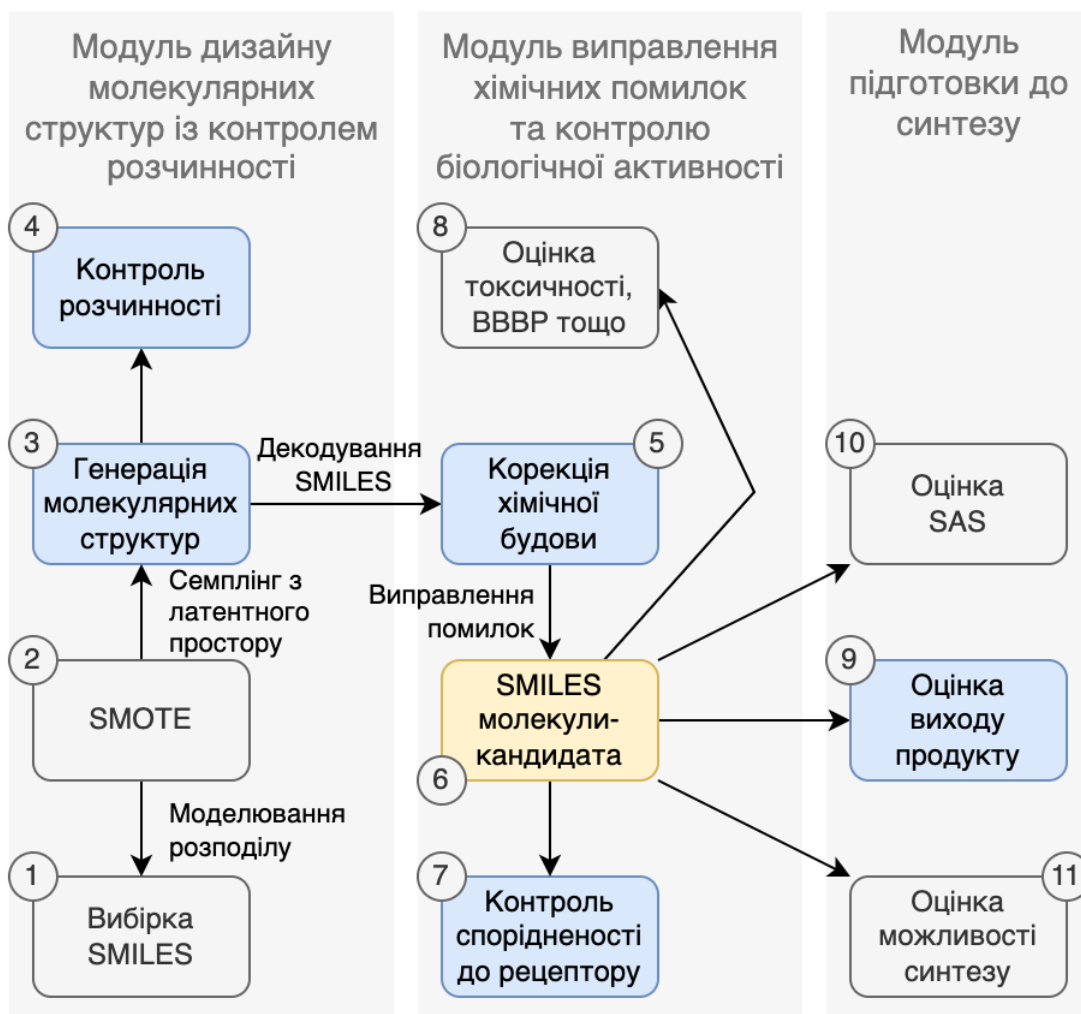


Рис. 5.1. Концептуальна діаграма інформаційної системи для розробки лікарських речовин. Синім позначено авторські технології, розроблені у межах даної дисертації. інформаційна система складається із наступних елементів: **1** - вибірка SMILES відповідно очікуваних властивостей ліків; **2** - модуль SMOTE [275] для ініціації нових точок з латентного простору; **3** - генератор, що декодує вдубування SMOTE та контролює розчинність **4** утворених молекул; **5** - модуль виправлення помилок хімічної будови; **6** - хімічно коректні молекули-кандидати. **7** - модуль визначення очікуваної спорідненості до обраного рецептору; **8** - модуль оцінки токсичності, рівня проникності через гемато-енцефалічний бар'єр (BBBP) та інших показників схожості на ліки [288]; **9** - модуль прогнозування фактичного виходу продукту; **10** - оцінка синтетичної доступності [284]; **11** - визначення фактичної можливості синтезу обраної молекули у лабораторії за допомогою ChemSpace API [289].

5.2 Детальна архітектурна діаграма інформаційної системи для розробки лікарських речовин.

Діаграма інформаційної системи для розробки лікарських речовин показана на рис. 5.2:

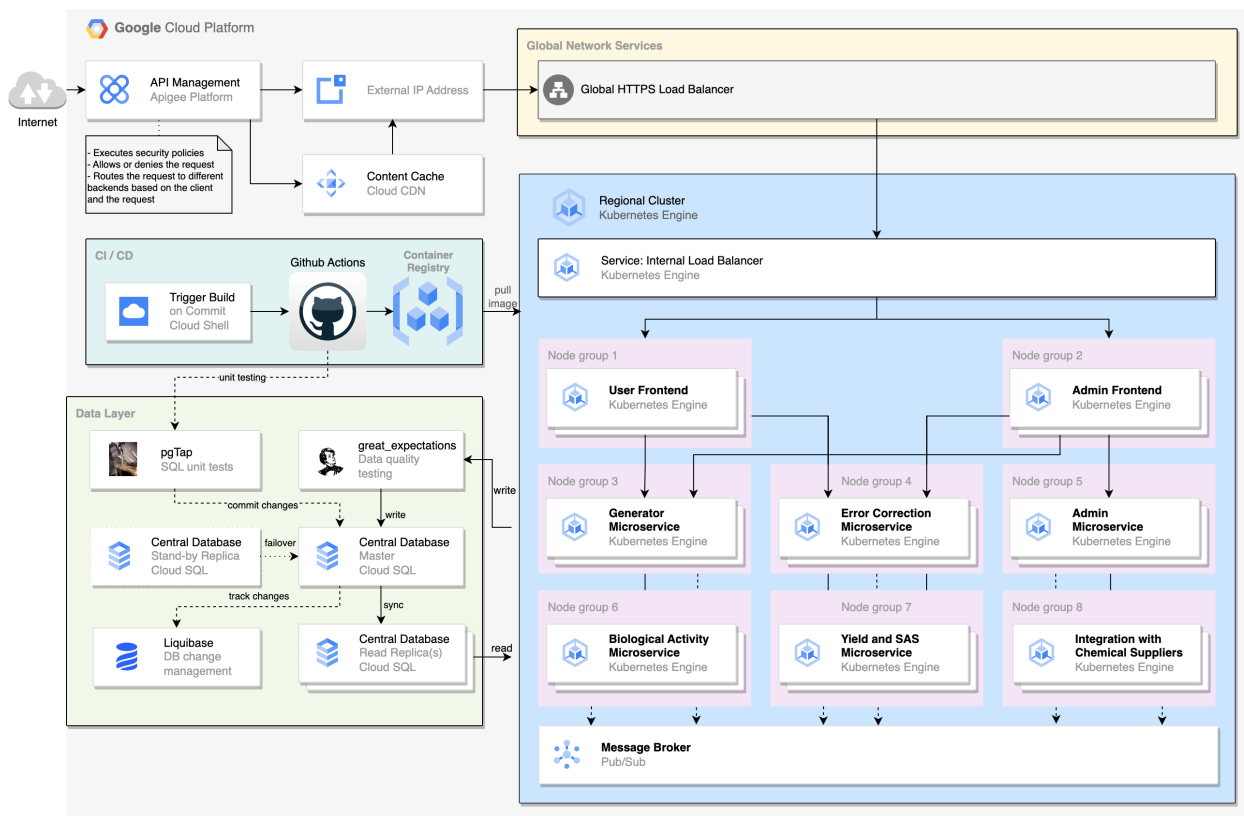


Рис. 5.2. Детальна архітектурна діаграма інформаційної системи для розробки лікарських речовин.

Опис елементів діаграми 5.2:

1. API Management (Google Apigee Platform) забезпечує повні можливості життєвого циклу API та автомасштабування. Доступ через мобільні пристрої та ПК.
2. Зовнішня IP-адреса (сервіс GCP) має пул статичних IP-адрес як інтерфейс із зовнішнім публічним Інтернетом.
3. Global HTTPS Load Balancer - єдина глобальна точку входу в систему. Має статичний публічний IP.
4. Кеш контенту (Cloud CDN) - механізм периферійного кешування картинок та великих файлів.

5. Регіональний кластер (Kubernetes Regional Cluster) - один кластер на регіон. Кластер розміщує всі мікросервіси системи. Усі кластери підключені до Global HTTPS Load Balancer.
6. Internal Load Balancer - автоматичне балансування навантаження всередині одного регіонального кластеру Kubernetes між нодами та колбеки для перевірки функціональної активності мікросервісів та горизонтального автомасштабування.
7. Інтерфейс користувача (Kubernetes Deployment) - інтерфейс кінцевого користувача реалізований як горизонтально автоматично масштабований мікросервіс. Відповідно до налаштувань рівня доступу, може взаємодіяти з одним або кількома мікросервісами бізнес-шару, окрім Admin Microservice.
8. Інтерфейс адміністратора (Kubernetes Deployment) - інтерфейс адміністратора сервісу. Підтверджує або скасовує підписки користувачів, встановлює та змінює їх рівні доступу. Інтерфейс адміністратора реалізований як горизонтально автоматично масштабований мікросервіс. Взаємодіє з Admin Microservice.
9. Мікросервіс адміністратора (Kubernetes Deployment) - містить бізнес-логіку керування користувачами та їх доступами, затвердження та скасування підписок, додавання та видалення постачальників органічних молекул. Публікує результати у топик notifications.admin брокера повідомлень Pub/Sub. Підписаний на тему notifications.user.
10. Мікросервіс генерації (Kubernetes Deployment) - містить бізнес-логіку керування генерацією молекулярних структур, реалізований як горизонтально автоматично масштабований мікросервіс. Публікує результати у топик notifications.smiles, notifications.logs брокера повідомлень Pub/Sub. Підписаний на теми notifications.user і notifications.admin.
11. Мікросервіс виправлення помилок (Kubernetes Deployment) - містить бізнес-логіку керування виправленням помилок молекулярних структур, реалізований як горизонтально автоматично

масштабований мікросервіс. Публікує результати у топик `notifications.smiles_corrected` брокера повідомлень Pub/Sub. Підписаний на теми `notifications.smiles` і `notifications.user`.

12. Мікросервіс біологічної активності (Kubernetes Deployment) - містить бізнес-логіку керування визначенням молекулярної спорідненості, оцінки токсичності, проникності через гематоенцефалічний бар'єр, реалізований як горизонтально автоматично масштабований мікросервіс. Публікує результати у топик `notifications.biology` брокера повідомлень Pub/Sub. Підписаний на теми `notifications.smiles_corrected` і `notifications.user`.
13. Мікросервіс виходу продукту (Kubernetes Deployment) - містить бізнес-логіку керування визначенням відсотку виходу продукту хімічної реакції та оцінки синтетичної доступності, реалізований як горизонтально масштабований мікросервіс. Публікує результати у топик `notifications.chemistry` брокера повідомлень Pub/Sub. Підписаний на теми `notifications.smiles_corrected` і `notifications.user`.
14. Мікросервіс інтеграцій із постачальниками органічних речовин (Kubernetes Deployment) - містить бізнес-логіку керування інтеграціями із постачальниками органічних речовин. Публікує результати у топик `notifications.supplier` брокера повідомлень Pub/Sub. Підписаний на теми `notifications.smiles_corrected`, `notifications.user` і `notifications.admin`.
15. CI/CD складається з дій Github і реєстру контейнерів GCP (`gcr.io` або `Docker Hub`). Конвеєр запускається автоматично під час кожного коміту у головну гілку або вручну з `Cloud Console`. У цьому випадку новий образ `Docker` створюється та розгортається у відповідній мікрослужбі автоматично.
16. `Data Layer` - складається з однієї головної бази даних `Cloud SQL` на регіон, однієї чи кількох реплік для читання та резервної репліки, яка автоматично активується у разі збою основного головного екземпляра `Cloud SQL`. Основна база даних використовується для запитів на запис, тоді як репліки на читання обробляють навантаження

на читання. Автоматичне керування версіями БД і розгортання надаються Liquibase [290]. Тестування модулів SQL для процедур PL/SQL надається разом із pgTap [291]. pgTap — це пакет для написання модульних тестів у форматі PL/SQL функцій для баз даних. Дані для запису у базу даних, перевіряються за допомогою Great Expectations [292]. Great Expectations допомагає командам обробки даних уникнути технічного боргу шляхом попереднього тестування даних, документування та профілювання.

5.3 Висновки.

У розділі запропоновано концептуальну діаграму та діаграму для розгортання інформаційного застосунку для розробки ліків. Інформаційна система дозволяє поєднати генерацію молекул-кандидатів та комплексний контроль їх синтетичної доступності, молекулярної маси, розчинності, топологічної полярної площі поверхні, очікуваного виходу продукту у реакції синтезу та молекулярної спорідненості до обраного рецептору - сукупність цих інформаційних технологій дозволить створювати молекули-кандидати із високим рівнем успішності на клінічних дослідженнях. Важливою характеристикою системи є безпосередня інтеграція із постачальниками органічних речовин. Ця особливість робить інформаційну систему практичним та ефективним інструментом для створення лікарських речовин із бажаними біологічними та фізико-хімічними властивостями.

ВИСНОВКИ

У дисертаційній роботі розв'язане актуальне наукове завдання розроблення методів та засобів аналізу хімічних сполук засобами штучного інтелекту.

Основні результати подано нижче.

1. Здійснено аналіз предметної області, методів та підходів машинного навчання до прогнозування молекулярної спорідненості, фактичного виходу продукту хімічної реакції та генерації молекулярних структур. Це дало змогу виділити невирішені проблеми, недоліки існуючий методів, перспективні області покращення та здійснити постановку задач дисертаційної роботи (див. Розділ 1).
2. Розроблено метод аналізу молекулярної спорідненості лікарських молекул до обраного рецептору [5, 3, 4, 12]. Метод прогнозує активність молекул-кандидатів не лише якісно (активний або неактивний), але й кількісно (значення K_i). Показано, що поєднання моделей методом мета-стекингу збільшує відгук (Recall) класифікації на **34,9%**, підвищує загальну точність та статистичну достовірність результатів (R^2) на **21%**; дозволяє виключити (у випадку класифікації) або компенсувати (у випадку регресії) помилки, допущені іншими моделями ансамблю (див. Розділ 2).
3. Розроблено нову архітектуру графової нейронної мережі для передбачення відсотку виходу продукту хімічної реакції [6]. Мережа поєднує структурну інформацію про учасників трансформації, а також дескриптори рівня молекули та реакції. Ефективність графової нейронної мережі порівнювалася із логістичною регресією, методом опорних векторів, градієнтним бустингом та нейронними мережами-трансформерами. Отримані результати (R^2 **0.86**, $RMSE$ **10.35**) перевершили усі відомі підходи на час публікації [112, 118, 109] (найкращі результати на тому ж наборі даних - R^2 0.81, $RMSE$ 12.07) (див. Розділ 3).
4. Розроблено метод дизайну лікарських речовин, що дозволяє контролювати одну або більше властивостей створюваних молекул,

а також має послідовний модуль виправлення помилок хімічної будови [11, 8, 7]. Модель виправлення хімічних помилок підвищує вихід коректних молекулярних структур на **20%**, вихід унікальних молекул - на **67%** (див. Розділ 4).

5. Вдосконалено метод редукції [230, 231, 232, 233]. Покращений метод редуктивного спрощення [9] динамічно визначає кількість та видаляє групи нейронів за одну ітерацію. Така модифікація дозволяє методу видаляти усі "зайві" ваги за малу кількість ітерацій. Навчання двох варіантів моделі на даних зі збуреннями, середнє значення яких дорівнює нулю, стабілізує навчання та усуває ризик адаптації моделі до певної величини збурень. Перехресна оцінка здатності до генералізації за участі усіх зразків у даних дозволяє уникнути потенційного ризику пристосування спрощеної моделі до фіксованого тестового набору. Видалення ваг завершується за неможливості подальшого видалення параметрів без погіршення здатності моделі до узагальнення. На прикладних задачах по передбаченню молекулярної афінності (див. Розділ 2) та відсотку виходу продукту хімічної реакції (див. Розділ 3) показано, що редуктивне спрощення зменшує кількість активних ваг моделей на **86.88%** та **29.21%** і одночасно з цим підвищує коефіцієнт детермінації (R^2) відповідних моделей на **2.8%** та **15.43%** [9].
6. Розроблено мікросервісну архітектуру системи повного циклу розрахункового дизайну лікарських речовин із заданими властивостями, яка поєднує описані вище методи (див. Розділ 5). Система орієнтована на гнучке масштабування за пікового навантаження та модульність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] Jeremy Hall, Stelvia Matos, Stefan Gold, and Liv S. Severino. “The paradox of sustainable innovation: The ‘Eroom’ effect (Moore’s law backwards)”. In: Computational and Theoretical Chemistry 172 (2018), pp. 3487–3497. DOI: 10.1016/j.jclepro.2017.07.162.
- [2] Jack W. Scannell, Alex Blanckley, Helen Boldon, and Brian Warrington. “Diagnosing the decline in pharmaceutical R&D efficiency”. In: Nature Reviews Drug Discovery 11 (2012), pp. 191–200. DOI: 10.1038/nrd3681.
- [3] Tymofii Nikolaienko, Oleksandr Gurbych, and Maksym Druchok. “Complex machine learning model needs complex testing: Examining predictability of molecular binding affinity by a graph neural network”. In: Journal of Computational Chemistry 43.10 (2022), pp. 728–739. DOI: 10.1002/jcc.26831.
- [4] Maksym Druchok, Dzvenymyra Yarish, Sofiya Garkot, Tymofii Nikolaenko, and Oleksandr Gurbych. “Ensembling machine learning models to boost molecular affinity prediction”. In: Computational Biology and Chemistry 93 (2021). DOI: 10.1016/j.compbiolchem.2021.107529.
- [5] Олександр Гурбич. “Метод мета-навчання для визначення молекулярної спорідненості”. In: Вісник Хмельницького національного університету 307.2 (2022), pp. 14–24. DOI: 10.31891/2307-5732-2022-307-2-14-24.
- [6] Dzvenymyra Yarish, Sofiya Garkot, Oleksandr Grygorenko, Dmytro Radchenko, Yurii Moroz, and Oleksandr Gurbych. “Advancing molecular graphs with descriptors for the prediction of chemical reaction yields”. In: Journal of Computational Chemistry 43.28 (2022), pp. 1887–1935. DOI: 10.1002/jcc.27016.
- [7] Олександр Гурбич. “Метод машинного навчання для створення нових лікарських речовин із заданими властивостями”. In: Наук. вісник Ужгород. ун-ту 40.1 (2022), pp. 126–145. DOI: 10.24144/2616-7700.2022.1(40).126-145.

- [8] Maksym Druchok, Dzvenymyra Yarish, Oleksandr Gurbych, and Mykola Maksymenko. “Toward efficient generation, correction, and properties control of unique drug-like structures”. In: Journal of Computational Chemistry 42.11 (2021), pp. 746–760. DOI: 10.1002/jcc.26494.
- [9] Oleksandr Gurbych and Maksym Prymachenko. “Method for reductive pruning of neural networks and its applications”. In: Computer Systems and Information Technologies 3 (2022), pp. 40–48. DOI: 10.31891/csit-2022-3-5.
- [10] Grygoriy Dolgonos, Alexey Tsukanov, Sergey Psakhie Psakhie, Oleg Lukin, Oleksandr Gurbych, and Alexander Shivanyuk Shivanyuk. “Theoretical studies of capsular complexes of C₂V-symmetrical resorcin[4]arene tetraesters with tetramethylammonium cation”. In: Computational and Theoretical Chemistry 1159 (2019), pp. 12–17. DOI: 10.1016/j.comptc.2019.05.006.
- [11] Maksym Druchok, Dzvenymyra Yarish, Oleksandr Gurbych, and Mykola Maksymenko. “Towards Efficient Generation, Correction and Properties Control of Unique Drug-like Structures”. In: ChemRxiv (2019). DOI: 10.26434/chemrxiv.9941858.v1.
- [12] Oleksandr Gurbych, Maksym Druchok, Dzvenymyra Yarish, and Sofiya Garkot. “High throughput screening with machine learning”. In: Proceedings of the Advances in Neural Information Processing Systems 32 (NeurIPS) (Vancouver, Canada). Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Dec. 2019. ISBN: 9781713807933. DOI: 10.48550/arXiv.2012.08275.
- [13] M. Dickson and J.P. Gagnon. In: Nat. Rev. Drug Discov. 3 (2004), pp. 417–429. DOI: 10.1038/nrd1382.
- [14] A. Jahan, M.Y. Ismail, S.M. Sapuan, and Mustapha F. In: Mater. Des. 31 (2010), pp. 696–705. DOI: 10.1016/j.matdes.2009.08.013.
- [15] A. Schuhmacher, O. Gassmann, and M. Hinder. In: J. Transl. Med. 14 (2016), p. 105. DOI: 10.1186/s12967-016-0838-4.

- [16] J.C. Babiarz. In FDA Regulatory affairs. A guide for prescription drugs, medicine, and biologics. Ed. by D.J. Pisano and D.S. Mantus. New York: Informa Healthcare, 2008, pp. 34–45.
- [17] E. Petrova. In Innovation and Marketing in the Pharmaceutical Industry. Ed. by M. Ding. New York: Springer-Verlag, 2014, pp. 34–45. DOI: 10.1007/978-1-4614-7801-0.
- [18] S. Kim, P.A. Thiessen, E.E Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B.A. Shoemaker, J. Wang, B. Yu, J. Zhang, and S.H. Bryant. In: Nucleic Acids Res. 44(D1) (2016), pp. D1202–13. DOI: 10.1093/nar/gkv951.
- [19] P. Kirkpatrick and C. Ellis. In: Nature 432 (2004), p. 823. DOI: 10.1038/432823a.
- [20] N. Bloom, C.I. Jones, J. Van Reenen, and M. Webb. In: American Economic Review 110(4) (2020), pp. 1104–1144. DOI: 10.3386/w23782.
- [21] Y. LeCun, Y. Bengio, and G. Hinton. In: Nature 521 (2015), pp. 436–444. DOI: 10.1038/nature14539.
- [22] G.B. Goh, N.O. Hodas, and A. Vishnu. In: J. Comput. Chem. 38 (2017), pp. 1291–1307. DOI: 10.1002/jcc.24764.
- [23] R. Miotto, F. Wang, S. Wang, X. Jiang, and J.T. Dudley. In: Brief. Bioinform. 19(6) (2018), pp. 1236–1246. DOI: 10.1093/bib/bbx044.
- [24] G. Schneider. In: Nat. Rev. Drug Discov. 17 (2018), pp. 97–113. DOI: 10.1038/nrd.2017.232.
- [25] J. Bostrom, D.G. Brown, R.J. Young, and G.M. Keseru. In: Nat. Rev. Drug Discov. 17 (2018), pp. 709–727. DOI: 10.1038/nrd.2018.116.
- [26] K.T. Butler, D.W. Davies, H. Cartwright, O. Isayev, and A. Walsh. In: Nature 559 (2018), pp. 547–555. DOI: 10.1038/s41586-018-0337-2.

- [27] A. Zhavoronkov, Y.A. Ivanenkov, A. Aliper, M.S. Veselov, V.A. Aladinskiy, A.V. Aladinskaya, V.A. Terentiev, D.A. Polykovskiy, M.D. Kuznetsov, A. Asadulaev, Y. Volkov, A. Zholus, R.R. Shayakhmetov, A. Zhebrak, L.I. Minaeva, B.A. Zagribelnyy, L.H. Lee, R. Soll, D. Madge, L. Xing, T. Guo, and A. Aspuru-Guzik. In: Nat. Biotechnol. 37 (2019), pp. 1038–1040. DOI: 10.1038/s41587-019-0224-x.
- [28] B.K. Shoichet, S.L. McGovern, B. Wei, and J.J. Irwin. “Lead discovery using molecular docking”. In: Current Opinion in Chemical Biology 6.4 (2002), pp. 439–446. DOI: [https://doi.org/10.1016/S1367-5931\(02\)00339-3](https://doi.org/10.1016/S1367-5931(02)00339-3).
- [29] N.S. Pagadala, K. Syed, and J. Tuszynski. “Software for molecular docking: a review”. In: Biophysical Reviews 9 (2017), pp. 91–102. DOI: 10.1007/s12551-016-0247-1.
- [30] Walter Filgueira de Azevedo Jr., ed. Docking Screens for Drug Discovery. New York, NY: Humana Press, 2019. DOI: 10.1007/978-1-4939-9752-7.
- [31] T.M. Frimurer, G.H. Peters, L.F. Iversen, H.S. Andersen, N.P.H. Møller, and O.H. Olsen. “Ligand-Induced Conformational Changes: Improved Predictions of Ligand Binding Conformations and Affinities”. In: Biophysical Journal 84 (4 2003), pp. 2273–2281. DOI: 10.1016/S0006-3495(03)75033-4.
- [32] Y.-C. Chen. “Beware of docking!” In: Trends in Pharmacological Sciences 36.2 (2015), pp. 78–95. DOI: 10.1016/j.tips.2014.12.001.
- [33] R.D. King, J.D. Hirst, and M.J.E. Sternberg. “Comparison of artificial intelligence methods for modeling pharmaceutical QSARs”. In: Applied Artificial Intelligence 9.2 (1995), pp. 213–233. DOI: 10.1080/08839519508945474.
- [34] R.N. Jorissen and M.K. Gilson. “Virtual Screening of Molecular Databases Using a Support Vector Machine”. In: Journal of Chemical Information and Modeling 45.3 (2005), pp. 549–561. DOI: 10.1021/ci049641u.

- [35] K. Yugandhar and M.M. Gromiha. “Feature selection and classification of protein-protein complexes based on their binding affinities using machine learning approaches”. In: Proteins: Structure, Function, and Bioinformatics 82.9 (2014), pp. 2088–2096. DOI: 10.1002/prot.24564.
- [36] L. Li, C.C. Koh, D. Reker, J.B. Brown, H. Wang, N.K. Lee, H.-h. Liow, H. Dai, H.-M. Fan, L. Chen, and D.-Q. Wei. “Predicting protein-ligand interactions based on bow-pharmacological space and Bayesian additive regression trees”. In: Scientific Reports 9 (2019), p. 7703. DOI: 10.1038/s41598-019-43125-6.
- [37] J.L. Durant, B.A. Leland, D.R. Henry, and J.G. Nourse. “Re-optimization of MDL Keys for Use in Drug Discovery”. In: Journal of Chemical Information and Computer Sciences 42.6 (2002), pp. 1273–1280. DOI: 10.1021/ci010132r.
- [38] G.S. Heck, V.O. Pintro, R.R. Pereira, M.B. de Avila, N.M.B. Levin, and W.F. de Azevedo Jr. “Supervised Machine Learning Methods Applied to Predict Ligand-Binding Affinity”. In: Current Medicinal Chemistry 24.23 (2017), pp. 2459–2470. DOI: 10.2174/0929867324666170623092503.
- [39] T. Pahikkala, A. Airola, S. Pietilä, S. Shakyawar, A. Szwajda, J. Tang, and T. Aittokallio. “Toward more realistic drug-target interaction predictions”. In: Briefings in Bioinformatics 16.2 (Apr. 2014), pp. 325–337. DOI: 10.1093/bib/bbu010.
- [40] T. He, M. Heidemeyer, F. Ban, A. Cherkasov, and M. Ester. “SimBoost: a read-across approach for predicting drug-target binding affinities using gradient boosting machines”. In: Journal of Cheminformatics 9 (2017), p. 24. DOI: 10.1186/s13321-017-0209-z.
- [41] H. Öztürk, A. Özgür, and E. Ozkirimli. “DeepDTA: deep drug-target binding affinity prediction”. In: Bioinformatics 34.17 (Sept. 2018), pp. i821–i829. DOI: 10.1093/bioinformatics/bty593.
- [42] H. Öztürk, E. Ozkirimli, and A. Özgür. WideDTA: prediction of drug-target binding affinity. 2019. eprint: arXiv:1902.04166.

- [43] T. Nguyen, H. Le, T.P. Quinn, T. Nguyen, T.D. Le, and S. Venkatesh. “GraphDTA: Predicting drug-target binding affinity with graph neural networks”. In: Bioinformatics (Oct. 2020). DOI: 10.1093/bioinformatics/btaa921.
- [44] J. Shim, Z.-Y. Hong, I. Sohn, and C. Hwang. “Prediction of drug-target binding affinity using similarity-based convolutional neural network”. In: Scientific Reports 11 (2021), p. 4416. DOI: 10.1038/s41598-021-83679-y.
- [45] M.I. Davis, J. Hunt, S. Herrgard, P. Ciceri, L. Wodicka, G. Pallares, M. Hocker, D. Treiber, and P. Zarrinkar. “Comprehensive analysis of kinase inhibitor selectivity”. In: Nature Biotechnology 29 (2011), pp. 1046–1051.
- [46] J. Tang, A. Szwajda, S. Shakyawar, T. Xu, P. Hintsanen, K. Wennerberg, and K. Aittokallio. “Making Sense of Large-Scale Kinase Inhibitor Bioactivity Data Sets: A Comparative and Integrative Analysis”. In: Journal of Chemical Information and Modeling 54(3) (2014), pp. 735–43.
- [47] D.J. Lipman and W.R. Pearson. “Rapid and sensitive protein similarity searches”. In: Science 227.4693 (1985), pp. 1435–1441. DOI: 10.1126/science.2983426.
- [48] W.R. Pearson and D.J. Lipman. “Improved tools for biological sequence comparison”. In: Proceedings of the National Academy of Sciences 85.8 (1988), pp. 2444–2448. DOI: 10.1073/pnas.85.8.2444.
- [49] D. Weininger. “SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules”. In: Journal of Chemical Information and Computer Sciences 28.1 (1988), pp. 31–36. DOI: 10.1021/ci00057a005.
- [50] D. Weininger, A. Weininger, and J.L. Weininger. “SMILES. 2. Algorithm for generation of unique SMILES notation”. In: Journal of Chemical Information and Computer Sciences 29.2 (1989), pp. 97–101. DOI: 10.1021/ci00062a008.

- [51] I. Kundu, G. Paul, and R. Banerjee. “A machine learning approach towards the prediction of protein-ligand binding affinity based on fundamental molecular properties”. In: RSC Adv. 8 (22 2018), pp. 12127–12137. DOI: 10.1039/C8RA00003D.
- [52] R. Wang, X. Fang, Y. Lu, and S. Wang. “The PDBbind Database: Collection of Binding Affinities for Protein-Ligand Complexes with Known Three-Dimensional Structures”. In: Journal of Medicinal Chemistry 47.12 (2004), pp. 2977–2980. DOI: 10.1021/jm0305801. eprint: <https://doi.org/10.1021/jm0305801>. URL: <https://doi.org/10.1021/jm0305801>.
- [53] M. Jiang, Z. Li, S. Zhang, S. Wang, X. Wang, Q. Yuan, and Z. Wei. “Drug–target affinity prediction using graph neural network and contact maps”. In: RSC Advances 10 (35 2020), pp. 20701–20712. DOI: 10.1039/D0RA02297G.
- [54] M. Michel, D. Menéndez Hurtado, and A. Elofsson. “PconsC4: fast, accurate and hassle-free contact predictions”. In: Bioinformatics 35.15 (2018), pp. 2677–2679. DOI: 10.1093/bioinformatics/bty1036.
- [55] J. Jiménez, M. Škalič, G. Martínez-Rosell, and G. De Fabritiis. “ K_{DEEP} : Protein-Ligand Absolute Binding Affinity Prediction via 3D-Convolutional Neural Networks”. In: Journal of Chemical Information and Modeling 58.2 (2018), pp. 287–296. DOI: 10.1021/acs.jcim.7b00650.
- [56] Y. Li, M.A. Rezaei, C. Li, and X. Li. “DeepAtom: A Framework for Protein-Ligand Binding Affinity Prediction”. In: 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBI 2019), pp. 303–310.
- [57] M.J. Hartshorn, M.L. Verdonk, G. Chessari, S.C. Brewerton, W.T.M. Mooij, P.N. Mortenson, and C.W. Murray. “Diverse, High-Quality Test Set for the Validation of Protein-Ligand Docking Performance”. In: Journal of Medicinal Chemistry 50.4 (2007), pp. 726–741. DOI: 10.1021/jm061277y.

- [58] Y. Kwon, W.-H. Shin, J. Ko, and J. Lee. “AK-Score: Accurate Protein-Ligand Binding Affinity Prediction Using an Ensemble of 3D-Convolutional Neural Networks”. In: International Journal of Molecular Sciences 21.22 (2020), p. 8424. DOI: [10.3390/ijms21228424](https://doi.org/10.3390/ijms21228424).
- [59] J.-Q. Chen, H.-Y. Chen, W.-j. Dai, Q.-J. Lv, and C.Y.-C. Chen. “Artificial Intelligence Approach to Find Lead Compounds for Treating Tumors”. In: Journal of Physical Chemistry Letters 10.15 (2019), pp. 4382–4400. DOI: [10.1021/acs.jpcllett.9b01426](https://doi.org/10.1021/acs.jpcllett.9b01426).
- [60] M. Schneider, J.-L. Pons, W. Bourguet, and G. Labesse. “Towards accurate high-throughput ligand affinity prediction by exploiting structural ensembles, docking metrics and ligand similarity”. In: Bioinformatics 36.1 (July 2019), pp. 160–168. DOI: [10.1093/bioinformatics/btz538](https://doi.org/10.1093/bioinformatics/btz538).
- [61] P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. Hunter, C. Bekas, and A. Lee. “Molecular Transformer: A Model for Uncertainty-Calibrated Chemical Reaction Prediction”. In: ACS Central Science 5 (2019), pp. 1572–1583.
- [62] J. Payne, M. Srouji, D.A. Yap, and V. Kosaraju. BERT Learns (and Teaches) Chemistry. 2020. eprint: [arXiv : 2007.16012](https://arxiv.org/abs/2007.16012).
- [63] A. Rives, S. Goyal, J. Meier, D. Guo, M. Ott, C.L. Zitnick, J. Ma, and R. Fergus. Biological Structure and Function Emerge from Scaling Unsupervised Learning. 2019. DOI: [10.1101/622803](https://doi.org/10.1101/622803). eprint: [bioRxiv](https://bioRxiv.org/).
- [64] B. Shin, S. Park, K. Kang, and J. Ho. Self-Attention Based Molecule Representation for Predicting Drug-Target Interactions. 2019. eprint: [arXiv:1908.06760](https://arxiv.org/abs/1908.06760).
- [65] M.S. Mottaqi, F. Mohammadipanah, and H. Sajedi. “Contribution of machine learning approaches in response to SARS-CoV-2 infection”. In: Informatics in Medicine Unlocked 23 (2021), p. 100526. DOI: <https://doi.org/10.1016/j.imu.2021.100526>.

- [66] Z. Jin, X. Du, Y. Xu, Y. Deng, M. Liu, Y. Zhao, B. Zhang, X. Li, L. Zhang, C. Peng, Y. Duan, J. Yu, L. Wang, K. Yang, F. Liu, R. Jiang, X. Yang, T. You, X. Liu, X. Yang, F. Bai, H. Liu, X. Liu, L.W. Guddat, W. Xu, G. Xiao, C. Qin, Z. Shi, H. Jiang, Z. Rao, and H. Yang. “Structure of M^{pro} from SARS-CoV-2 and discovery of its inhibitors”. In: Nature 582 (2020), pp. 289–293. DOI: 10.1038/s41586-020-2223-y.
- [67] K. Gao, D.D. Nguyen, J. Chen, R. Wang, and G.-W. Wei. “Repositioning of 8565 Existing Drugs for COVID-19”. In: Journal of Physical Chemistry Letters 11.13 (2020), pp. 5373–5382. DOI: 10.1021/acs.jpcllett.0c01579.
- [68] M. Nand, P. Maiti, T. Joshi, S. Chandra, V. Pande, J.C. Kuniyal, and M.A. Ramakrishnan. “Virtual screening of anti-HIV1 compounds against SARS-CoV-2: machine learning modeling, chemoinformatics and molecular dynamics simulation based analysis”. In: Scientific Reports 10 (2020), p. 20397. DOI: 10.1038/s41598-020-77524-x.
- [69] M.V.S. Santana and F.P. Silva-Jr. “De novo design and bioactivity prediction of SARS-CoV-2 main protease inhibitors using recurrent neural network-based transfer learning”. In: BMC Chemistry 15 (2020), p. 8. DOI: 10.1186/s13065-021-00737-2.
- [70] D. Mendez, A. Gaulton, A. Bento, J. Chambers, M.D. Veij, E. Félix, M.P. Magariños, J. Mosquera, P. Mutowo-Meullenet, M. Nowotka, M. Gordillo-Marañón, F. Hunter, L. Junco, G. Mugumbate, M. Rodriguez-Lopez, F. Atkinson, N. Bosc, C.J. Radoux, A. Segura-Cabrera, A. Hersey, and A. Leach. “ChEMBL: towards direct deposition of bioassay data”. In: Nucleic Acids Research 47 (2019), pp. D930–D940.
- [71] J. Kowalewski and A. Ray. “Predicting novel drugs for SARS-CoV-2 using machine learning from a >10 million chemical space”. In: Helion 6 (8 2020), e04639. DOI: 10.1016/j.heliyon.2020.e04639.
- [72] B.R. Beck, B. Shin, Y. Choi, S. Park, and K. Kang. “Predicting commercially available antiviral drugs that may act on the novel coronavirus (SARS-CoV-2) through a drug-target interaction deep learning model”. In: Computational and Structural Biotechnology Journal 18 (2020), pp. 784–790. DOI: 10.1016/j.csbj.2020.03.025.

- [73] O. Kadioglu, M. Saeed, H.J. Greten, and T. Efferth. “Identification of novel compounds against three targets of SARS CoV-2 coronavirus by combined virtual screening and supervised machine learning”. In: Computers in Biology and Medicine 133 (2021), p. 104359. DOI: 10.1016/j.combiomed.2021.104359.
- [74] Oleg Trott and Arthur Olson. “AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading”. In: Journal of Computational Chemistry 31 (2009), pp. 455–461. DOI: 10.1002/jcc.21334.
- [75] Jerome Eberhardt, Diogo Santos-Martins, Andreas Tillack, and Stefano Forli. “AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings”. In: J. Chem. Inf. Model. 61 (2021), pp. 3891–3898. DOI: 10.1021/acs.jcim.1c00203.
- [76] S.R. Ellingson, B. Davis, and J. Allen. “Machine learning and ligand binding predictions: A review of data, methods, and obstacles”. In: Biochimica et Biophysica Acta (BBA) - General Subjects 1864.6 (2020), p. 129545. DOI: 10.1016/j.bbagen.2020.129545.
- [77] Robin Winter, Floriane Montanari, Frank Noé, and Djork-Arné Clevert. “Learning Continuous and Data-Driven Molecular Descriptors by Translating Equivalent Chemical Representations”. In: Chemical Science 10 (Jan. 2019), pp. 1692–1701. DOI: 10.1039/C8SC04175J.
- [78] Rafael Gómez-Bombarelli, David Duvenaud, José Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy Hirzel, Ryan Adams, and Alán Aspuru-Guzik. “Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules”. In: ACS Central Science 4.2 (Oct. 2016), pp. 268–276. DOI: 10.1021/acscentsci.7b00572.
- [79] Benjamin Sanchez, Carlos Outeiral, Gabriel Guimaraes, and Alán Aspuru-Guzik. Optimizing distributions over molecular space. An Objective-Reinforced Generative Model. Aug. 2017. DOI: 10.26434/chemrxiv.5309668.v3. eprint: ChemRxiv: 10.26434/chemrxiv.5309668.v3.

- [80] Oleksii Prykhodko, Simon Johansson, Panagiotis-Christos Kotsias, Josep Arús-Pous, Esben Bjerrum, Ola Engkvist, and Hongming Chen. “A de novo molecular generation method using latent vector based generative adversarial network”. In: Journal of Cheminformatics 11 (Dec. 2019). DOI: 10.1186/s13321-019-0397-9.
- [81] Marwin Segler, Thierry Kogej, Christian Tyrchan, and Mark Waller. “Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks”. In: ACS Central Science 4 (Jan. 2017). DOI: 10.1021/acscentsci.7b00512.
- [82] Peter Ertl, Richard Lewis, Eric Martin, and Valery Polyakov. In silico generation of novel, drug-like chemical matter using the LSTM neural network. 2018. eprint: arXiv:1712.07449.
- [83] Maksym Druchok, Dzvenymyra Yarish, Oleksandr Gurbych, and Mykola Maksymenko. “Toward efficient generation, correction, and properties control of unique drug-like structures”. In: Journal of Computational Chemistry 42.11 (2021), pp. 746–760. DOI: 10.1002/jcc.26494.
- [84] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. “Deep reinforcement learning for de novo drug design”. In: Science Advances 4.7 (2018). ISSN: 2375-2548. DOI: 10.1126/sciadv.aap7885.
- [85] Bogdan Zagribelnyy, Alexander Zhavoronkov, Alex Aliper, Daniil Polykovskiy, Victor Terentiev, Vladimir Aladinskiy, Mark Veselov, Anastasiia Aladinskaia, Arip Asadulaev, Alexander Zhebrak, Lennart Lee, Richard Soll, David Madge, Li Xing, Tao Guo, Alán Aspuru-Guzik, Yan Ivanenkov, and Rim Shayakhmetov. “Deep learning enables rapid identification of potent DDR1 kinase inhibitors”. In: Nature Biotechnology 37 (Sept. 2019). DOI: 10.1038/s41587-019-0224-x.
- [86] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. “Molecular De Novo Design through Deep Reinforcement Learning”. In: Journal of Cheminformatics 9.48 (Sept. 2017). DOI: 10.1186/s13321-017-0235-x.

- [87] Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. “Optimizing Chemical Reactions with Deep Reinforcement Learning”. In: ACS Central Science 3.12 (2017), pp. 1337–1344. DOI: 10.1021/acscentsci.7b00492.
- [88] Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. “Optimizing Chemical Reactions with Deep Reinforcement Learning”. In: ACS Central Science 3.12 (2017), pp. 1337–1344. DOI: 10.1021/acscentsci.7b00492.
- [89] Marwin Segler, M. Preuss, and M. Waller. “Planning chemical syntheses with deep neural networks and symbolic AI”. In: Nature 555 (2018), pp. 604–610.
- [90] J. Schreck, Connor W. Coley, and K. Bishop. “Learning Retrosynthetic Planning through Simulated Experience”. In: ACS Central Science 5 (2019), pp. 970–981.
- [91] Shigeharu Kite, Tadashi Hattori, and Yuichi Murakami. “Estimation of catalytic performance by neural network — product distribution in oxidative dehydrogenation of ethylbenzene”. In: Applied Catalysis A: General 114.2 (1994), pp. L173–L178. ISSN: 0926-860X. DOI: [https://doi.org/10.1016/0926-860X\(94\)80169-X](https://doi.org/10.1016/0926-860X(94)80169-X).
- [92] Paul Raccuglia, Katherine C. Elbert, Philip Adler, Casey Falk, M. Wenny, Aurelio Mollo, M. Zeller, Sorelle A. Friedler, J. Schrier, and A. Norquist. “Machine-learning-assisted materials discovery using failed experiments”. In: Nature 533 (2016), pp. 73–76.
- [93] Craig L. Senese, J. Duca, D. Pan, A. J. Hopfinger, and Y. J. Tseng. “4D-Fingerprints, Universal QSAR and QSPR Descriptors”. In: Journal of Chemical Information and Computer Sciences 44.5 (2004), pp. 1526–1539. DOI: 10.1021/ci049898s.
- [94] David Rogers and Mathew Hahn. “Extended-Connectivity Fingerprints”. In: Journal of Chemical Information and Modeling 50.5 (2010), pp. 742–754. DOI: 10.1021/ci100050t.

- [95] Hirotomo Moriwaki, Yu-Shi Tian, Norihito Kawashita, and Tatsuya Takagi. “Mordred: a molecular descriptor calculator.” In: Journal of Cheminformatics 10.4 (2018).
- [96] E.N. Bess, A.J. Bischoff, and Sigman M.S. “Designer substrate library for quantitative, predictive modeling of reaction performance”. In: Proceedings of the National Academy of Sciences of the United States of America 111 (2014).
- [97] Anat Milo, Andrew Neel, Dean Toste, and Matthew Sigman. “A data-intensive approach to mechanistic elucidation applied to chiral anion catalysis”. In: Science 347.6223 (2015), pp. 737–743. ISSN: 0036-8075. DOI: 10.1126/science.1261043.
- [98] A. Milo, E.N. Bess, and M.S. Sigman. “Interrogating selectivity in catalysis using molecular vibrations”. In: Nature 507 (2014), pp. 210–214.
- [99] Akira Yada, Kenji Nagata, Yasunobu Ando, Tarojiro Matsumura, Sakina Ichinoseki, and Kazuhiko Sato. “Machine Learning Approach for Prediction of Reaction Yield with Simulated Catalyst Parameters”. In: Chemistry Letters 47.3 (2018), pp. 284–287. DOI: 10.1246/c1.171130.
- [100] M. K. Nielsen, D.T. Ahneman, O. Riera, and A. G. Doyle. “Deoxyfluorination with Sulfonyl Fluorides: Navigating Reaction Space with Machine Learning”. In: Journal of the American Chemical Society 140.15 (2018), pp. 5004–5008. DOI: 10.1021/jacs.8b01523.
- [101] Natalie S. Eyke, W. Green, and K. Jensen. “Iterative experimental design based on active machine learning reduces the experimental burden associated with reaction screening”. In: Reaction Chemistry and Engineering 5 (2020), pp. 1963–1972.
- [102] Zunyun Fu, Xutong Li, Z. Wang, Zhao-jun Li, X. Liu, Xiaolong Wu, J. Zhao, Xiaoyu Ding, Xiaozhe Wan, Feisheng Zhong, Dingyan Wang, X. Luo, K. Chen, Hong Liu, Jiang Wang, Heyan Jiang, and M. Zheng. “Optimizing chemical reaction conditions using deep learning: a case study for the Suzuki–Miyaura cross-coupling reaction”. In: Organic chemistry frontiers 7 (2020), pp. 2269–2277.

- [103] Derek T. Ahneman, Jesús G. Estrada, Shishi Lin, Spencer D. Dreher, and Abigail G. Doyle. “Predicting reaction performance in C–N cross-coupling using machine learning”. In: Science 360.6385 (2018), pp. 186–190. DOI: 10.1126/science.aar5169.
- [104] Damith Perera, Joseph W. Tucker, Shalini Brahmabhatt, Christopher J. Helal, Ashley Chong, William Farrell, Paul Richardson, and Neal W. Sach. “A platform for automated nanomole-scale reaction screening and micromole-scale synthesis in flow”. In: Science 359.6374 (2018), pp. 429–434. ISSN: 0036-8075. DOI: 10.1126/science.aap9112.
- [105] Kangway V. Chuang and Michael J. Keiser. “Comment on “Predicting reaction performance in C–N cross-coupling using machine learning””. In: Science 362.6416 (2018). ISSN: 0036-8075. DOI: 10.1126/science.aat8603.
- [106] Frederik Sandfort, Felix Strieth-Kalthoff, Marius Kühnemund, C. Beecks, and F. Glorius. “A Structure-Based Platform for Predicting Chemical Reactivity”. In: Chem 6 (2019), pp. 1379–1390.
- [107] S. Kariofillis, S. Jiang, A. Żurański, S. Gandhi, J. Martinez Alvarado, and A. Doyle. “Using Data Science to Guide Aryl Bromide Substrate Scope Analysis in a Ni /Photoredox-Catalyzed Cross-Coupling with Acetals as Alcohol-Derived Radical Sources”. In: Journal of the American Chemical Society 144 (2021), pp. 1045–1055. ISSN: 0002-7863.
- [108] Reaxys Database. 2017. eprint: www.reaxys.com.
- [109] J. Granda, Liva Donina, Vincenza Dragone, D. Long, and L. Cronin. “Controlling an organic synthesis robot with machine learning to search for new reactivity”. In: Nature 559 (2018), pp. 377–381.
- [110] Brandon J. Reizman, Yi-Ming Wang, Stephen L. Buchwald, and Klavs F. Jensen. “Suzuki–Miyaura cross-coupling optimization enabled by automated feedback”. In: React. Chem. Eng. 1 (6 2016), pp. 658–666. DOI: 10.1039/C6RE00153J.

- [111] Andrzej M. Zuranski, Jesus I. Martinez Alvarado, Benjamin J. Shields, and Abigail G. Doyle. "Predicting Reaction Yields via Supervised Learning". In: Accounts of Chemical Research 54.8 (2021), pp. 1856–1865. DOI: 10.1021/acs.accounts.0c00770.
- [112] G. Skoraczyński, P. Dittwald, B. Miasojedow, S. Szymkuć, E. P. Gajewska, B. Grzybowski, and A. Gambin. "Predicting the outcomes of organic reactions via machine learning: are current descriptors sufficient?" In: Scientific Reports 7 (2017).
- [113] Greg Landrum. RDKit: Open-source cheminformatics. URL: <http://www.rdkit.org>,.
- [114] Shion Honda, Shoi Shi, and H. Ueda. SMILES Transformer: Pre-trained Molecular Fingerprint for Low Data Drug D 2019. eprint: [arXiv:1911.04738](https://arxiv.org/abs/1911.04738).
- [115] Seyone Chithrananda, Gabriel Grand, and Bharath Ramsundar. ChemBERTa: Large-Scale Self-Supervised Pretraining for Molecular Property 2020. eprint: [arXiv:2010.09885](https://arxiv.org/abs/2010.09885).
- [116] Philippe Schwaller, D. Probst, Alain C. Vaucher, Vishnu H. Nair, David Kreutter, Teodoro Laino, and Jean-Louis | Reymond. Mapping the Space of Chemical Reactions Using Attention-Based Neural Netw 2020. eprint: [arXiv:2012.06051](https://arxiv.org/abs/2012.06051).
- [117] P. Schwaller, Riccardo Petraglia, Valerio Zullo, Vishnu H. Nair, Rico Häuselmann, Riccardo Pisoni, C. Bekas, A. Iuliano, and T. Laino. "Predicting retrosynthetic pathways using transformer-based models and a hyper-graph exploration strategy". In: Chemical Science 11 (2020), pp. 3316–3325.
- [118] Philippe Schwaller, Alain C Vaucher, Teodoro Laino, and Jean-Louis Reymond. "Prediction of chemical reaction yields using deep learning". In: Machine Learning: Science and Technology 2.1 (Mar. 2021), p. 015016. DOI: 10.1088/2632-2153/abc81d.
- [119] 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, Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.
- [120] J. Mayfield, D. Lowe, and R. Sayle. Pistachio. Version 2022-04-03. Reaction Data, Querying and Analytics. 2018. eprint: www.nextmovesoftware.com/pistachio.html.
- [121] Daniel Lowe. Chemical reactions from US patents (1976-Sep2016). figshare. Data. 2017. DOI: 10.6084/m9.figshare.5104873.v1. eprint: <https://doi.org/10.6084/m9.figshare.5104873.v1>.
- [122] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, Timothy D. Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. 2015. eprint: [arXiv:1509.09292v2](https://arxiv.org/abs/1509.09292v2).
- [123] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, and Alvaro Sanchez-Gonzalez. Relational inductive biases, deep learning, and graph networks. 2018. eprint: [arXiv:1806.01261](https://arxiv.org/abs/1806.01261).
- [124] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph Neural Networks: A Review of Methods and Applications. 2021. eprint: [arXiv:1812.08434](https://arxiv.org/abs/1812.08434).
- [125] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2978386.
- [126] Steph-Yves Louis, Yong Zhao, Alireza Nasiri, Xiran Wang, Yuqi Song, Fei Liu, and Jianjun Hu. “Graph convolutional neural networks with global attention for improved materials property prediction”. In: *Phys. Chem. Chem. Phys.* 22 (32 2020), pp. 18141–18148. DOI: 10.1039/DOCP01474E.

- [127] Zeren Shui and George Karypis. Heterogeneous Molecular Graph Neural Networks. 2020. DOI: 10.1109/ICDM50108.2020.00058. eprint: arXiv:2009.12710.
- [128] Ziyue Yang, Maghesree Chakraborty, and Andrew D. White. “Predicting chemical shifts with graph neural networks”. In: Chem. Sci. 12 (2021), pp. 10802–10809. DOI: 10.1039/D1SC01895G.
- [129] Wen Torng and Russ B. Altman. “Graph Convolutional Neural Networks for Predicting Drug-Target Interactions”. In: Journal of Chemical Information and Modeling 10 (2018), pp. 4131–4149. DOI: 10.1101/473074.
- [130] Maksym Druchok, Dzvenymyra Yarish, Sofiya Garkot, Tymofii Nikolaienko, and Oleksandr Gurbych. “Ensembling machine learning models to boost molecular affinity prediction”. In: Computational Biology and Chemistry 93 (2021), p. 107529. ISSN: 1476-9271. DOI: 10.1016/j.compbiolchem.2021.107529.
- [131] Victor Fung, Jiaxin Zhang, Eric Juarez, and Bobby Sumpter. “Benchmarking graph neural networks for materials chemistry”. In: npg Computational Materials 7.84 (Dec. 2021). DOI: 10.1038/s41524-021-00554-0.
- [132] Mandana Saebi, Bozhao Nan, John Herr, Jessica Wahlers, Olaf Wiest, and Nitesh Chawla. Graph Neural Networks for Predicting Chemical Reaction Performance. 2021. eprint: ChemRxiv:10.26434/chemrxiv.14589498.
- [133] Jian Du, Shanghang Zhang, Guanhang Wu, José M. F. Moura, and S. Kar. Topology adaptive graph convolutional networks. 2018. eprint: arXiv:1710.10370v5.
- [134] William L. Hamilton, Zhitao Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. 2018. eprint: arXiv:1706.02216v4.
- [135] Y. Li, Daniel Tarlow, Marc Brockschmidt, and R. Zemel. Gated Graph Sequence Neural Networks. 2017. eprint: arXiv:1511.05493v4.

- [136] Petar Velickovic, Guillem Cucurull, A. Casanova, Adriana Romero, P. Lio', and Yoshua Bengio. Graph Attention Networks. 2018. eprint: [arXiv:1710.10903v3](https://arxiv.org/abs/1710.10903v3).
- [137] H. Dai, Bo Dai, and L. Song. Discriminative Embeddings of Latent Variable Models. 2020. eprint: [arXiv:1603.05629v5](https://arxiv.org/abs/1603.05629v5).
- [138] Kevin Yang, Kyle Swanson, Wengong Jin, Connor W. Coley, Philipp Eiden, H. Gao, A. Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, T. Jaakkola, K. Jensen, and R. Barzilay. "Analyzing Learned Molecular Representations for Property Prediction". In: Journal of Chemical Information and Modeling 59 (2019), pp. 3370–3388.
- [139] Colin A. Grambow, L. Pattanaik, and W. Green. "Deep Learning of Activation Energies". In: The Journal of Physical Chemistry Letters 11 (2020), pp. 2992–2997.
- [140] M.O. Steinhauser and S. Hiermaier. In: Int. J. Mol. Sci. 10(12) (2009), pp. 5135–5216. DOI: [10.3390/ijms10125135](https://doi.org/10.3390/ijms10125135).
- [141] J. Behler. In: Chemical Modelling: Applications and Theory 7 (2010), pp. 1–41. DOI: [10.1039/9781849730884-00001](https://doi.org/10.1039/9781849730884-00001).
- [142] S.A. Ghasemi, A. Hofstetter, S. Saha, and S. Goedecker. In: Phys. Rev. B 92 (2015), p. 045131. DOI: [10.1103/PhysRevB.92.045131](https://doi.org/10.1103/PhysRevB.92.045131).
- [143] K.T. Schutt, F. Arbabzadah, S. Chmiela, K.R. Muller, and A. Tkatchenko. In: Nat. Comm. 8 (2017), p. 13890. DOI: [10.1038/ncomms13890](https://doi.org/10.1038/ncomms13890).
- [144] J. Carrasquilla and R.G. Melko. In: Nat. Phys. 13 (2017), pp. 431–434. DOI: [10.1038/nphys4035](https://doi.org/10.1038/nphys4035).
- [145] T. Xie and J.C. Grossman. In: Phys. Rev. Lett. 120 (2018), p. 145301. DOI: [10.1103/PhysRevLett.120.145301](https://doi.org/10.1103/PhysRevLett.120.145301).
- [146] K. Ryan, J. Lengyel, and M. Shatruk. In: J. Am. Chem. Soc. 140(32) (2018), pp. 10158–10168. DOI: [10.1021/jacs.8b03913](https://doi.org/10.1021/jacs.8b03913).

- [147] S. Amabilino, L.A. Bratholm, S.J. Bennie, A.C. Vaucher, M. Reiher, and D.R. Glowacki. In: J. Phys. Chem. A 123(20) (2019), pp. 4486–4499. DOI: 10.1021/acs.jpca.9b01006.
- [148] F.E. Bock, R.C. Aydin, C.J. Cyron, N. Huber, S.R. Kalidindi, and B. Klusemann. In: Front. Mater. 6:110 (2019). DOI: 10.3389/fmats.2019.00110.
- [149] M. Haghghatlari and J. Hachmann. In: Curr. Opin. Chem. Eng. 23 (2019), pp. 51–57. DOI: 10.1016/j.coche.2019.02.009.
- [150] S. Chiriki and S.S. Bulusu. In: Chem. Phys. Lett. 652 (2016), pp. 130–135. DOI: 10.1016/j.cplett.2016.04.013.
- [151] L. Shen and W. Yang. In: J. Chem. Theory Comput. 14 (2018), pp. 1442–1455. DOI: 10.1021/acs.jctc.7b01195.
- [152] S. Jindal and S.S. Bulusu. In: J. Chem. Phys. 149 (2018), p. 194101. DOI: 10.1063/1.5043247.
- [153] R. Kondor. A transferable artificial neural network model for atomic forces in 2018. eprint: arXiv:1810.06204.
- [154] K.T. Schutt, H.E. Saucedo, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. In: J. Chem. Phys. 148 (2018), p. 241722. DOI: 10.1063/1.5019779.
- [155] A. Perez, G. Martinez-Rosell, and G. De Fabritiis. In: Curr. Opin. Struct. Biol. 49 (2018), pp. 139–144. DOI: 10.1016/j.sbi.2018.02.004.
- [156] J. Herr, K. Yao, K. McIntyre, D.W. Toth, and J. Parkhill. In: J. Chem. Phys. 148 (2018), p. 241710. DOI: 10.1063/1.5020067.
- [157] K. Yao, J.E. Herr, D.W. Toth, R. MckIntyre, and J. Parkhill. In: Chem. Sci. 9 (2018), pp. 2261–2269. DOI: 10.1039/C7SC04934J.
- [158] H. Wang, L. Zhang, J. Han, and W. E. In: Comput. Phys. Commun. 228 (2018), pp. 178–184. DOI: 10.1016/j.cpc.2018.03.016.
- [159] L. Zhang, H. Wang, J. Han, R. Car, and W. E. In: Phys. Rev. Lett. 120 (2018), p. 143001. DOI: 10.1103/PhysRevLett.120.143001.

- [160] L. Zhang, H. Wang, and W. E. In: J. Chem. Phys. 149 (2018), p. 154107. DOI: 10.1063/1.5042714.
- [161] L. Zhang, J. Han, H. Wang, R. Car, and W. E. In: J. Chem. Phys. 149 (2018), p. 034101. DOI: 10.1063/1.5027645.
- [162] A. Lusci, G. Pollastri, and P. Baldi. In: J. Chem. Inf. Model. 53 (2013), pp. 1563–1575. DOI: 10.1021/ci400187y.
- [163] G.E. Dahl, N. Jaitly, and R. Salakhutdinov. Multi-task Neural Networks for QSAR Predictions. 2014. eprint: [arXiv:1406.1231](https://arxiv.org/abs/1406.1231).
- [164] E.O. Pyzer-Knapp, K. Li, and A. Aspuru-Guzik. In: Adv. Funct. Mater. 25 (2015), pp. 6495–6502. DOI: 10.1002/adfm.201501919.
- [165] B. Alipanahi, A. DeLong, M.T. Weirauch, and B.J. Frey. In: Nature Biotechnol 33 (2015), pp. 831–838. DOI: 10.1038/nbt.3300.
- [166] I. Wallach, M. Dzamba, and A. Heifets. AtomNet: A Deep Convolutional Neural Network for Bioactivity Prediction in 2015. eprint: [arXiv:1510.02855](https://arxiv.org/abs/1510.02855).
- [167] A. Mayr, G. Klambauer, T. Unterthiner, and S. Hochreiter. In: Front. Environ. Sci. 3:80 (2016). DOI: 10.3389/fenvs.2015.00080.
- [168] E.J. Bjerrum. SMILES Enumeration as Data Augmentation for Neural Network 2017. eprint: [arXiv:1703.07076](https://arxiv.org/abs/1703.07076).
- [169] A.K. Sharma, G.N. Srivastava, A. Roy, and V.K. Sharma. In: Front. Pharmacol. 8:880 (2017). DOI: 10.3389/fphar.2017.00880.
- [170] S. Kearnes, B. Goldman, and V. Pande. Modeling Industrial ADMET Data with Multitask Networks. 2017. eprint: [arXiv:1606.08793](https://arxiv.org/abs/1606.08793).
- [171] G.B. Goh, N.O. Hodas, C. Siegel, and A. Vishnu. SMILES2Vec: An Interpretable General-Purpose Deep Neural Network for Pre 2018. eprint: [arXiv:1712.02034](https://arxiv.org/abs/1712.02034).
- [172] G.B. Goh, C. Siegel, A. Vishnu, and N.O. Hodas. Using Rule-Based Labels for Weak Supervised Learning: A ChemNet for Tran 2018. eprint: [arXiv:1712.02734](https://arxiv.org/abs/1712.02734).

- [173] N. Ståhl, G. Falkman, A. Karlsson, G. Mathiason, and J. Boström. In: J. Integr. Bioinform. 20180065 (2018), pp. 1613–4516. DOI: 10.1515/jib-2018-0065.
- [174] C.A. Lipinski, F. Lombardo, B.W. Dominy, and P.J. Feeney. In: Adv. Drug Deliv. Rev. 46 (2018), pp. 3–26. DOI: 10.1016/S0169-409X(96)00423-1.
- [175] A.K. Ghose, V.N. Viswanadhan, and J.J. Wendoloski. In: J. Comb. Chem. 1 (1999), pp. 55–68. DOI: 10.1021/cc9800071.
- [176] W.J. Egan, K.M. Merz, and J.J. Baldwin. In: J. Med. Chem. 43 (2000), pp. 3867–3877. DOI: 10.1021/jm000292e.
- [177] I. Muegge, S.L. Heald, and D. Brittelli. In: J. Med. Chem. 44 (2001), pp. 1841–1846. DOI: 10.1021/jm015507e.
- [178] D.F. Veber, S.R. Johnson, Cheng H.Y., B.R. Smith, K.W. Ward, and K.D. Kopple. In: J. Med. Chem. 45 (2002), pp. 2615–2623. DOI: 10.1021/jm020017n.
- [179] M.H. Segler, T. Kogej, C. Tyrchan, and M.P. Waller. In: ACS Cent. Sci. 4(1) (2018), pp. 120–131. DOI: 10.1021/acscentsci.7b00512.
- [180] M.J. Kusner, B. Paige, and J.M. Hernández-Lobato. Grammar Variational Autoencoder. 2017. eprint: arXiv : 1703 . 01925v1.
- [181] G.B. Goh, C. Siegel, A. Vishnu, N.O. Hodas, and N. Baker. Chemistry Does a Deep Neural Network Need to Know to Make Accurate Pre 2018. eprint: arXiv:1710.02238.
- [182] G.B. Goh, K. Sakloth, C. Siegel, A. Vishnu, and J. Pfandtner. Multimodal Deep Neural Networks using Both Engineered and Learned Repre 2018. eprint: arXiv:1808.04456.
- [183] D. Kuzminykh, D. Polykovskiy, A. Kadurin, A. Zhebrak, I. Baskov, S. Nikolenko, R. Shayakhmetov, and A. Zhavoronkov. In: Mol. Pharmaceutics 15 (2018), pp. 4378–4385. DOI: 10.1021/acs.molpharmaceut.7b01134.

- [184] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P.S. Yu. A Comprehensive Survey on Graph Neural Networks. 2019. eprint: [arXiv:1901.00596v2](https://arxiv.org/abs/1901.00596v2).
- [185] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph Neural Networks: A Review of Methods and Applications. 2019. eprint: [arXiv:1812.08434v3](https://arxiv.org/abs/1812.08434v3).
- [186] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley. “Molecular graph convolutions: moving beyond fingerprints”. In: Journal of Computer-Aided Molecular Design 30 (2016), pp. 595–608. DOI: [10.1007/s10822-016-9938-8](https://doi.org/10.1007/s10822-016-9938-8).
- [187] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R.P. Adams. In: Advances in Neural Information Processing Systems 28 (NIPS 2015). Montreal, Canada: Curran Associates, Inc.: Red Hook, NY, 2016, pp. 2224–2232.
- [188] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. 2019. eprint: [arXiv:1806.02473v3](https://arxiv.org/abs/1806.02473v3).
- [189] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. In: Advances in Neural Information Processing Systems 30 (NIPS 2017). Long Beach, CA, USA: Curran Associates, Inc.: Red Hook, NY, 2018, pp. 6530–6539.
- [190] M. Zitnik, M. Agrawal, and J. Leskovec. Modeling polypharmacy side effects with graph convolutional networks. 2018. eprint: [arXiv:1802.00543v2](https://arxiv.org/abs/1802.00543v2).
- [191] N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecule synthesis. 2018. eprint: [arXiv:1805.11973v1](https://arxiv.org/abs/1805.11973v1).
- [192] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. In: Advances in Neural Information Processing Systems 27 (NIPS 2014). Montreal, Canada: Curran Associates, Inc.: Red Hook, NY, 2015, pp. 2672–2680.
- [193] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A.A. Bharath. In: IEEE Signal Processing Magazine 35(1) (2018), pp. 53–65. DOI: [10.1109/MSP.2017.2765202](https://doi.org/10.1109/MSP.2017.2765202).

- [194] P.B. Jørgensen, M.N. Schmidt, and O. Winther. In: Mol. Inf. 37 (2018), p. 1700133. DOI: 10.1002/minf.201700133.
- [195] R. Gómez-Bombarelli, J.N. Wei, D. Duvenaud, J.M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T.D. Hirzel, R.P. Adams, and A. Aspuru-Guzik. In: ACS Cent. Sci. 4(2) (2018), pp. 268–276. DOI: 10.1021/acscentsci.7b00572.
- [196] A. Kadurin, S. Nikolenko, K. Khrabrov, A. Aliper, and A. Zhavoronkov. In: Mol. Pharmaceutics 14 (2017), pp. 3098–3104. DOI: 10.1021/acs.molpharmaceut.7b00346.
- [197] E. Putin, A. Asadulaev, Q. Vanhaelen, Y. Ivanenkov, A.V. Aladin-skaya, A. Aliper, and A. Zhavoronkov. In: Mol. Pharmaceutics 15 (2018), pp. 4386–4397. DOI: 10.1021/acs.molpharmaceut.7b01137.
- [198] T. Blaschke, M. Olivecrona, O. Engkvist, J. Bajorath, and H. Chen. In: Mol. Inf. 37 (2018), p. 1700123. DOI: 10.1002/minf.201700123.
- [199] E.J. Bjerrum and B. Sattarov. In: Biomolecules 8 (2018), p. 131. DOI: 10.3390/biom8040131.
- [200] G. Guimaraes, B. Sanchez-Lengeling, C. Out-eiral, P.L.C. Farias, and A. Aspuru-Guzik. Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence 2018. eprint: arXiv:1705.10843v3.
- [201] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song. Syntax-Directed Variational Autoencoder for Structured Data. 2018. eprint: arXiv:1802.08786v1.
- [202] Shailima Rampogu, Amir Zeb, Ayoung Baek, Chanin Park, Minky Son, and Keun Woo Lee. “Discovery of Potential Plant-Derived Peptide Deformylase (PDF) Inhibitors for Multidrug-Resistant Bacteria Using Computational Studies”. In: Journal of Clinical Medicine 7.12 (2018). DOI: 10.3390/jcm7120563.
- [203] M. Gilson, T. Liu, M. Baitaluk, G. Nicola, L. Hwang, and J. Chong. “BindingDB in 2015: A public database for medicinal chemistry, computational chemistry and systems pharmacology”. In: Nucleic Acids Research 44 (2016), pp. D1045–D1053.

- [204] M.M. Mysinger, M. Carchia, J.J. Irwin, and B. Shoichet. “Directory of Useful Decoys, Enhanced (DUD-E): Better Ligands and Decoys for Better Benchmarking”. In: Journal of Medicinal Chemistry 55 (2012), pp. 6582–6594.
- [205] RDKit: Open-source cheminformatics software. URL: <https://www.rdkit.org>.
- [206] Jianing Lu, Song Xia, Jieyu Lu, and Yingkai Zhang. “Dataset Construction to Explore Chemical Space with 3D Geometry and Deep Learning”. In: J. Chem. Inf. Model. 61 (2021).
- [207] Mark van der Laan, Eric Polley, and Alan Hubbard. “Super Learner”. In: Journal of the American Statistical Applications in Genetics and Molecular Bi 6.25 (2007), pp. 273–297. DOI: 10.2202/1544-6115.1309.
- [208] Leo Breiman. “Stacked Regressions”. In: Machine Learning 24 (1996), pp. 49–64.
- [209] David Wolpert. “Stacked Generalization”. In: Neural Networks 5 (1992).
- [210] C. Cortes and V. Vapnik. “Support-vector networks”. In: Machine learning 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.
- [211] L. Breiman. “Random Forests”. In: Machine Learning 45.1 (2001), pp. 5–32. ISSN: 0885–6125. DOI: 10.1023/A:1010933404324.
- [212] A.V. Dorogush, V. Ershov, and A. Gulin. CatBoost: gradient boosting with categorical features support. 2018. eprint: [arXiv:1810.11363](https://arxiv.org/abs/1810.11363).
- [213] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. “The Graph Neural Network Model”. In: IEEE Transactions on Neural Networks 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.
- [214] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Underst 2018. eprint: [arXiv:1810.04805](https://arxiv.org/abs/1810.04805).
- [215] D. Rogers and M. Hahn. “Extended-Connectivity Fingerprints”. In: Journal of Chemical Information and Modeling 50.5 (2010), pp. 742–754. DOI: 10.1021/ci100050t.

- [216] D. Beck, G. Haffari, and T. Cohn. “Graph-to-Sequence Learning using Gated Graph Neural Networks”. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 273–283. DOI: 10.18653/v1/P18-1026.
- [217] V. Chupakhin, G. Marcou, I. Baskin, A. Varnek, and D. Rognan. “Predicting Ligand Binding Modes from Neural Networks Trained on Protein-Ligand Interaction Fingerprints”. In: Journal of Chemical Information and Modeling 53.4 (2013), pp. 763–772. DOI: 10.1021/ci300200r.
- [218] A. Arora, A. Candel, J. Lanford, E. LeDell, and V. Parmar. Deep Learning with H2O. 6th ed. 2022. URL: <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf>.
- [219] C. Click, J. Lanford, M. Malohlava, V. Parmar, and H. Roark. Gradient Boosted Models with H2O. 7th ed. 2022. URL: <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/GBMBooklet.pdf>.
- [220] P. McCullagh and J.A. Nelder. “Generalized Linear Models”. In: (1989).
- [221] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: Journal of Machine Learning Research 12 (2011), pp. 2825–2830.
- [222] John C. Platt. “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods”. In: ADVANCES IN LARGE MARGIN CLASSIFIERS (1999).
- [223] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. 2017. eprint: [arXiv:1710.10903](https://arxiv.org/abs/1710.10903).
- [224] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, and M. Zheng. “Pushing the Boundaries of Molecular Representation for Drug Discovery with the Graph Attention Mecha-

- nism”. In: Journal of Medicinal Chemistry 63.16 (2020), pp. 8749–8760. DOI: 10.1021/acs.jmedchem.9b00959.
- [225] DGL-LifeSci: Bringing graph neural networks to chemistry and biology. URL: <https://github.com/aws-labs/dgl-lifesci>.
- [226] T.-Y. Lin, P. Goyal, R.B. Girshick, K. He, and P. Dollár. “Focal Loss for Dense Object Detection”. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 42 (2020), pp. 318–327.
- [227] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A.M. Rush. HuggingFace’s Transformers: State-of-the-art Natural Language Processing. 2019. eprint: arXiv:1910.03771.
- [228] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. Shoemaker, P.A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. Bolton. “PubChem 2019 update: improved access to chemical data”. In: Nucleic Acids Research 47 (2019), pp. D1102–D1109.
- [229] D. Kingma and J. Ba. Adam: A method for stochastic optimization. 2017. eprint: arXiv:1412.6980.
- [230] Я. М. Матвійчук. Математичне макромодельовання динамічних систем: Теорія та застосування. Львів, 2000.
- [231] Я. М. Матвійчук and О. І. Карчевська. “Принцип редукції математичних моделей”. In: Обчислювальні методи і системи перетворення інформації. Вип.3: Зб. праць. Львів, Україна, 2014, pp. 80–83.
- [232] Yaroslav Matviychuk and Olga Karchevska. “Increasing the Correctness of Mathematical Models by Novel Reduction Principle”. In: CPEE’2015 Proceeding. Lviv, Ukraine, 2015, pp. 100–112. ISBN: 9786176078036.

- [233] Y. Matviychuk. “Improvement of Mathematical Models by Applying the Reduction Principle”. In: Information Technologies and Computer Modeling (2019), pp. 284–288.
- [234] Enamine Ltd. A global provider of screening compounds, building blocks, and
URL: <https://enamine.net/>.
- [235] Philippe Schwaller, Alain C. Vaucher, Teodoro Laino, and Jean-Louis Reymond. Data augmentation strategies to improve reaction yield predictions and estima
Nov. 2020. DOI: 10.26434/chemrxiv.13286741. eprint: ChemRxiv: 10.26434/chemrxiv.13286741.v1.
- [236] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. “Focal Loss for Dense Object Detection”. In: 2017 IEEE International Conference on Computer Vision (ICCV). 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.
- [237] Artem Mavrin. Focal Loss. 2021. eprint: <https://github.com/artemmavrin/focal-loss>.
- [238] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: Machine learning 20.3 (1995), pp. 273–297.
- [239] Hakime Ozturk, Elif Ozkirimli, and Arzucan Özgür. “A novel methodology on distributed representations of proteins using their interacting ligands”. In: Bioinformatics 34.13 (2018), pp. 295–303. DOI: 10.1093/bioinformatics/bty287.
- [240] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: Journal of Machine Learning Research 12 (2011), pp. 2825–2830.
- [241] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality 2013. eprint: arXiv:1310.4546v1.
- [242] ChEMBL. URL: <https://www.ebi.ac.uk/chembl/>.

- [243] Anna Gaulton, Anne Hersey, Michał Nowotka, A Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa Bellis, Elena Cibrián-Uhalte, Mark Davies, Nathan Dedman, Anneli Karlsson, María Magariños, John Overington, George Papadatos, Ines Smit, and Andrew Leach. “The ChEMBL database in 2017”. In: Nucleic acids research 45 (Nov. 2016). DOI: 10.1093/nar/gkw1074.
- [244] PubChem. URL: <https://pubchem.ncbi.nlm.nih.gov/>.
- [245] Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A Shoemaker, Paul A Thiessen, Bo Yu, Leonid Zaslavsky, Jian Zhang, and Evan E Bolton. “PubChem in 2021: new data content and improved web interfaces”. In: Nucleic Acids Research 49.D1 (Nov. 2020), pp. D1388–D1395. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa971.
- [246] J. Bergstra, Daniel Yamins, and D. Cox. “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures”. In: ICML’13: Proceedings of the 30th International Conference on International C Vol. 28. 2013, pp. 115–123.
- [247] RXNFP - chemical reaction fingerprints. Software. eprint: <https://github.com/rxn4chemistry/rxnfp>.
- [248] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. 2017. eprint: [arXiv:1704.01212](https://arxiv.org/abs/1704.01212).
- [249] Kyle Swanson. Message Passing Neural Networks for Molecular Property Prediction. June 2019.
- [250] Kevin Yang. Are Learned Molecular Representations Ready for Prime Time? June 2019.
- [251] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Tim Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs F. Jensen, and Regina Barzilay. “Analyzing Learned Molecular Representations for Property Prediction”. In:

- Journal of Chemical Information and Modeling* 59.8 (July 2019), pp. 3370–3388. DOI: 10.1021/acs.jcim.9b00237.
- [252] Mikołaj Kowalik, Chris M. Gothard, Aaron M. Drews, Nosheen A. Gothard, Alex Weckiewicz, Patrick E. Fuller, Bartosz A. Grzybowski, and Kyle J. M. Bishop. “Parallel Optimization of Synthetic Pathways within the Network of Organic Chemistry”. In: *Angewandte Chemie* 51.32 (Aug. 2012), pp. 7982–7932. DOI: 10.1002/anie.201202209.
- [253] Claudia Houben and Alexei Lapkin. “Automatic discovery and optimization of chemical processes”. In: *Current Opinion in Chemical Engineering* 9 (Aug. 2015), pp. 1–7. DOI: 10.1016/j.coche.2015.07.001.
- [254] Bartosz Zielinski, Anna Plichta, Krzysztof Misztal, Przemyslaw Spurek, Monika Brzychczy-Wloch, and Ochonska Dorota. “Deep learning approach to bacterial colony classification”. In: *PLoS ONE* 12.9 (2017). DOI: 10.1371/journal.pone.0184554.
- [255] Yasunari Matsuzaka and Yoshihiro Uesawa. “Optimization of a Deep-Learning Method Based on the Classification of Images Generated by Parameterized Deep Snap a Novel Molecular-Image-Input Technique for Quantitative Structure–Activity Relationship (QSAR) Analysis”. In: *Frontiers in Bioengineering and Biotechnology* (2019). DOI: 10.3389/fbioe.2019.00065.
- [256] P.S. Gromski, A.B. Henson, and J.M. et al. Granda. “Automatic discovery and optimization of chemical processes”. In: *Nature Reviews Chemistry* 3 (Jan. 2019), pp. 119–128.
- [257] *Indigo Toolkit. Software.* eprint: <https://lifescience.opensource.epam.com/indigo/index.html>.
- [258] *DescriptaStorus. Software.* eprint: <https://github.com/bp-kelley/descriptastorus>.
- [259] Andrey Bogolyubsky, Olena Savych, Anton Zhemera, Sergey Pipko, Alexander Grishchenko, Anzhelika Konovets, Roman Doroshchuk, Dmytro Khomenko, Volodymyr Brovarets, Yurii Moroz, and Mykhailo Vybornyi. “A facile one-pot parallel synthesis of 3-amino-1,2,4 - Tri-

- zoles”. In: ACS Combinatorial Science 20 (June 2018). DOI: 10.1021/acscombsci.8b00060.
- [260] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: Journal of Machine Learning Research 9 (2008), pp. 2579–2605.
- [261] J. Huuskonen. In: J. Chem. Inf. Comput. Sci. 40 (2000), pp. 773–777. DOI: 10.1021/ci9901338.
- [262] T. Hou, K. Xia, W. Zhang, and X. Xu. In: J. Chem. Inf. Comput. Sci. 44 (2004), pp. 266–275. DOI: 10.1021/ci034184n.
- [263] J.S. Delaney. In: J. Chem. Inf. Comput. Sci. 44 (2004), pp. 1000–1005. DOI: 10.1021/ci034243x.
- [264] DLS-100 Solubility Dataset. DOI: 10.17630/3a3a5abc-8458-4924-8e6c-b804347605e8.
- [265] A. Llinàs, R.C. Glen, and J.M. Goodman. In: J. Chem. Inf. Model. 48 (2008), pp. 1289–1303. DOI: 10.1021/ci800058v.
- [266] A.J. Hopfinger, E.X. Esposito, A. Llinàs, R.C. Glen, and J.M. Goodman. In: J. Chem. Inf. Model. 49 (2009), pp. 1–5. DOI: 10.1021/ci800436c.
- [267] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. In: Proceedings of the 25th International Conference on Machine Learning. Helsinki, Finland: Omnipress: Madison, WI, USA, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.
- [268] G.E. Hinton and R.S. Zemel. In: Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS 1993). Denver, CO, USA: Morgan Kaufmann Publishers Inc.: San Francisco, CA, 1994, pp. 3–10.
- [269] D.P. Kingma and M. Welling. Auto-Encoding Variational Bayes. 2014. eprint: arXiv:1312.6114v10.
- [270] I. Sutskever, O. Vinyals, and Q.V. Le. In: Advances in Neural Information Processing Systems 27 (NIPS 2014). Montreal, Canada: Curran Associates, Inc.: Red Hook, NY, 2015, pp. 3104–3112.

- [271] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. 2014. eprint: [arXiv:1409.0473](https://arxiv.org/abs/1409.0473).
- [272] S. Hochreiter and J. Schmidhuber. In: Neural Comput. 9(8) (1997), pp. 1735–1780. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735).
- [273] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean. In: Advances in Neural Information Processing Systems 26 (NIPS 2013). Lake Tahoe, NV, USA: Curran Associates, Inc.: Red Hook, NY, 2014, pp. 3111–3119.
- [274] A. Lamb, A. Goyal, S. Zhang, A.C. Courville, and Y. Bengio. In: Advances in Neural Information Processing Systems 29 (NIPS 2016). Lake Tahoe, NV, USA: Curran Associates, Inc.: Red Hook, NY, 2017, pp. 4601–4609.
- [275] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. In: J. Artif. Int. Res. 16 (2002), pp. 321–357. DOI: [10.1613/jair.953](https://doi.org/10.1613/jair.953).
- [276] G. Lemaître, F. Nogueira, and C.K. Aridas. In: J. Mach. Learn. Res. 18 (2017), pp. 1–5.
- [277] H.J.C. Berendsen, J.R. Grigera, and T.P. Straatsma. In: J. Phys. Chem. 91 (1987), pp. 6269–6271. DOI: [10.1021/j100308a038](https://doi.org/10.1021/j100308a038).
- [278] W.L. Jorgensen, D.S. Maxwell, and J. Tirado-Rives. In: J. Am. Chem. Soc. 118 (1996), pp. 11225–11236. DOI: [10.1021/ja9621760](https://doi.org/10.1021/ja9621760).
- [279] W. Smith, T. Forester, and I. Todorov. The DL POLY Classic User Manual. 2010. URL: http://www.ccp5.ac.uk/DL%5C_POLY%5C_C/.
- [280] S. Melchionna, G. Ciccotti, and B.L. Holian. In: Molec. Phys. 78 (1993), pp. 533–544. DOI: [10.1080/00268979300100371](https://doi.org/10.1080/00268979300100371).
- [281] P. Ertl. “An algorithm to identify functional groups in organic molecules”. In: Journal of Cheminformatics 9 (2017), p. 36. DOI: [10.1186/s13321-017-0225-z](https://doi.org/10.1186/s13321-017-0225-z).

- [282] SMARTS Theory Manual, Daylight Chemical Information Systems.
URL: <https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html>.
- [283] A. Wildman and G.M. Crippen. In: J. Chem. Inf. Comput. Sci. 395 (1999), pp. 868–873. DOI: 10.1021/ci9903071.
- [284] P. Ertl and A. Schuffenhauer. In: J. Cheminformatics 1 (2009), p. 8. DOI: 10.1186/1758-2946-1-8.
- [285] L. Li, T. Totton, and D. Frenkel. In: J. Chem. Phys. 146 (2017), p. 214110. DOI: 10.1063/1.4983754.
- [286] R. Gillet, A. Fierro, L.M. Valenzuela, and J.R. Pérez-Correa. In: Fluid Phase Equilib. 472 (2018), pp. 85–93. DOI: 10.1016/j.fluid.2018.05.013.
- [287] G. Duarte Ramos Matos and D.L. Mobley. In: F1000Research 7 (2019), p. 686. DOI: 10.12688/f1000research.14960.2.
- [288] Mufei Li, Jinjing Zhou, Jiajing Hu, Wenxuan Fan, Yangkang Zhang, Yaxin Gu, and George Karypis. “DGL-LifeSci: An Open-Source Toolkit for Deep Learning on Graphs in Life Science”. In: ACS Omega (2021).
- [289] ChEMSPACE. The largest online catalog of small molecules and biologics in the
URL: <https://chem-space.com/>.
- [290] Liquibase. URL: <https://www.liquibase.org/>.
- [291] pgTap. URL: <https://pgtap.org/>.
- [292] Great Expectations. URL: <https://greatexpectations.io/>.

I Додаток А. АКТИ ВПРОВАДЖЕННЯ



АКТ використання наукових результатів дисертаційної роботи Гурбича Олександра Вікторовича представленої на здобуття наукового ступеня доктора філософії

Комісія в складі: голови комісії - начальника науково-дослідної частини д.т.н., с.н.с. Небесного Р.В. та членів комісії - завідувача кафедри СШ Шаховської Н.Б., професора кафедри СШ Яковини В.С., доцента кафедри Хавалка В.М., доцента кафедри СШ Кривенчука Ю.П. цим актом підтверджують, що результати дисертаційної роботи Гурбича О.В., зокрема:

- метод прогнозу молекулярної спорідненості;
- метод передбачення відсотку виходу продукту хімічної реакції;
- метод генерації молекулярних структур із заданими властивостями;
- архітектура платформи для розробки нових лікарських речовин;

використано у науково-дослідних роботах фінансованих Міністерством освіти і науки України, що виконувались на кафедрі систем штучного інтелекту і включено до звіту “Інформаційна технологія формування психофізичного портету в умовах стресових ситуацій” (№ держ. реєстру 0119U002257).

Отримані автором результати використано:

- при розробленні систем прогнозу молекулярної спорідненості;
- при розробленні засобів передбачення відсотку виходу продукту хімічної реакції;
- при розробленні засобів генерації молекулярних структур із заданими властивостями;
- при розробленні систем для розробки нових лікарських речовин.

Голова комісії:

начальник науково-дослідної частини
(д.т.н., с.н.с.)

(підпис)

Р.В. Небесний
(прізвище та ініціали)

Члени комісії:

завідувач кафедри СШ

(підпис)

Н.Б. Шаховська
(прізвище та ініціали)

доцент кафедри СШ

(підпис)

В.С. Яковини
(прізвище та ініціали)

доцент кафедри СШ

(підпис)

В.М. Хавалко
(прізвище та ініціали)

доцент кафедри СШ

(підпис)

Ю.П. Кривенчук
(прізвище та ініціали)

“ЗАТВЕРДЖУЮ”

Проректор з науково-педагогічної роботи
Національного університету
“Львівська політехніка”

О.Р.Давидчак

2023 р.



АКТ

про впровадження в навчальний процес результатів
дисертаційної роботи

Гурбича Олександра Вікторовича

Цей акт складено про те, що результати дисертаційної роботи Гурбича Олександра Вікторовича впроваджено у навчальний процес кафедри “Системи штучного інтелекту” Національного університету “Львівська політехніка”.

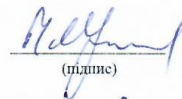
Впровадження результатів дисертаційної роботи полягає у їхньому використанні при викладанні навчальних дисциплін як окремих розділів лекційних курсів, так і в циклах лабораторних робіт.

Зокрема, для викладання дисципліни “Нейромеревеві технології та їх застосування” для студентів кваліфікаційного рівня “бакалавр”, що навчаються за напрямком 122 “Комп’ютерні науки”, використано такі результати:

- метод прогнозу молекулярної спорідненості;
- метод передбачення відсотку виходу продукту хімічної реакції;
- метод генерації молекулярних структур із заданими властивостями;
- архітектура платформи для розробки нових лікарських речовин.

Директор ІКНІ

(д.т.н., проф.)


(підпис)

М.О.Медиковський

(прізвище та ініціали)

завідувач кафедри СШП

(д.т.н., проф.)


(підпис)

Н.Б.Шаховська

(прізвище та ініціали)

доцент кафедри СШП

(к.е.н., проф.)


(підпис)

Н.І.Бойко

(прізвище та ініціали)

ЗАТВЕРДЖУЮ

Директор ТОВ «СофтСерв-Індустрія»

_____ Кулик Орест

м. Львів

«___» _____ 2023 р.

АКТ
про впровадження результатів дисертаційної роботи
аспіранта кафедри “Системи штучного інтелекту”
Національного університету “Львівська політехніка”
Гурбича Олександра Вікторовича

Цей Акт підтверджує, що результати дисертаційної роботи Гурбича О.В. були використані для створення комп'ютерних систем ранньої розробки лікарських речовин у ТОВ «СофтСерв-Індустрія» м. Львів, у 2020-2021рр., а саме:

- Прогноз молекулярної спорідненості.
- Передбачення виходу продукту хімічної реакції.

Даний Акт не є підставою для взаємних фінансових розрахунків.

Директор
ТОВ «СОФТСЕРВ-ІНДУСТРІЯ»



КУЛИК ОРЕСТ



ANNEX 2

№ SOW-07-06-22 dated 07.06.2022

**Statement of Work for
Software Development Services**

to the Agreement № 010322-01HTG dated 01.03.2022

London, UK

Date: 07.06.2022

Prepared	for	Prepared	by:
HTG Molecular Diagnostics, Inc., having its registered address at 3430 E. Global Loop, Tucson, AZ 85706 USA		Blackthorn AI, Ltd., having its registered address at Kemp House 160 City Road, London, United Kingdom, EC1V 2NX	

WARNING: The enclosed material is confidential and is owned by blackthorn.ai. This material is presented for the purposes of the services evaluation and may not be disclosed to anyone other than the addressee or authorized representatives of Customer named here within. The material is also prohibited for use in software development by anyone other than BLACKTHORN AI LTD.

1. Statement of Work

This Statement of Work (“SoW”) is entered by and between **BLACKTHORN AI LTD** (referred to as the “Contractor”) and **HTG Molecular Diagnostics, Inc.** (referred to as “Customer”) as of the date above (the “Effective Date”) for the purposes of the Contractor providing Software Development Services (“Services”) to the Customer.

2. Project Overview

2.1 Executive summary

1. **HTG Molecular Diagnostics, Inc.**, having its registered address at **3430 East Global Loop, Tucson, Arizona**, represented its Senior Vice President, Stephen Barat
2. **BLACKTHORN AI LTD** is a limited liability company organized and existing under the laws of England and Wales, under company number **13335565**, having its registered address at **Kemp House, 160 City Road, London, EC1V 2NX, United Kingdom**, represented by its Director **Oleksandr Gurbych**

2.2 Customer Responsibilities

1. The Customer will provide access to requested personnel (directors, employees, and officers) and contractors, information, documentation, and systems requested by the Contractor via means of communication agreed by the Parties that are required for the provision of the Services according to the Agreement.
2. The Customer will respond to the Contractor’s inquiries within two business days. In the event of no response, the Contractor will resubmit such inquiry. If there is no response to the second inquiry, the Contractor may proceed with the best information available or adjust the schedule of the Services’ provision.
3. The Customer will review the deliverables and check for compliance with the Customer’s business goals and requirements within five business days.
4. The Customer provides required data as requested, so the respective models can be trained and validated.
5. The data provided by the Customer is sufficient to calculate baseline metrics.
6. The manual labeling of images and tabular data will be carried out by Customer specialists.

2.3 Support and Maintenance

Subject to the new arrangements in the future, the Contractor shall perform the technical support, maintenance, and further development of the **HTG Molecular Diagnostics Inc.** in an efficient, competent, and timely manner and reasonable care, skill, and diligence in the performance of that service. The fees and the terms of the technical support, maintenance, and further development of the **HTG Molecular Diagnostics, Inc.** services will be agreed upon and described in a separate agreement, statement of work or communication between the Parties, which clearly provides significant conditions of arrangements (including, without limitation, via

WARNING: The enclosed material is confidential and is owned by blackthorn.ai. This material is presented for the purposes of the evaluation of the services and may not be disclosed to anyone other than the addressee or authorized representatives of the Customer named here within. The material is also prohibited for use in software development by anyone other than BLACKTHORN AI LTD.

3. Epics

#	Description
E-1	Predict compound activity toward molecular targets using transcriptomic data generated from cell lines.
E-2	Generation of hit-like molecules from transcriptomics data via reinforcement learning.
E-3	Generation of hit-like molecules from transcriptomics data via GANs.
E-4	Ranking of RASL-seq compound hits.
E-5	Screening therapeutics targeting patient groups using gene expression features.

4. Implementation Roadmap

4.1 Predict compound activity using GES

#	Epic	Task	Min	Max
1	E-1	Literature review		
2	E-1	Collect and preprocess activity data (PubChem BioAssay database). Select sufficiently represented targets (must have more than 50 known active compounds)		
3	E-1	Feature engineering (Morgan fingerprints, HTSFPs)		
4	E-1	Visualization of chemical (compounds) and biological (GES) spaces (t-SNE)		
5	E-1	Development of ML models for biological activity prediction. One target - one model (Note: 990 models reported in the paper). Selectivity of a compound to a target is a result of the voting of the models.		
6	E-1	Feature importance analysis, dropping low-importance features. Retraining the models (all 990).		
7	E-1	Hyperparameters tuning		
8	E-1	Cross-validation of the models		
9	E-1	Selecting the best models		
10	E-1	Meta-learning experiments to get the best possible results		
11	E-1	Clean code preparation, documenting the code, knowledge sharing, presenting the results		
Total, business days per FTE				

NOTE: the time estimates are in days per FTE. The project total duration does not add up to the sum of FTE days as some tasks are done in parallel. Also, all calculated days are labor days, i.e., 5 FTE days are equal to 7 calendar days (with Saturday and Sunday included)

WARNING: The enclosed material is confidential and is owned by blackthorn.ai. This material is presented for the purposes of the evaluation of the services and may not be disclosed to anyone other than the addressee or authorized representatives of the Customer named here within. The material is also prohibited for use in software development by anyone other than BLACKTHORN AI LTD.

4.2 Generation of hit-like molecules from transcriptomics data via reinforcement learning

#	Epic	Task	Min	Max
1	E-2	Literature review		
2	E-2	Collect and preprocess cancer genome profiles, and small drug-like molecules biological activity data (TCGA, ChEMBL, GDSC, Tox21, SIDER databases).		
3	E-2	Feature engineering (Morgan fingerprints, HTSFPs)		
4	E-2	Visualization of chemical (compounds) and biological (GES) spaces (t-SNE)		
5	E-2	Pretraining the disease context (gene expression or profile) generator VAE (PVAE)		
6	E-2	Pretraining the therapeutic molecules' SMILES generator VAE (SVAE)		
7	E-2	Critic (PaccMann) design and implementation as the reward function. The critique consists of the following submodules: - <i>IC50 predictor</i> design and implementation (on ChEMBL) - <i>environmental toxicity predictor</i> design and implementation (on Tox21). Benchmark: ROC-AUC 0.877 - <i>adverse effects of the drug</i> (on SIDER) design and implementation. Benchmark: ROC-AUC 0.835		
8	E-2	Joint training of PVAE and SVAE with PaccMann as the reward function to generate novel compounds with cellular IC50 as the target treatment response to the generated compound.		
9	E-2	Conditioning of the compound-generator on gene expression profiles of cancer subtypes (on GDSC): breast (carcinoma), lung (carcinoma), prostate (carcinoma), autonomic ganglia (neuroblastoma)		
10	E-2	Assessment of structural similarity of the generated drug candidates to known anticancer molecules. Making visuals.		
11	E-2	Analysis of physicochemical properties of the generated drug candidates. Making visuals.		
12	E-2	Clean code preparation, documenting the code, knowledge sharing, presenting the results		
Total, business days per FTE				

NOTE: the time estimates are in days per FTE. The project total duration does not add up to the sum of FTE days as some tasks are done in parallel. Also, all calculated days are labor days, i.e., 5 FTE days are equal to 7 calendar days (with Saturday and Sunday included)

WARNING: The enclosed material is confidential and is owned by blackthorn.ai. This material is presented for the purposes of the evaluation of the services and may not be disclosed to anyone other than the addressee or authorized representatives of the Customer named here within. The material is also prohibited for use in software development by anyone other than BLACKTHORN AI LTD.

4.3 Generation of hit-like molecules from transcriptomics data via GANs

#	Epic	Task	Min	Max
1	E-3	Literature review		
2	E-3	Collect and preprocess gene expression profiles, and small drug-like molecules biological activity data (TCGA, ChEMBL, GDSC, Tox21, SIDER databases).		
3	E-3	Feature engineering (Morgan fingerprints, HTSFPs)		
4	E-3	Visualization of chemical (compounds) and biological (GES) spaces (t-SNE)		
5	E-3	Design and development of the SMILES encoder head		
6	E-3	Design and development of the SMILES decoder head		
7	E-3	Design and development of the latent space autoencoder		
8	E-3	Design and development of the gene expression encoder head		
9	E-3	Design and development of the molecular property discriminator		
10	E-3	Design and development of the drug-like molecule recognizer head		
11	E-3	Training of the composite hit-like molecules GAN		
12	E-3	Refinement of the composite hit-like molecules GAN		
13	E-3	Clean code preparation, documenting the code, knowledge sharing, presenting the results		
Total, business days per FTE				

NOTE: the time estimates are in days per FTE. The project total duration does not add up to the sum of FTE days as some tasks are done in parallel. Also, all calculated days are labor days, i.e., 5 FTE days are equal to 7 calendar days (with Saturday and Sunday included)

4.4 Ranking of RASL-seq compound hits

#	Epic	Task	Min	Max
1	E-4	Literature review		
2	E-4	Get the data - The RASL-seq dataset is available as Supplementary Data. The mRNA-seq dataset is available at Gene Expression Omnibus, GSE143144.		
3	E-4	Exploratory data analysis		

WARNING: The enclosed material is confidential and is owned by blackthorn.ai. This material is presented for the purposes of the evaluation of the services and may not be disclosed to anyone other than the addressee or authorized representatives of the Customer named here within. The material is also prohibited for use in software development by anyone other than BLACKTHORN AI LTD.

4	E-4	Compound ranking, R56 to calculate Euclidean distance in 11 dimensions (corresponding to each gene in the profile)			
5	E-4	Verification of the results			
6	E-4	Clean code preparation, documenting the code, knowledge sharing, presenting the results			
Total, business days per FTE					

NOTE: the time estimates are in days per FTE. The project total duration does not add up to the sum of FTE days as some tasks are done in parallel. Also, all calculated days are labor days, i.e. 5 FTE days are equal to 7 calendar days (with Saturday and Sunday included)

4.5 Screening therapeutics targeting patient groups using gene expression features

#	Epic	Task	Min	Max
1	E-5	Literature review		
2	E-5	Data preprocessing (OCTAD - http://octad.org), other public data sources (see the list above)		
3	E-5	Exploratory data analysis		
4	E-5	Reference tissue selection		
5	E-5	Disease signature creation		
6	E-5	Reversal of cancer expression		
7	E-5	Hit prediction and selection		
8	E-5	Verification of the results		
9	E-5	Clean code preparation, documenting the code, knowledge sharing, presenting the results		
Total, business days per FTE				

NOTE: the time estimates are in days per FTE. The project total duration does not add up to the sum of FTE days as some tasks are done in parallel. Also, all calculated days are labor days, i.e. 5 FTE days are equal to 7 calendar days (with Saturday and Sunday included)

4.6 Team Structure

Role	Responsibilities	Count	Involvement
Scientific Supervisor / Sr. Solution Architect	<ul style="list-style-type: none"> - Business requirements collection and analysis - System analysis and design - System requirements specification - Team technical leadership - Team scientific supervision - Code review 	1	0.2

WARNING: The enclosed material is confidential and is owned by blackthorn.ai. This material is presented for the purposes of the evaluation of the services and may not be disclosed to anyone other than the addressee or authorized representatives of the Customer named here within. The material is also prohibited for use in software development by anyone other than BLACKTHORN AI LTD.

6. Details and Signatures of the Parties

CONTRACTOR:
BLACKTHORN AI LTD

Company number: 13335565

E-mail: alex@blackthorn.ai,
gurbycholeksandr@gmail.com

Address: Kemp House, 160 City Road,
London, EC1V 2NX, UNITED KINGDOM

Name: Oleksandr Gurbych

Title: Director

Date: 6/14/2022

(signature)

DocuSigned by:

618A0E769B974BE...

CUSTOMER:
HTG Molecular Diagnostics, Inc.

Company number: n/a

E-mail: sbarat@htgmolecular.com

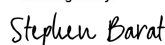
Address: 3430 E. Global Loop, Tucson,
AZ 85706 USA

Name: Stephen Barat

Title: Senior Vice President

Date: 6/14/2022

(signature)

DocuSigned by:

7973A61B124E492...

Certificate Of Completion

Envelope Id: D4C56DD1C55B44DEBBAD5FBEEB58613D	Status: Completed
Subject: Please Sign: HTG Molecular - SoW V1.2, Final Approved, 14 Jun 2022.pdf	
Source Envelope:	
Document Pages: 15	Signatures: 2
Certificate Pages: 2	Initials: 0
AutoNav: Enabled	Envelope Originator:
Enveloped Stamping: Enabled	Oleksandr Hurbych
Time Zone: (UTC-08:00) Pacific Time (US & Canada)	KEMP HOUSE, 160 CITY ROAD
	nil
	London, London, City of EC1V 2NX
	alex@blackthorn.ai
	IP Address: 78.94.180.210


Record Tracking

Status: Original	Holder: Oleksandr Hurbych	Location: DocuSign
6/14/2022 1:03:40 PM	alex@blackthorn.ai	

Signer Events

Oleksandr Hurbych
alex@blackthorn.ai
Director
BLACKTHORN AI LTD
Security Level: Email, Account Authentication (None)

Signature

DocuSigned by:

618A0E769B974BE...
Signature Adoption: Pre-selected Style
Signed by link sent to alex@blackthorn.ai
Using IP Address: 78.94.180.210

Timestamp

Sent: 6/14/2022 1:17:06 PM
Viewed: 6/14/2022 1:17:26 PM
Signed: 6/14/2022 1:17:35 PM

Electronic Record and Signature Disclosure:
Not Offered via DocuSign

Stephen Barat
sbarat@htgmolecular.com
Security Level: Email, Account Authentication (None)

DocuSigned by:

7973A81B124E492...
Signature Adoption: Pre-selected Style
Signed by link sent to sbarat@htgmolecular.com
Using IP Address: 164.39.85.124

Sent: 6/14/2022 1:17:06 PM
Viewed: 6/14/2022 1:19:29 PM
Signed: 6/14/2022 1:20:27 PM

Electronic Record and Signature Disclosure:
Not Offered via DocuSign

In Person Signer Events	Signature	Timestamp
Editor Delivery Events	Status	Timestamp
Agent Delivery Events	Status	Timestamp
Intermediary Delivery Events	Status	Timestamp
Certified Delivery Events	Status	Timestamp
Carbon Copy Events	Status	Timestamp
Witness Events	Signature	Timestamp
Notary Events	Signature	Timestamp
Envelope Summary Events	Status	Timestamps

II Додаток Б. ПРОГРАМНИЙ КОД МЕТОДУ ПРОГНОЗУВАННЯ МОЛЕКУЛЯРНОЇ СПОРІДНЕНОСТІ

i Підготовка даних

```
from affinity_module.protein_graph_loaders import ProteinGraphMaker
from dgllife.utils import smiles_to_biggraph
from dgl.data.utils import save_graphs, load_graphs
from rdkit import Chem
import os
import dgl.backend as F
import numpy as np
import pandas as pd
import pickle
from torch.utils.data import DataLoader
import json

class VPLGDataset(object):
    def __init__(self,
                 smiles_to_graph=smiles_to_biggraph,
                 smiles_node_featurizer=None,
                 smiles_edge_featurizer=None,
                 pdb_id_col_name="pdbId",
                 smiles_col_name="ligandSMILES",
                 target_col_name="logKx",
                 foldId_col_name=None, # if not None, will read and save
                                     # foldId
                 master_data_table = "",
                 pdb2graph_translator = None, # must be pre-created class
                                     # derived from ProteinGraphMaker
                 log_every=500,
                 cache_size=500,
                 load=False, # whether to load graphs from cache
                 inference=False,
                 append_itemId = False, # whether to return item id in
                                     # __getitem__
                 columns_to_ignore = []
    ):

        assert isinstance(pdb2graph_translator, ProteinGraphMaker)
        self.pdb2graph_translator = pdb2graph_translator

        # Note that for now, we still re-read .csv file even if load is
        # set to True; this is needed at
        # least to be able to handle fold_ids
        self.df = pd.read_csv(master_data_table)
        self.smiles = []
        self.smiles_graphs = []
        self.fasta_graphs = []
        self.labels = []

        cache_basePath = self.pdb2graph_translator.get_graph_cache_folder
        ()

        self.smiles_graphs_cache_path = cache_basePath + "smiles_graphs/"
        smiles_dglgraph"
        self.fasta_graphs_cache_path = cache_basePath + "fasta_graphs/"
        fasta_dglgraph"
        self.smiles_cache_path = cache_basePath + "smiles/smiles"
```



```

self.labels_cache_path = cache_basePath + "labels/label"

self.cache_size = cache_size
self.inference = inference
self.append_itemId = append_itemId

self._mkdir(self.smiles_graphs_cache_path)
self._mkdir(self.fasta_graphs_cache_path)
self._mkdir(self.smiles_cache_path)
self._mkdir(self.labels_cache_path)

if target_col_name is None:
    self.task_names = self.df.columns.drop([smiles_col_name,
        pdb_id_col_name]+columns_to_ignore).tolist()
else:
    self.task_names = [target_col_name]
#

if foldId_col_name is not None:
    self._fold_list = self.df[foldId_col_name].values.tolist() #
        raw data, to be used to fill in self.foldIds
    self.foldIds = [] # will be filled in, and will contain the
        number of elems equal to num of VALID graphs
else:
    self._fold_list = None
    self.foldIds = None
#

self.n_tasks = len(self.task_names)
self._pre_process(smiles_to_graph, smiles_node_featurizer,
    smiles_edge_featurizer,
        smiles_col_name, pdb_id_col_name, load,
        log_every)

def _mkdir(self, path):
    dirname = os.path.dirname(path)
    if not os.path.exists(dirname):
        os.makedirs(dirname)

def _pre_process(self, smiles_to_graph, node_featurizer,
    edge_featurizer,
        smiles_col_name, pdb_id_col_name, load, log_every):

    mis_options_file = os.path.join(os.path.dirname(self.
        fasta_graphs_cache_path), 'misc_options.json')

    smiles_graphs_cache_dir = os.path.dirname(self.
        smiles_graphs_cache_path)
    fasta_graphs_cache_dir = os.path.dirname(self.
        fasta_graphs_cache_path)
    smiles_cache_dir = os.path.dirname(self.smiles_cache_path)
    labels_dir = os.path.dirname(self.labels_cache_path)

    smiles_graphs_cache_list = sorted([f for f in os.listdir(
        smiles_graphs_cache_dir) if f.endswith('.bin')])
    fasta_graphs_cache_list = sorted([f for f in os.listdir(
        fasta_graphs_cache_dir) if f.endswith('.bin')])
    smiles_cache_list = sorted([f for f in os.listdir(smiles_cache_dir
        ) if f.endswith('.bin')])

    if not self.inference:
        labels_list = sorted([f for f in os.listdir(labels_dir) if f.
            endsuffix('.bin')])

    if not load:
        # remove all previously existing graph cache files first!
        print('deleting old cache...')

```

```

dir_list = [smiles_graphs_cache_dir, fasta_graphs_cache_dir,
            smiles_cache_dir]
if not self.inference:
    dir_list.append(labels_dir)
#
for _dir in dir_list:
    flist = sorted([f for f in os.listdir(_dir) if f.endswith(
        '.bin')])
    for fnm in flist:
        os.remove(os.path.join(_dir, fnm))
    print('%s: %d deleted' % (_dir, len(flist)))
#
#
if load:
    # DGLGraphs have been constructed before, reload them
    print('Loading previously saved dgl graphs...')

    if self.inference:
        for smiles_graphs_cache_file,\
            fasta_graphs_cache_file,\
            smiles_cache_file,\
            fasta_cache_file in zip(smiles_graphs_cache_list,
                                    fasta_graphs_cache_list,
                                    smiles_cache_list,
                                    fasta_cache_list):
            smiles_graphs_cache_file = os.path.join(
                smiles_graphs_cache_dir, smiles_graphs_cache_file)
            fasta_graphs_cache_file = os.path.join(
                fasta_graphs_cache_dir, fasta_graphs_cache_file)
            smiles_cache_file = os.path.join(smiles_cache_dir,
                smiles_cache_file)
            fasta_cache_file = os.path.join(fasta_cache_dir,
                fasta_cache_file)

            smiles_graphs, _ = load_graphs(
                smiles_graphs_cache_file)
            fasta_graphs, _ = load_graphs(fasta_graphs_cache_file)

            self.smiles_graphs.extend(smiles_graphs)
            self.fasta_graphs.extend(fasta_graphs)

            with open(smiles_cache_file, 'rb') as f:
                self.smiles.extend(pickle.load(f))
    else:
        for smiles_graphs_cache_file,\
            fasta_graphs_cache_file,\
            smiles_cache_file,\
            labels_cache_file in zip(smiles_graphs_cache_list, #
                                     [:32] to limit cache blocks
                                     fasta_graphs_cache_list, #
                                     [:32]
                                     smiles_cache_list, # [:32]
                                     labels_list): # [:32]
            smiles_graphs_cache_file = os.path.join(
                smiles_graphs_cache_dir, smiles_graphs_cache_file)
            fasta_graphs_cache_file = os.path.join(
                fasta_graphs_cache_dir, fasta_graphs_cache_file)
            smiles_cache_file = os.path.join(smiles_cache_dir,
                smiles_cache_file)
            labels_cache_file = os.path.join(labels_dir,
                labels_cache_file)

            smiles_graphs, _ = load_graphs(
                smiles_graphs_cache_file)
            fasta_graphs, _ = load_graphs(fasta_graphs_cache_file)

```

```

        self.smiles_graphs.extend(smiles_graphs)
        self.fasta_graphs.extend(fasta_graphs)

        with open(smiles_cache_file, 'rb') as f:
            self.smiles.extend(pickle.load(f))
        with open(labels_cache_file, 'rb') as f:
            self.labels.extend(pickle.load(f))

    # restore misc. flags&options
    with open(mis_options_file) as jf:
        dat = json.load(jf)
        self.foldIds = dat['foldIds'] # will return a list or None

    print('smiles:', len(self.smiles))
    print('smiles_graphs:', len(self.smiles_graphs))
    print('fasta_graphs:', len(self.fasta_graphs))
    if not self.inference:
        self.labels = F.zerocopy_from_numpy(np.nan_to_num(self.labels).astype(np.float32))
        print('labels:', len(self.labels))
else:
    j_offset = 0
    i_offset = 0 #1
    print('Processing dgl graphs from scratch...')
    smiles_raw = self.df[smiles_col_name].values.tolist()
    pdb_id = self.df[pdb_id_col_name].values.tolist()
    labels_raw = self.df[self.task_names].values.tolist()
    last = len(smiles_raw) - 1
    prev = 0
    j = 0

    print('...to load:', len(smiles_raw), len(pdb_id), len(labels_raw))

    for i, (s, f, l) in enumerate(zip(smiles_raw, pdb_id, labels_raw)):
        if (i + i_offset) % log_every == 0:
            print('Processing graph {:d}/{:d}'.format(i+i_offset, len(self)))

        _loaded_ok = True
        try:
            sg = smiles_to_graph(s, node_featurizer=
                node_featurizer, edge_featurizer=edge_featurizer)
        except Exception as e:
            _loaded_ok = False
            print('Exception in smiles_to_graph in line: {}, for SMILES; {}, PDB_ID: {}'.format(i, s, f))
            print(e)

        try:
            fg = self.pdb2graph_translator.pdbId_to_graph(f)
        except Exception as e:
            _loaded_ok = False
            print('Exception in pdbId_to_graph in line: {}, for SMILES; {}, PDB_ID: {}'.format(i, s, f))
            print(e)
        if len(fg.ndata['h'])==0:
            _loaded_ok = False
            print('Error: PDB_ID {} in line {} yields graph with no nodes'.format(f, i))

        try:
            assert 'h' in fg.ndata, ('no nodes in graph obtained from', plg_filenames)
            assert 'e' in fg.edata, ('no edges in graph obtained from', plg_filenames)

```

```

        # skip iteration if any fbg keys except 'a', 'p', 'm'
        are present
except Exception as e:
    _loaded_ok = False
    print('Exception no_h/no_e in line: {}, for SMILES;
          {}, PDB_ID: {}'.format(i, s, f))
    print(e)

if _loaded_ok:
    self.smiles_graphs.append(sg)
    self.fasta_graphs.append(fg)
    self.smiles.append(s)

    if self.foldIds is not None:
        self.foldIds.append(self._fold_list[i]) # only for
            those graphs which have been loaded with no
            errors!

    if not self.inference:
        self.labels.append(l)
    j += 1

if (j + 1) % self.cache_size == 0 or i == last:
    print('Caching graph {:d}/{:d}'.format(j+j_offset, len
      (self)))
    save_graphs('{}_{:07d}.bin'.format(self.
      smiles_graphs_cache_path, j+j_offset+1), self.
      smiles_graphs[prev:j])
    save_graphs('{}_{:07d}.bin'.format(self.
      fasta_graphs_cache_path, j+j_offset+1), self.
      fasta_graphs[prev:j])

    with open('{}_{:07d}.bin'.format(self.
      smiles_cache_path, j+j_offset+1), 'wb') as f:
        pickle.dump(self.smiles[prev:j], f)
    if not self.inference:
        with open('{}_{:07d}.bin'.format(self.
          labels_cache_path, j+j_offset+1), 'wb') as f:
            pickle.dump(self.labels[prev:j], f)

    # dump misc. flags&options
    with open(mis_options_file, 'w') as jf:
        json.dump({'foldIds': self.foldIds}, jf)

    prev = j

print('smiles:', len(self.smiles))
print('smiles_graphs:', len(self.smiles_graphs))
print('fasta_graphs:', len(self.fasta_graphs))
if not self.inference:
    print('labels:', len(self.labels))
    # np.nan_to_num will also turn inf into a very large
    number
    self.labels = F.zerocopy_from_numpy(np.nan_to_num(self.
      labels).astype(np.float32))

def __getitem__(self, item):
    if self.append_itemId:
        return self.fasta_graphs[item], self.smiles[item], self.
          smiles_graphs[item], item

    if self.inference:
        return self.fasta_graphs[item], self.smiles[item], self.
          smiles_graphs[item]
    else:
        return self.fasta_graphs[item], self.smiles[item], self.
          smiles_graphs[item], self.labels[item]

```

```

#

def __len__(self):
    """Size for the dataset
    Returns
    -----
    int
        Size for the dataset
    """
    return len(self.smiles)

#

#

class FoldsOf_VPLGDataset:
    # Selects the items from source Dataset, according to conditions on
    # fold and/or on graph size
    def __init__(self, src = [], max_nodes = 7000):
        """ src - VPLGDataset to get data from;
            folds - list of foldIds to be returned by this object, of []
            to use all data
            max_nodes - int or function (returning True to use the graph,
            arg == graph)
        """
        assert isinstance(src, VPLGDataset)

        if callable(max_nodes): # type(max_nodes) is function:
            graph_filter = max_nodes
        else:
            # use the default condition
            def _default_filter_func(fg):
                if len(fg.ndata['h']) == 0:
                    print("Error: len(fg.ndata['h']) == 0")
                    return False

                #
                if max_nodes <= 0:
                    return True

                #
                res = len(fg.ndata['h']) <= max_nodes
                if not res:
                    print("skip: len(fg.ndata['h']) ==", len(fg.ndata['h']
                    ))
                return res

            #
            graph_filter = _default_filter_func

        #

        self.src = src
        self.raw_indices = []
        folds = set(folds)
        assert len(folds) >= 0
        if len(folds) > 0:
            assert src.foldIds is not None

        for j, item in enumerate(self.src):
            if len(folds) > 0:
                # first check, if fold is appropriate
                if src.foldIds[j] not in folds:
                    continue

            # of course, don't skip anything if len(folds) == 0 -- 'use
            # all folds'

            # Now, check if the graph satisfies selection conditions;
            # item is tuple of either (fasta_graph, smiles, smiles_graph)
            # or of
            # (fasta_graph, smiles, smiles_graph, label)

```

```

        if not graph_filter(item[0]):
            continue
        # else:
        self.raw_indices.append(j)
#
#
def asDataLoader(self, **kwargs):
    return DataLoader(dataset=self, **kwargs)

def __getitem__(self, item):
    return self.src[ self.raw_indices[item] ]

def __len__(self):
    return len(self.raw_indices)

```

ii Завантаження графів

```

import glob
import re
import torch
from dgl import DGLGraph
import numpy as np

class ProteinGraphMaker:
    """ 'Abstract' class describing the protein-to-graph converter; its
    derived classes
    are supposed to implement methods for parsing the specific input
    files (be it plain FASTA
    sequence, VPLG program outputs or whatever) and building DGLGraph
    as a result, as well
    as auxiliary methods for maintaining cache folder / best_model
    file naming conventions.
    For parsing, the implemented method should take pdb-id as the
    input and manage by itself
    all operations needed to find/load all paths/files/folder.
    """
    def __init__(self):
        typical_aa = ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L',
                     ', 'K', # 12
                     'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V'] # 8
        self.typical_aminoacids = {k: i for i,k in enumerate(typical_aa)}

    def get_graph_cache_folder(self):
        return "Graph_Cache/"

    def get_best_model_filename(self):
        return 'affinity_best.pth'

    def pdbId_to_graph(self, pdbId):
        return None

    def _append_edge(self, proteinEdges, proteinEdgeFeatures, jA, jB,
        feature_vec):
        " a simple method added just to keep double-adding consistent"
        # Note:
        # Edges in DGLGraph are directed by default -- https://docs.dgl.ai/tutorials/models/3\_generative\_model/5\_dgmg.html?highlight=undirected
        # DGLGraph is always directed. & In converting an undirected
        NetworkX graph into a DGLGraph, DGL internally
        # converts undirected edges to two directed edges -- https://docs.dgl.ai/guide/graph-external.html?highlight=undirected%20edges
        #
        # Note further that it might not seem nice to add each node twice
        here, but such choice will simplify

```

```

# the below code for inserting edge features into the graph
proteinEdges.append( (jA, jB) )
proteinEdges.append( (jB, jA) )

# add twice, to be consistent with (a,b),(b,a) duplication of
# edges in proteinEdges
proteinEdgeFeatures.append( feature_vec )
proteinEdgeFeatures.append( feature_vec )

def lists_to_graph(self, node_features, proteinEdges,
proteinEdgeFeatures):
# convert all the accumulated data into a graph
proteinGraph = DGLGraph()
nNodes = len(node_features)

# Nodes:
proteinGraph.add_nodes( nNodes )
proteinGraph.ndata['h'] = torch.Tensor(node_features)

# Edges:
for i,j in proteinEdges:
    proteinGraph.add_edge(i,j)
#
proteinGraph.edata['e'] = torch.Tensor(proteinEdgeFeatures)

return proteinGraph

class DSSP_loader(ProteinGraphMaker):
def __init__(self,
dssp_files_path,
includeAminoacidPhyschemFeatures = True,
cache_dir_prefix = ""):
#
super().__init__()

self.dssp_path = dssp_files_path

_all_letters_upper = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z']
_all_letters_lower = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
'z']

self.dssp_categorical_node_props_inv = {
'aminoacid': [
'V', 'M', 'D', 'N', 'R', 'L', 'I', 'A', 'T', 'G',
'E', 'Y', 'K', 'P', 'S', 'F', # 16 letters
'C', 'H', 'Q', 'W', #
'X', 'Z' # 'Z' is used by DSSP 4.0, but not by
CMBI DSSP
# +see an ad hoc tricj to include lowercase
letters is done below
],
'sse_class': ['', 'T', 'B', 'S', 'H', 'E', 'I', 'G', 'P'], # '
P' is used by original DSSP 4.0, but not by CMBI DSSP
'3_turns_helix': ['', '3', '<', '>', 'X'],
'4_turns_helix': ['', '4', '<', '>', 'X'],
'5_turns_helix': ['', '5', '<', '>', 'X'],
'geometrical_bend': ['', 'S'],
'chirality': ['', '+', '-'],
'beta_bridge_label_1': ['', ] + _all_letters_upper +
_all_letters_lower,
'beta_bridge_label_2': ['', ] + _all_letters_upper +

```

```

        _all_letters_lower,
        'beta_bridge_sheet_label': [''] + _all_letters_upper
} # actually, the inverse of it will be used to build features

# Convert it to property->str->one-hot-index map:
_map = {}
for k,v in self.dssp_categorical_node_props_inv.items():
    _map[k] = { vv: i for i,vv in enumerate(v) }
    _map[k][None] = len(v)

# map all 'unusual' aminoacids to 'others'
_map['aminoacid'][None] += 1
idxOtherAA = len(self.dssp_categorical_node_props_inv['aminoacid'])
# the one-hot index for 'others'
for x in _all_letters_lower:
    _map['aminoacid'][x] = idxOtherAA
#
self.dssp_categorical_node_props = _map

self.featureSetPrefix = 'noPCFAA'

self.cache_dir_prefix = cache_dir_prefix

#

#override
def get_graph_cache_folder(self):
    return '%s_dssp_%s/' % (self.cache_dir_prefix, self.
        featureSetPrefix)

#override
def get_best_model_filename(self):
    return '%s_best_model_dssp_%s.pth' % (self.cache_dir_prefix, self.
        featureSetPrefix)

def _parse_dssp_line(self, line):
    """ Parses a single line in the main section of .dssp file, and
        returns extracted
        data as dict
    """
    # 0+      10+      20+      30+      | 40+      50+
    # 60+      70+      80+      90
    #
    012345678901234567890123456789012345678|90123456789012345678901234567890123456789
    #'  5      6 A F E      -ab 36 227A  0 |  221, -1.6  223, -1.3
    30, -0.2      2, -0.5  -0.987'
    # 90+      100+      110+      120+      130+ |  140+
    150+      160+      170+      180
    #
    123456789012345678901234567890123456789012345|6789012345678901234567890123456789
    #'  14.3-175.8-133.0 122.3  96.3  35.3  60.4|
    A      5      '
    # 180+      190+      200+      210+      220+      230+      240+
    250+      260
    #
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
    #'  6      36      227      221      223      30
    2'
    r = {}
    r['dssp_res_num'] = int(line[:5]) # DSSP's sequential residue
        number, including _chain breaks_ (TODO)
    #r['pdb_res_num'] = int(line[5:11]) # crystallographers' 'residue
        sequence number'-given for reference only!
    r['chain_id'] = line[11].strip() # make chain breaks be ''
    r['aminoacid'] = line[13] # one letter amino acid code, lower case

```



```

        for SS-bridge CYS.
r['sse_class'] = line[16].strip() # TODO: line[17]=='*' when line
    [16]=='!' (chain break)
# line[16]=='!' - chain break residue detected as a discontinuity
    of backbone coordinates,
# DSSP also detects a discontinuity in the PDB-supplied chain
    identifier, recorded as '*'==line[17]
# // https://swift.cmbi.umcn.nl/gv/dssp/HTML/descrip.html

# TODO: dict/indices for the following 'secondary structure
    summary' (based on columns 19-38)
r['3_turns_helix'] = line[18].strip() # make it '' if empty (' ')
r['4_turns_helix'] = line[19].strip()
r['5_turns_helix'] = line[20].strip()
r['geometrical_bend'] = line[21].strip()
r['chirality'] = line[22].strip()
r['beta_bridge_label_1'] = line[23].strip()
r['beta_bridge_label_2'] = line[24].strip()

# residue number of first and second bridge partner followed by
    one letter sheet label:
r['beta_bridge_partner_resnum_1'] = int(line[25:29])
r['beta_bridge_partner_resnum_2'] = int(line[29:33])
r['beta_bridge_sheet_label'] = line[33].strip()

r['num_waters'] = int(line[34:38]) # number of water molecules in
    contact with this residue *10,
# or residue water exposed surface in Angstrom**2.

# Hydrogen bonding data:
for w, offs in zip(['first_NH_0', 'first_O_HN', 'second_NH_0', '
    second_O_HN'], [0,50-39,61-39,72-39]):
    r['%s_idx' % w] = int(line[offs+39 : offs+45])
    r['%s_E' % w] = float(line[offs+46 : offs+50])
#
r['cos_CO_CO'] = float(line[83:91]) # TCO - cosine of angle
    between C=O of residue I and C=O of residue I-1
r['kappa_bend_angle'] = float(line[91:97]) # KAPPA - bend angle,
    defined by C-alpha of residues I-2,I,I+2
r['alpha_torsion'] = float(line[97:103]) #ALPHA - torsion angle,
    defined by C-alpha atoms
    # of residues I-1,I,I+1,I+2. Used to define
    chirality (structure code '+' or '-').
r['phi'] = float(line[103:109]) # IUPAC peptide backbone torsion
    angles
r['psi'] = float(line[109:115])
r['r_Calpha'] = np.array([float(line[i:i+7]) for i in [115, 122,
    129]]) # C-alpha atom coordinates

return r

def _parse_dssp(self, fname):
# note that the lines in dssp files seem to consist of fixed-width
    fields, so that, e.g.,
# ' 27.5-102.6' is perfectly valid pair of floats
templ_totCount = 'TOTAL NUMBER OF RESIDUES, NUMBER OF CHAINS,
    NUMBER OF ' + \
        'SS-BRIDGES(TOTAL,INTRACHAIN,INTERCHAIN)'
templ_mainHdr = 'RESIDUE AA STRUCTURE BP1 BP2 ACC N-H-->O
    O-->H-N N-H-->O O-->H-N ' + \
        'TCO KAPPA ALPHA PHI PSI X-CA Y-CA Z-CA
    ,
nTotResidues = None
#nChains = None
active_section = None
chains = {} # chainId -> {'property':
    list_of_values_for_all_residues }

```

```

chains2 = {} # chainId -> [list of {property: value} dicts for all
              residues ]
#
with open(fname) as f:
    for line in f:
        line = line.rstrip()
        if line.find(templ_totCount) != -1:
            nTotResidues = int(line[:5])
            #nChains = int(line[5:8]) # when chain breaks are
            # present, this will
            # count 'pieces', not '
            # chains'

        if active_section == 'main':
            dat = self._parse_dssp_line(line)
            chainId = dat['chain_id']
            if len(chainId) > 0: # if not chain break record
                if chainId not in chains:
                    chains[chainId] = {}
                    chains2[chainId] = []
                del dat['chain_id']
                for k,v in dat.items():
                    if k not in chains[chainId]:
                        chains[chainId][k] = []
                        chains[chainId][k].append(v)
                #
                chains2[chainId].append(dat)
            #
            if line.find(templ_mainHdr) != -1:
                active_section = 'main'
        #
#
# because of numbering chain breaks as a separate entities, we can
# not do something line
#assert len(chains.keys()) == nChains, ('inconsistent number of
# chains in', fname)
return chains, chains2

def explore_property_values(self, pdbId, dest_dict):
    """ Merely parses .dssp files and updates dest_dict with possible
        str-type values of the residue properties
    """
    assert type(dest_dict) is dict
    #
    chains, _ = self._parse_dssp(self.dssp_path + pdbId + '.cif.dssp')
    for chainId, chain in chains.items():
        for prop_name, prop_list in chain.items():
            # len(prop_list)>0 always - otherwise, prop_name is not in
            # this dict
            for p in prop_list:
                if type(p) is str:
                    if prop_name not in dest_dict:
                        dest_dict[prop_name] = {}
                    if p not in dest_dict[prop_name]:
                        dest_dict[prop_name][p] = 0
                    dest_dict[prop_name][p] += 1
    return dest_dict # for convenience only; dest_dict has already
                    # been updated
#

def _build_node_features_vec(self, rec):
    self.ignored_props = ['dssp_res_num',
                          'beta_bridge_label_1', 'beta_bridge_label_2',
                          , 'beta_bridge_sheet_label',
                          'first_NH_0_idx', 'first_O_HN_idx', ,
                          second_NH_0_idx', 'second_O_HN_idx',
                          'first_NH_0_E', 'first_O_HN_E', ,
                          second_NH_0_E', 'second_O_HN_E',

```

```

        'beta_bridge_partner_resnum_1', '
        beta_bridge_partner_resnum_2',
        'alpha_torsion', 'phi', 'psi', 'r_Calpha'
    ]
    #self.int_and_float_props = [ 'num_waters', 'cos_CO_CO', '
        kappa_bend_angle' ] #TODO: should cos_CO_CO be here?
    result = []
    # one-hot encodings for categorial properties:
    #for prop_name, propVal2idx in self.dssp_categorical_node_props.
        items():
    for prop_name in ['aminoacid', 'sse_class']:
        propVal2idx = self.dssp_categorical_node_props[prop_name]

        vec = [0] * propVal2idx[None]
        str_prop = rec[prop_name]
        prop_idx = propVal2idx[str_prop]
        vec[ prop_idx ] = 1
        result += vec

    return result

def pdbId_to_graph(self, pdbId):
    """ The key method to read-in fasta from file associated with
        pdbId,
        parse it, convert to a graph and return the obtained DGLGraph
    """

    node_features = []
    proteinEdges = []
    proteinEdgeFeatures = []

    _, chains2 = self._parse_dssp(self.dssp_path + pdbId + '.cif.dssp'
        )

    resnum2idx = {} # dsspResNum -> indexOfNodeIn_node_features; will
        be needed to make edges for H-bonds

    _pre_h_bonds = [] # list of edges: [(nodeFrom, nodeTo, h_bond_type
        ('NH_0' or 'O_HN'), energy ), ...]

    num_edge_features = 5 # not including the 0-th feature: 1==peptide
        , 0==h-bond
    # E_hb; alpha, phi, psi, reserved

    for chainId, theChain in chains2.items():
        for iResidue, residue_rec in enumerate(theChain):
            ftrs = self._build_node_features_vec(residue_rec)
            node_features.append( ftrs )
            idDsspResnum = residue_rec['dssp_res_num']
            resnum2idx[idDsspResnum] = len(node_features)-1
            # TODO: check recipr.
            for ik, k in enumerate(['first_NH_0', 'first_O_HN', '
                second_NH_0', 'second_O_HN']):
                jj = residue_rec[k+'_idx']
                hb_E = residue_rec[k+'_E']
                if jj != 0:
                    # here, we only save this edge, but don't add it
                        to the graph yet!
                    _pre_h_bonds.append( (idDsspResnum, idDsspResnum +
                        jj, k[-4:], hb_E ) )

            # edges describing the peptide bonds (within the current
                chain only!):
            if iResidue >= 1:
                I, J = len(node_features)-1, len(node_features)-2 # '
                    global' node numbers, so to say
                edge_feature_vec = [1] + [0]*num_edge_features

```

```

# ^+- this first '1' distinguishes
# peptide bond from h-bond
self._append_edge(proteinEdges, proteinEdgeFeatures, I
, J, edge_feature_vec)
#
#
uniq_hbonds = {}
for i,j,hb_type,E_hb in _pre_h_bonds:
    ij = (i,j) if i<j else (j,i)
    if ij not in uniq_hbonds:
        uniq_hbonds[ij] = []
    uniq_hbonds[ij].append( E_hb )
for (i,j), E_hbs in uniq_hbonds.items():
    # create only a single edge connecting these residues, and set
    # its energy to sum of HB energies
    # in this way we'll characterize the total 'strength' of the
    # edge
    edge_feature_vec = [0] + [0]*num_edge_features
    # ^+- this first '0' distinguishes hydrogen
    # bond from peptide bond
    edge_feature_vec[1] = np.sum(E_hbs)
    assert i in resnum2idx, (pdbId, i, 'not in resnum2idx')
    assert j in resnum2idx, (pdbId, j, 'not in resnum2idx')
    I,J = resnum2idx[i], resnum2idx[j]
    self._append_edge(proteinEdges, proteinEdgeFeatures, I, J,
    edge_feature_vec)
# now just convert all the accumulated data into a graph
return self.lists_to_graph(node_features, proteinEdges,
proteinEdgeFeatures)
#
#

```

iii Архітектура графової нейронної мережі для прогнозування молекулярної спорідненості

```

import torch

import torch.nn as nn

from dgllife.model.gnn.attentivefp import AttentiveFPGNN
from dgllife.model.readout import AttentiveFPReadout
from dgllife.model.model_zoo.mlp_predictor import MLPPredictor

__all__ = ['mhGANN']

class mhGANN(nn.Module):
    """AttentiveFP for regression and classification on graphs.
    AttentiveFP is introduced in
    'Pushing the Boundaries of Molecular Representation for Drug Discovery
    with the Graph
    Attention Mechanism. <https://www.ncbi.nlm.nih.gov/pubmed/31408336>'
    """
    def __init__(self,
        smiles_node_feat_size,
        smiles_edge_feat_size,
        fasta_node_feat_size,
        fasta_edge_feat_size,
        smiles_num_layers=2,
        smiles_num_timesteps=2,
        smiles_graph_feat_size=200,
        fasta_num_layers=2,
    ):

```

```

        fasta_num_timesteps=2,
        fasta_graph_feat_size=200,
        n_tasks=1,
        dropout=0.,
        mlp_hidden_layer=256):
super(mhGANN, self).__init__()

self.smiles_gnn = AttentiveFPGNN(node_feat_size=
    smiles_node_feat_size,
                                edge_feat_size=
                                smiles_edge_feat_size,
                                num_layers=smiles_num_layers,
                                graph_feat_size=
                                smiles_graph_feat_size,
                                dropout=dropout)

self.fasta_gnn = AttentiveFPGNN(node_feat_size=
    fasta_node_feat_size,
                                edge_feat_size=
                                fasta_edge_feat_size,
                                num_layers=fasta_num_layers,
                                graph_feat_size=
                                fasta_graph_feat_size,
                                dropout=dropout)

self.smiles_readout = AttentiveFPReadout(feats_size=
    smiles_graph_feat_size,
                                          num_timesteps=
                                          smiles_num_timesteps,
                                          dropout=dropout)

self.fasta_readout = AttentiveFPReadout(feats_size=
    fasta_graph_feat_size,
                                          num_timesteps=
                                          fasta_num_timesteps,
                                          dropout=dropout)

self.predict = MLPPredictor(
    smiles_graph_feat_size+fasta_graph_feat_size,
    # input layer size
    smiles_graph_feat_size+fasta_graph_feat_size,
    # 400, # hidden layer size
    n_tasks, dropout)

#
def forward(self, smiles_g, fasta_g,
            smiles_node_feats, smiles_edge_feats,
            plg_node_feats, plg_edge_feats, get_node_weight=False):
    """ Graph-level regression
    """
    smiles_node_feats = self.smiles_gnn(smiles_g, smiles_node_feats,
                                        smiles_edge_feats)
    plg_node_feats = self.fasta_gnn(fasta_g, plg_node_feats,
                                    plg_edge_feats)
    if get_node_weight:
        smiles_g_feats, smiles_node_weights = self.smiles_readout(
            smiles_g, smiles_node_feats, get_node_weight)
        plg_g_feats, fasta_node_weights = self.fasta_readout(fasta_g,
                                                             plg_node_feats,
                                                             get_node_weight)
        g_feats = torch.cat((smiles_g_feats, plg_g_feats), dim=1)
        return self.predict(g_feats), smiles_node_weights,
            fasta_node_weights
    else:
        smiles_g_feats = self.smiles_readout(smiles_g,
                                             smiles_node_feats)
        plg_g_feats = self.fasta_readout(fasta_g, plg_node_feats)
        g_feats = torch.cat((smiles_g_feats, plg_g_feats), dim=1)

```

```
return self.predict(g_feats)
```

iv Налаштування графової нейронної мережі

```
from dgllife.utils import smiles_to_complete_graph, WeaveAtomFeaturizer,
    WeaveEdgeFeaturizer
from functools import partial
```

```
current_config = {
    'random_seed': 8,
    'mlp_hidden_layer': 256,
    'smiles_num_layers': 2,
    'smiles_num_timesteps': 2,
    'smiles_graph_feat_size': 200,
    'fasta_num_layers': 2,
    'fasta_num_timesteps': 2,
    'fasta_graph_feat_size': 200,
    'n_tasks': 1,
    'dropout': 0.05,
    'weight_decay': 10 ** (-5.0),
    'lr': 0.001,
    'batch_size': 32,
    'num_epochs': 1000,
    'frac_train': 0.8,
    'frac_val': 0.1,
    'frac_test': 0.1,
    'patience': 50,
    'metric_name': 'mae',
    'model': 'mhGANN',
    'mode': 'lower',
    'smiles_to_graph': partial(smiles_to_complete_graph, add_self_loop=
        True),
    'smiles_node_featurizer': WeaveAtomFeaturizer(),
    'smiles_edge_featurizer': WeaveEdgeFeaturizer(max_distance=2),
    'load_checkpoint': False,
    'inference': False,
    'argv_valFold': 4,
    'argv_testFold': 4,
    'cache_dir_prefix': 'graph_cache',
    'dssp_files_path': 'example_data/dssp/',
    'master_data_table': "example_data/input.csv"
}

def get_config():
    return current_config
```

v Алгоритм тренування графової нейронної мережі

```
#!/usr/bin/env python
# coding: utf-8
```

```
import torch
import torch.nn as nn

from affinity_module.config import get_config

import numpy as np

from dgllife.utils import EarlyStopping
from affinity_module.utils import set_random_seed, load_model
```

```

from affinity_module.utils import run_a_train_epoch, run_stat_epoch,
    run_an_eval_epoch
from affinity_module.utils import Collate
from affinity_module.dataset import VPLGDataset, FoldsOf_VPLGDataset

from affinity_module.protein_graph_loaders import DSSP_loader

from torch.backends import cudnn

cudnn.deterministic = True
cudnn.benchmark = False

args = get_config()
collate = Collate(args)
args['device'] = torch.device("cuda: 0") if torch.cuda.is_available() else
    torch.device("cpu")
set_random_seed(args['random_seed'])

argv_valFold = args['argv_valFold']
argv_testFold = args['argv_testFold']
cache_dir_prefix = args['cache_dir_prefix']

pdb2graph_translator = DSSP_loader(dssp_files_path = args['dssp_files_path
    '],
                                includeAminoacidPhyschemFeatures =
                                    False,
                                cache_dir_prefix = cache_dir_prefix)

best_model_filename = pdb2graph_translator.get_best_model_filename()

_colNames = dict(master_data_table = args['master_data_table'],
                 pdb_id_col_name="PDBs", smiles_col_name="SMILES",
                 target_col_name="logKi",
                 foldId_col_name='Fold')

dataset = VPLGDataset(
    smiles_to_graph=args['smiles_to_graph'],
    smiles_node_featurizer=args['smiles_node_featurizer'],
    smiles_edge_featurizer=args['smiles_edge_featurizer'],
    **_colNames,
    pdb2graph_translator = pdb2graph_translator,
    load=False)

args['device'] = torch.device("cuda: 0") if torch.cuda.is_available() else
    torch.device("cpu")

raw_dataset = VPLGDataset(
    smiles_to_graph=args['smiles_to_graph'],
    smiles_node_featurizer=args['smiles_node_featurizer'],
    smiles_edge_featurizer=args['smiles_edge_featurizer'],
    **_colNames,
    pdb2graph_translator = pdb2graph_translator,
    load=True)

args['fasta_node_feat_size'] = raw_dataset.fasta_graphs[0].ndata['h'].
    shape[1]
args['fasta_edge_feat_size'] = raw_dataset.fasta_graphs[0].edata['e'].
    shape[1]
print(args['fasta_node_feat_size'], args['fasta_edge_feat_size'])

print('will save best model to:', best_model_filename)

```

```

shfl = True

p = dict(batch_size=args['batch_size'], shuffle=shfl, collate_fn=collate.
collate_molgraphs)

mx_nodes = 5000 # filter out large graphs
folds5 = [0,1,2,3,4]
folds5.remove(argv_valFold)
if argv_testFold in folds5:
    folds5.remove(argv_testFold)
print('Folds to use: train=%s, val=%s, test=%s' % (str(folds5), str(
argv_valFold), str(argv_testFold)) )
#
train_loader = FoldsOf_VPLGDataset(raw_dataset, folds5, max_nodes =
mx_nodes).asDataLoader(**p)
val_loader = FoldsOf_VPLGDataset(raw_dataset, [argv_valFold], max_nodes
= mx_nodes ).asDataLoader(**p)
test_loader = FoldsOf_VPLGDataset(raw_dataset, [argv_testFold], max_nodes
= mx_nodes ).asDataLoader(**p)
print('number of batches: train %d, val %d, test %d' % (len(train_loader),
len(val_loader), len(test_loader)))

torch.cuda.empty_cache()

model = load_model(args)
loss_fn = nn.MSELoss(reduction='none')

optimizer = torch.optim.Adam(model.parameters(), lr=args['lr'],
weight_decay=args['weight_decay'])

stopper = EarlyStopping(mode=args['mode'],
patience=args['patience'],
filename=best_model_filename)

if args['load_checkpoint']:
    print('Loading checkpoint...')
    stopper.load_checkpoint(model)
model.to(args['device'])

for epoch in range(args['num_epochs']):
    # Train
    run_a_train_epoch(args, epoch, model, train_loader, loss_fn, optimizer
    )

    # Validation and early stop
    val_score_ext = run_stat_epoch(args, model, val_loader)
    val_score = val_score_ext[ args['metric_name'] ]

    test_score = run_an_eval_epoch(args, model, test_loader)
    early_stop = stopper.step(val_score, model)
    print('epoch {:d}/{:d}, validation {} {:.4f}, test {} {:.4f}, best
validation {} {:.4f}'.format(
epoch + 1, args['num_epochs'], args['metric_name'], val_score,
args['metric_name'], test_score,
args['metric_name'], stopper.best_score),
', now R2 = %.4f' % val_score_ext['R2'])

    if early_stop:
        break

print('-'*80)
stopper.load_checkpoint(model)

print()

all_metrics = {}
for dsName, data_loader in zip(['train', 'val', 'test'], [train_loader,

```



```

    val_loader, test_loader]):
        metrics, _, y_true, y_pred = run_stat_epoch(args, model, data_loader,
            return_pred=True)
        all_metrics[dsName] = (metrics, y_true, y_pred)

print('-'*50)

all_metric_names = set()
for dsName in ['train', 'val', 'test']:
    metrics, _, _ = all_metrics[dsName]
    for k in metrics.keys():
        all_metric_names.add(k)
#

print('%25s' % '', end='')
for dsName in ['train', 'val', 'test']:
    print('%12s' % dsName, end='')
print()
for mName in all_metric_names:
    print('%25s' % mName, end='')
    for dsName in ['train', 'val', 'test']:
        metrics, _, _ = all_metrics[dsName]
        print('%12.5f' % metrics[mName], end='')
    print()
#
print('-'*50)

_baseFname = pdb2graph_translator.get_best_model_filename().replace('.pth',
, '')

for dsName in ['train', 'val', 'test']:
    _, y_true, y_pred = all_metrics[dsName]
    y_true = np.array(y_true)[: , 0]
    y_pred = np.array(y_pred)

    np.savetxt(_baseFname+'_yy_%s.txt' % dsName, np.vstack((y_true, y_pred
    )).T,
        header='y_true, y_pred (%s)' % dsName )

```

III Додаток В. ПРОГРАМНИЙ КОД МЕТОДУ ПРОГНОЗУВАННЯ ВИХОДУ ПРОДУКТУ ХІМІЧНОЇ РЕАКЦІЇ

i Архітектура графової нейронної мережі

```
from argparse import Namespace
from typing import List, Union
from copy import deepcopy
import torch
import torch.nn as nn
import numpy as np

from .mpn import MPN, MPNDiff
from chemprop.features import BatchMolGraph
from chemprop.nn_utils import get_activation_function, initialize_weights

class Model(nn.Module):
    """Base class for molecule and reaction models, which only differ in
    their encoding."""

    def __init__(self, classification: bool, multiclass: bool):
        """
        Initializes the Model.

        :param classification: Whether the model is a classification model
        """
        super(Model, self).__init__()

        self.classification = classification
        if self.classification:
            self.sigmoid = nn.Sigmoid()
        self.multiclass = multiclass
        if self.multiclass:
            self.multiclass_softmax = nn.Softmax(dim=2)
        assert not (self.classification and self.multiclass)

    def create_encoder(self, args: Namespace):
        """
        Creates the encoder for the model.

        Should be overridden by all subclasses.
        """
        raise NotImplementedError

    def create_ffn(self, args: Namespace):
        """
        Creates the feed-forward network for the model.

        :param args: Arguments.
        """
        self.multiclass = args.dataset_type == 'multiclass'
        if self.multiclass:
            self.num_classes = args.multiclass_num_classes
        if args.features_only:
            first_linear_dim = args.features_size
        else:
            first_linear_dim = args.diff_hidden_size if args.reaction else
            args.hidden_size
```

```

        if args.use_input_features:
            first_linear_dim += args.features_dim

dropout = nn.Dropout(args.dropout)
activation = get_activation_function(args.activation)

# Create FFN layers
if args.ffn_num_layers == 1:
    ffn = [
        dropout,
        nn.Linear(first_linear_dim, args.output_size)
    ]
else:
    ffn = [
        dropout,
        nn.Linear(first_linear_dim, args.ffn_hidden_size)
    ]
    for _ in range(args.ffn_num_layers - 2):
        ffn.extend([
            activation,
            dropout,
            nn.Linear(args.ffn_hidden_size, args.ffn_hidden_size),
        ])
    ffn.extend([
        activation,
        # dropout,
        # nn.Linear(args.ffn_hidden_size, args.output_size),
    ])

# Create FFN model
self.ffn = nn.Sequential(*ffn)
ffn2 = [nn.Dropout(args.dropout), nn.Linear(args.ffn_hidden_size,
        args.output_size)]
self.ffn2 = nn.Sequential(*ffn2)

def forward(self, *input):
    """
    Defines the computation performed at every call.

    Should be overridden by all subclasses.
    """
    raise NotImplementedError

class MoleculeModel(Model):
    """A MoleculeModel is a model which contains a message passing network
    following by feed-forward layers."""

    def __init__(self, classification: bool, multiclass: bool):
        """
        Initializes the MoleculeModel.

        :param classification: Whether the model is a classification model
        """
        super(MoleculeModel, self).__init__(classification, multiclass)

    def create_encoder(self, args: Namespace):
        """
        Creates the message passing encoder for the model.

        :param args: Arguments.
        """
        self.encoder = MPN(args)

        if args.freeze_mpn:
            for param in self.encoder.parameters():

```

```

        param.requires_grad = False

def forward(self, *input):
    """
    Runs the MoleculeModel on input.

    :param input: Input.
    :return: The output of the MoleculeModel.
    """
    output = self.ffn(self.encoder(*input))

    # Don't apply sigmoid during training b/c using BCEWithLogitsLoss
    if self.classification and not self.training:
        output = self.sigmoid(output)
    if self.multiclass:
        output = output.reshape((output.size(0), -1, self.num_classes)
                                ) # batch size x num targets x num classes per target
        if not self.training:
            output = self.multiclass_softmax(output) # to get
                probabilities during evaluation, but not during
                training as we're using CrossEntropyLoss

    return output

class ReactionModel(Model):
    """
    A ReactionModel is a model which contains the same message passing
    network for the reactant and product molecules, followed by the
    formation of difference features and subsequent encoding with a
    difference message passing network, followed by feed-forward layers.
    """

    def __init__(self, classification: bool, multiclass: bool):
        """
        Initializes the ReactionModel.

        :param classification: Whether the model is a classification model
        """
        super(ReactionModel, self).__init__(classification, multiclass)

    def create_encoder(self, args: Namespace):
        """
        Creates the message passing encoders for the model.

        :param args: Arguments.
        """
        self.encoder = MPN(args, return_atom_hiddens=True)
        self.diff_encoder = MPNDiff(args, atom_fdim=args.hidden_size)

        if args.freeze_mpn:
            for param in self.encoder.parameters():
                param.requires_grad = False
        if args.freeze_mpn_diff:
            for param in self.diff_encoder.parameters():
                param.requires_grad = False

    def forward(self,
                rbatch: Union[List[str], BatchMolGraph],
                pbatch: Union[List[str], BatchMolGraph],
                features_batch: List[np.ndarray] = None) -> torch.
                FloatTensor:
        """
        Runs the ReactionModel on input.

        :param rbatch: A list of SMILES strings or a BatchMolGraph for the

```

```

        reactants.
    :param pbatch: A list of SMILES strings or a BatchMolGraph for the
        products.
    :param features_batch: A list of ndarrays containing additional
        features.
    :return: The output of the ReactionModel.
    """
    r_atom_features = self.encoder(rbatch)
    p_atom_features = self.encoder(pbatch)

    diff_features = p_atom_features - r_atom_features

    output, num_not_zero_diff, middle = self.diff_encoder(
        diff_features, pbatch, features_batch)

    output = self.ffn(output) # use product graph
    after_diff_encoder_layer = output.detach().data.cpu().numpy()
    output = self.ffn2(output)
    # Don't apply sigmoid during training b/c using BCEWithLogitsLoss
    if self.classification and not self.training:
        output = self.sigmoid(output)
    if self.multiclass:
        output = output.reshape(
            (output.size(0), -1, self.num_classes)) # batch size x
            num targets x num classes per target
        if not self.training:
            output = self.multiclass_softmax(
                output) # to get probabilities during evaluation, but
                not during training as we're using
                CrossEntropyLoss

    return output, num_not_zero_diff, middle, after_diff_encoder_layer

def build_model(args: Namespace) -> nn.Module:
    """
    Builds a MoleculeModel or ReactionModel, which is a message passing
    neural network + feed-forward layers.

    :param args: Arguments.
    :return: A MoleculeModel/ReactionModel containing the MPN encoder
        along with final linear layers with parameters initialized.
    """
    output_size = args.num_tasks
    args.output_size = output_size
    if args.dataset_type == 'multiclass':
        args.output_size *= args.multiclass_num_classes

    if args.reaction:
        model = ReactionModel(classification=args.dataset_type == '
            classification',
                               multiclass=args.dataset_type == 'multiclass'
                               )
    else:
        model = MoleculeModel(classification=args.dataset_type == '
            classification',
                               multiclass=args.dataset_type == 'multiclass'
                               )

    model.create_encoder(args)
    model.create_ffn(args)

    initialize_weights(model)

    return model

from argparse import Namespace
from typing import List, Union

```

```

import torch
import torch.nn as nn
import numpy as np

from copy import deepcopy

from chemprop.features import BatchMolGraph, get_atom_fdim, get_bond_fdim,
    mol2graph
from chemprop.nn_utils import index_select_ND, get_activation_function

class MPNEncoder(nn.Module):
    """A message passing neural network for encoding a molecule."""

    def __init__(self, args: Namespace, atom_fdim: int, bond_fdim: int,
        return_atom_hiddens: bool = False):
        """Initializes the MPNEncoder.

        :param args: Arguments.
        :param atom_fdim: Atom features dimension.
        :param bond_fdim: Bond features dimension.
        :param return_atom_hiddens: Return hidden atom feature vectors
            instead of mol vector.
        """
        super(MPNEncoder, self).__init__()
        self.return_atom_hiddens = return_atom_hiddens
        self.atom_fdim = atom_fdim
        self.bond_fdim = bond_fdim
        self.hidden_size = args.hidden_size
        self.bias = args.bias
        self.depth = args.depth
        self.dropout = args.dropout
        self.layers_per_message = 1
        self.undirected = args.undirected
        self.atom_messages = args.atom_messages
        self.features_only = args.features_only
        self.use_input_features = args.use_input_features
        self.args = args

        if self.features_only:
            return

        # Dropout
        self.dropout_layer = nn.Dropout(p=self.dropout)

        # Activation
        self.act_func = get_activation_function(args.activation)

        # Cached zeros
        self.cached_zero_vector = nn.Parameter(torch.zeros(self.
            hidden_size), requires_grad=False)

        # Input
        input_dim = self.atom_fdim if self.atom_messages else self.
            bond_fdim
        self.W_i = nn.Linear(input_dim, self.hidden_size, bias=self.bias)

        if self.atom_messages:
            w_h_input_size = self.hidden_size + self.bond_fdim
        else:
            w_h_input_size = self.hidden_size

        # Shared weight matrix across depths (default)
        if self.depth > 1:
            self.W_h = nn.Linear(w_h_input_size, self.hidden_size, bias=
                self.bias)

```

```

self.W_o = nn.Linear(self.atom_fdim + self.hidden_size, self.
    hidden_size)

def forward(self,
            mol_graph: BatchMolGraph,
            features_batch: List[np.ndarray] = None) -> torch.
            FloatTensor:
    """
    Encodes a batch of molecular graphs.

    :param mol_graph: A BatchMolGraph representing a batch of
        molecular graphs.
    :param features_batch: A list of ndarrays containing additional
        features.
    :return: A PyTorch tensor of shape (num_molecules, hidden_size)
        containing the encoding of each molecule.
    """
    # print(self.use_input_features)
    if self.use_input_features and not self.return_atom_hiddens:
        features_batch = torch.from_numpy(np.stack(features_batch)).
            float()

        if self.args.cuda:
            features_batch = features_batch.cuda()

        if self.features_only:
            return features_batch

    f_atoms, f_bonds, a2b, b2a, b2revb, a_scope, b_scope = mol_graph.
        get_components()

    if self.atom_messages:
        a2a = mol_graph.get_a2a()

    if self.args.cuda or next(self.parameters()).is_cuda:
        f_atoms, f_bonds, a2b, b2a, b2revb = f_atoms.cuda(), f_bonds.
            cuda(), a2b.cuda(), b2a.cuda(), b2revb.cuda()

        if self.atom_messages:
            a2a = a2a.cuda()

    # Input
    if self.atom_messages:
        input = self.W_i(f_atoms) # num_atoms x hidden_size
    else:
        input = self.W_i(f_bonds) # num_bonds x hidden_size
    message = self.act_func(input) # num_bonds x hidden_size

    # Message passing
    for depth in range(self.depth - 1):
        if self.undirected:
            message = (message + message[b2revb]) / 2

        if self.atom_messages:
            nei_a_message = index_select_ND(message, a2a) # num_atoms
                x max_num_bonds x hidden
            nei_f_bonds = index_select_ND(f_bonds, a2b) # num_atoms x
                max_num_bonds x bond_fdim
            nei_message = torch.cat((nei_a_message, nei_f_bonds),
                dim=2) # num_atoms x
                    max_num_bonds x hidden +
                    bond_fdim
            message = nei_message.sum(dim=1) # num_atoms x hidden +
                bond_fdim
        else:
            #  $m(a_1 \rightarrow a_2) = [\sum_{a_0 \in \text{nei}(a_1)} m(a_0 \rightarrow a_1)] - m(a_2)$ 

```

```

        -> a1)
        # message      a_message = sum(nei_a_message)
        rev_message
        nei_a_message = index_select_ND(message, a2b) # num_atoms
        x max_num_bonds x hidden
        a_message = nei_a_message.sum(dim=1) # num_atoms x hidden
        rev_message = message[b2revb] # num_bonds x hidden
        message = a_message[b2a] - rev_message # num_bonds x
        hidden

        message = self.W_h(message)
        message = self.act_func(input + message) # num_bonds x
        hidden_size
        message = self.dropout_layer(message) # num_bonds x hidden

a2x = a2a if self.atom_messages else a2b
nei_a_message = index_select_ND(message, a2x) # num_atoms x
max_num_bonds x hidden
a_message = nei_a_message.sum(dim=1) # num_atoms x hidden
a_input = torch.cat([f_atoms, a_message], dim=1) # num_atoms x (
atom_fdim + hidden)
atom_hiddens = self.act_func(self.W_o(a_input)) # num_atoms x
hidden
atom_hiddens = self.dropout_layer(atom_hiddens) # num_atoms x
hidden

if self.return_atom_hiddens:
    return atom_hiddens

# Readout
mol_vecs = []
for i, (a_start, a_size) in enumerate(a_scope):
    if a_size == 0:
        mol_vecs.append(self.cached_zero_vector)
    else:
        cur_hiddens = atom_hiddens.narrow(0, a_start, a_size)
        mol_vec = cur_hiddens # (num_atoms, hidden_size)

        mol_vec = mol_vec.sum(dim=0) / a_size
        mol_vecs.append(mol_vec)

mol_vecs = torch.stack(mol_vecs, dim=0) # (num_molecules,
hidden_size)

if self.use_input_features:
    features_batch = features_batch.to(mol_vecs)
    if len(features_batch.shape) == 1:
        features_batch = features_batch.view([1, features_batch.
shape[0]])
    mol_vecs = torch.cat([mol_vecs, features_batch], dim=1) # (
num_molecules, hidden_size)

return mol_vecs # num_molecules x hidden

```

```

class MPNDiffEncoder(nn.Module):
    """A message passing neural network for encoding of custom (difference
) features."""

    def __init__(self, args: Namespace, atom_fdim: int):
        """Initializes the MPNDiffEncoder.

        :param args: Arguments.
        :param atom_fdim: Atom features dimension.
        """
        super(MPNDiffEncoder, self).__init__()
        self.atom_fdim = atom_fdim

```



```

self.bond_fdim = get_bond_fdim(args)
self.hidden_size = args.diff_hidden_size
self.bias = args.bias
self.depth = args.depth_diff
self.dropout = args.dropout
self.layers_per_message = 1
self.undirected = args.undirected
self.use_input_features = args.use_input_features
self.args = args

# Dropout
self.dropout_layer = nn.Dropout(p=self.dropout)

# Activation
self.act_func = get_activation_function(args.activation)

# Cached zeros
self.cached_zero_vector = nn.Parameter(torch.zeros(self.
    hidden_size), requires_grad=False)

# Input
self.W_i = nn.Linear(self.atom_fdim, self.hidden_size, bias=self.
    bias)

# Shared weight matrix across depths (default)
if self.depth > 1:
    self.W_h = nn.Linear(self.hidden_size + self.bond_fdim, self.
        hidden_size, bias=self.bias)

if self.depth > 0:
    self.W_o = nn.Linear(self.atom_fdim + self.hidden_size, self.
        hidden_size)

def forward(self,
    atom_features: torch.FloatTensor,
    mol_graph: BatchMolGraph,
    features_batch: List[np.ndarray] = None) -> torch.
    FloatTensor:
    """
    Encodes a batch of molecular graphs with custom features.

    :param atom_features: Atom features for the BatchMolGraph.
    :param mol_graph: A BatchMolGraph representing a batch of
        molecular graphs.
    :param features_batch: A list of ndarrays containing additional
        features.
    :return: A PyTorch tensor of shape (num_molecules, hidden_size)
        containing the encoding of each molecule.
    """
    num_not_zero_diff = []
    if self.use_input_features:
        features_batch = torch.from_numpy(np.stack(features_batch)).
            float()

        if self.args.cuda:
            features_batch = features_batch.cuda()

    f_atoms, f_bonds, a2b, b2a, b2revb, a_scope, b_scope = mol_graph.
        get_components()
    a2a = mol_graph.get_a2a()

    if self.args.cuda or next(self.parameters()).is_cuda:
        f_bonds, a2b, a2a = f_bonds.cuda(), a2b.cuda(), a2a.cuda()

    for i, (a_start, a_size) in enumerate(a_scope):
        af = atom_features.narrow(0, a_start, a_size)
        num_not_zero_diff.append([torch.sum((torch.sum(af, dim=1) > 0)

```

```

        ).item(), a_size])

# Input
input = self.W_i(atom_features) # num_atoms x atom_fdim
message = self.act_func(input) # num_atoms x hidden_size

if self.depth > 0:
    # Message passing
    for depth in range(self.depth - 1):
        nei_a_message = index_select_ND(message, a2a) # num_atoms
            x max_num_bonds x hidden
        nei_f_bonds = index_select_ND(f_bonds, a2b) # num_atoms x
            max_num_bonds x (bond_fdim (+ atom_fdim_MPN))

        # If using bond messages in MPN, bond features include
        # some atom features,
        # but we only want the pure bond features here
        nei_f_bonds = nei_f_bonds[:, :, -self.bond_fdim:]

        nei_message = torch.cat((nei_a_message, nei_f_bonds),
                                dim=2) # num_atoms x
                                    max_num_bonds x hidden +
                                    bond_fdim
        message = nei_message.sum(dim=1) # num_atoms x hidden +
            bond_fdim

        message = self.W_h(message)
        message = self.act_func(input + message) # num_atoms x
            hidden_size
        message = self.dropout_layer(message) # num_atoms x
            hidden

        nei_a_message = index_select_ND(message, a2a) # num_atoms x
            max_num_bonds x hidden #TODO: why a2a not a2b
        a_message = nei_a_message.sum(dim=1) # num_atoms x hidden
        a_input = torch.cat([atom_features, a_message], dim=1) #
            num_atoms x (atom_fdim + hidden)
        atom_hiddens = self.act_func(self.W_o(a_input)) # num_atoms x
            hidden
        atom_hiddens = self.dropout_layer(atom_hiddens) # num_atoms x
            hidden
    else:
        atom_hiddens = self.dropout_layer(message)

# Readout
vecs = []
for i, (a_start, a_size) in enumerate(a_scope):
    if a_size == 0:
        vecs.append(self.cached_zero_vector)
    else:
        cur_hiddens = atom_hiddens.narrow(0, a_start, a_size)
        mol_vec = cur_hiddens # (num_atoms, hidden_size)

        mol_vec = mol_vec.sum(dim=0) / a_size
        vecs.append(mol_vec)

vecs = torch.stack(vecs, dim=0) # (num_samples, hidden_size)
middle_layer = vecs.detach().data.cpu().numpy()

if self.use_input_features:
    features_batch = features_batch.to(vecs)
    if len(features_batch.shape) == 1:
        features_batch = features_batch.view([1, features_batch.
            shape[0]])
    vecs = torch.cat([vecs, features_batch], dim=1) # (
        num_samples, hidden_size)

```

```

        return vecs, num_not_zero_diff, middle_layer # num_samples x
            hidden

class MPN(nn.Module):
    """A message passing neural network for encoding a molecule."""

    def __init__(self,
                 args: Namespace,
                 atom_fdim: int = None,
                 bond_fdim: int = None,
                 graph_input: bool = False,
                 return_atom_hiddens: bool = False):
        """
        Initializes the MPN.

        :param args: Arguments.
        :param atom_fdim: Atom features dimension.
        :param bond_fdim: Bond features dimension.
        :param graph_input: If true, expects BatchMolGraph as input.
            Otherwise expects a list of smiles strings as input.
        :param return_atom_hiddens: Return hidden atom feature vectors
            instead of mol vector.
        """
        super(MPN, self).__init__()
        self.args = args
        self.atom_fdim = atom_fdim or get_atom_fdim(args)
        self.bond_fdim = bond_fdim or get_bond_fdim(args) + (not args.
            atom_messages) * self.atom_fdim
        self.graph_input = graph_input
        self.return_atom_hiddens = return_atom_hiddens
        self.encoder = MPNEncoder(self.args, self.atom_fdim, self.
            bond_fdim, self.return_atom_hiddens)

    def forward(self,
                batch: Union[List[str], BatchMolGraph],
                features_batch: List[np.ndarray] = None) -> torch.
                    FloatTensor:
        """
        Encodes a batch of molecular SMILES strings.

        :param batch: A list of SMILES strings or a BatchMolGraph (if self
            .graph_input is True).
        :param features_batch: A list of ndarrays containing additional
            features.
        :return: A PyTorch tensor of shape (num_molecules, hidden_size)
            containing the encoding of each molecule.
        """
        if not self.graph_input: # if features only, batch won't even be
            used
            batch = mol2graph(batch, self.args)

        output = self.encoder.forward(batch, features_batch)

        return output

class MPNDiff(nn.Module):
    """A message passing neural network for encoding of custom (difference
    ) features."""

    def __init__(self,
                 args: Namespace,
                 atom_fdim: int,
                 graph_input: bool = False):
        """Initializes the MPNDiff.

```

```

        :param args: Arguments.
        :param atom_fdim: Atom features dimension.
        :param graph_input: If true, expects BatchMolGraph as input.
            Otherwise expects a list of smiles strings as input.
        """
        super(MPNDiff, self).__init__()
        self.args = args
        self.atom_fdim = atom_fdim
        self.graph_input = graph_input
        self.encoder = MPNDiffEncoder(self.args, self.atom_fdim)

    def forward(self,
                atom_features: torch.FloatTensor,
                batch: Union[List[str], BatchMolGraph],
                features_batch: List[np.ndarray] = None) -> torch.
                FloatTensor:
        """
        Encodes a batch of molecular SMILES strings with custom features.

        :param atom_features: Atom features for the batch.
        :param batch: A list of SMILES strings or a BatchMolGraph (if self
            .graph_input is True).
        :param features_batch: A list of ndarrays containing additional
            features.
        :return: A PyTorch tensor of shape (num_molecules, hidden_size)
            containing the encoding of each molecule.
        """
        if not self.graph_input: # if features only, batch won't even be
            used
            batch = mol2graph(batch, self.args)

        output = self.encoder.forward(atom_features, batch, features_batch
        )

        return output

```

ii Підготовка даних

```

from argparse import Namespace
import random
from typing import Callable, List, Tuple, Union

import numpy as np
from torch.utils.data.dataset import Dataset as TorchDataset
from rdkit import Chem

from .scaler import StandardScaler
from chemprop.features import get_features_generator
from chemprop.mol_utils import str_to_mol

class Datapoint:
    """Base class for other Datapoint classes."""

    def set_features(self, features: np.ndarray):
        """
        Sets the features.

        :param features: A 1-D numpy array of features.
        """
        self.features = features

    def num_tasks(self) -> int:
        """
        Returns the number of prediction tasks.

```

```

        :return: The number of tasks.
        """
        return len(self.targets)

def set_targets(self, targets: List[float]):
    """
    Sets the targets.

    :param targets: A list of floats containing the targets.
    """
    self.targets = targets

class MoleculeDatapoint(Datapoint):
    """A MoleculeDatapoint contains a single molecule and its associated
    features and targets."""

    def __init__(self,
                 line: List[str],
                 args: Namespace = None,
                 features: np.ndarray = None,
                 use_compound_names: bool = False):
        """
        Initializes a MoleculeDatapoint, which contains a single molecule.

        :param line: A list of strings generated by separating a line in a
            data CSV file by comma.
        :param args: Arguments.
        :param features: A numpy array containing additional features (ex.
            Morgan fingerprint).
        :param use_compound_names: Whether the data CSV includes the
            compound name on each line.
        """
        if args is not None:
            self.features_generator = args.features_generator
            self.args = args
        else:
            self.features_generator = self.args = None

        if features is not None and self.features_generator is not None:
            raise ValueError('Currently cannot provide both loaded
                features and a features generator.')

        self.features = features

        if use_compound_names:
            self.compound_name = line[0] # str
            line = line[1:]
        else:
            self.compound_name = None

        self.smiles = line[0] # str
        self.mol = str_to_mol(self.smiles, explicit_hydrogens=args.
            explicit_hydrogens if args is not None else False)

        # Generate additional features if given a generator
        if self.features_generator is not None:
            self.features = []

            for fg in self.features_generator:
                features_generator = get_features_generator(fg)
                if self.mol is not None and self.mol.GetNumHeavyAtoms() >
                    0:
                    self.features.extend(features_generator(self.mol))

            self.features = np.array(self.features)

```

```

# Fix nans in features
if self.features is not None:
    replace_token = 0
    self.features = np.where(np.isnan(self.features),
                             replace_token, self.features)

# Create targets
self.targets = [float(x) if x != '' else None for x in line[1:]]

class ReactionDatapoint(Datapoint):
    """A ReactionDatapoint contains a single reaction (two molecules) and
    its associated features and targets."""

    def __init__(self,
                 line: List[str],
                 args: Namespace = None,
                 features: np.ndarray = None,
                 use_compound_names: bool = False):
        """
        Initializes a ReactionDatapoint, which contains a single reaction.

        :param line: A list of strings generated by separating a line in a
            data CSV file by comma.
        :param args: Arguments.
        :param features: A numpy array containing additional features (ex.
            Morgan fingerprint).
        :param use_compound_names: Whether the data CSV includes the
            compound names on each line.
        """
        if args is not None:
            self.features_generator = args.features_generator
            self.args = args
        else:
            self.features_generator = self.args = None

        if features is not None and self.features_generator is not None:
            raise ValueError('Currently cannot provide both loaded
                features and a features generator.')

        self.features = features

        if use_compound_names:
            self.compound_name = line[0] + '>' + line[1] # str
            line = line[2:]
        else:
            self.compound_name = None

        self.rsmiles = line[0] # str
        self.psmiles = line[1] # str
        self.rmol = str_to_mol(self.rsmiles, explicit_hydrogens=args.
            explicit_hydrogens if args is not None else False)
        self.pmol = str_to_mol(self.psmiles, explicit_hydrogens=args.
            explicit_hydrogens if args is not None else False)

        # Generate additional features if given a generator
        if self.features_generator is not None:
            self.features = []

            for fg in self.features_generator:
                features_generator = get_features_generator(fg)
                if self.rmol is not None and self.pmol is not None:
                    # Use difference features
                    diff_feat = np.asarray(features_generator(self.pmol))
                    - np.asarray(features_generator(self.rmol))
                    self.features.extend(diff_feat)

```

```

        self.features = np.array(self.features)

        # Fix nans in features
        if self.features is not None:
            replace_token = 0
            self.features = np.where(np.isnan(self.features),
                                    replace_token, self.features)

        # Create targets
        self.targets = [float(x) if x != '' else None for x in line[2:]]

class Dataset(TorchDataset):
    """Base class for other Dataset classes."""

    def __init__(self, data: List[Datapoint]):
        """
        Initializes a Dataset, which contains a list of Datapoints (e.g. a
        list of molecules or reactions).

        :param data: A list of Datapoints.
        """
        self.data = data
        self.args = self.data[0].args if len(self.data) > 0 else None
        self.scaler = None

    def smiles(self):
        """
        Returns the smiles strings or tuples thereof associated with the
        data points.

        Should be overridden by all subclasses.
        """
        raise NotImplementedError

    def mols(self):
        """
        Returns the RDKit molecules or tuples thereof associated with the
        data points.

        Should be overridden by all subclasses.
        """
        raise NotImplementedError

    def compound_names(self) -> List[str]:
        """
        Returns the compound names associated with the dataset.

        :return: A list of compound names or None if the dataset does not
            contain compound names.
        """
        if len(self.data) == 0 or self.data[0].compound_name is None:
            return None

        return [d.compound_name for d in self.data]

    def features(self) -> List[np.ndarray]:
        """
        Returns the features associated with each data point (if they
        exist).

        :return: A list of 1D numpy arrays containing the features for
            each data point or None if there are no features.
        """
        if len(self.data) == 0 or self.data[0].features is None:
            return None

```

```

    return [d.features for d in self.data]

def targets(self) -> List[List[float]]:
    """
    Returns the targets associated with each data point.

    :return: A list of lists of floats containing the targets.
    """
    return [d.targets for d in self.data]

def num_tasks(self) -> int:
    """
    Returns the number of prediction tasks.

    :return: The number of tasks.
    """
    return self.data[0].num_tasks() if len(self.data) > 0 else None

def features_size(self) -> int:
    """
    Returns the size of the features array associated with each data
    point.

    :return: The size of the features.
    """
    return len(self.data[0].features) if len(self.data) > 0 and self.
        data[0].features is not None else None

def shuffle(self, seed: int = None):
    """
    Shuffles the dataset.

    :param seed: Optional random seed.
    """
    if seed is not None:
        random.seed(seed)
    random.shuffle(self.data)

def normalize_features(self, scaler: StandardScaler = None,
    replace_nan_token: int = 0) -> StandardScaler:
    """
    Normalizes the features of the dataset using a StandardScaler (
    subtract mean, divide by standard deviation).

    If a scaler is provided, uses that scaler to perform the
    normalization. Otherwise fits a scaler to the
    features in the dataset and then performs the normalization.

    :param scaler: A fitted StandardScaler. Used if provided.
        Otherwise a StandardScaler is fit on
        this dataset and is then used.
    :param replace_nan_token: What to replace nans with.
    :return: A fitted StandardScaler. If a scaler is provided, this is
        the same scaler. Otherwise, this is
        a scaler fit on this dataset.
    """
    if len(self.data) == 0 or self.data[0].features is None:
        return None

    if scaler is not None:
        self.scaler = scaler

    elif self.scaler is None:
        features = np.vstack([d.features for d in self.data])
        self.scaler = StandardScaler(replace_nan_token=
            replace_nan_token)
        self.scaler.fit(features)

```



```

    for d in self.data:
        d.set_features(self.scaler.transform(d.features.reshape(1, -1)
            )[0])

    return self.scaler

def set_targets(self, targets: List[List[float]]):
    """
    Sets the targets for each item in the dataset. Assumes the targets
    are aligned with the datapoints.

    :param targets: A list of lists of floats containing targets for
        each data point. This must be the
        same length as the underlying dataset.
    """
    assert len(self.data) == len(targets)
    for i in range(len(self.data)):
        self.data[i].set_targets(targets[i])

def sort(self, key: Callable):
    """
    Sorts the dataset using the provided key.

    :param key: A function on a Datapoint to determine the sorting
        order.
    """
    self.data.sort(key=key)

def __len__(self) -> int:
    """
    Returns the length of the dataset (e.g. the number of molecules or
    reactions).

    :return: The length of the dataset.
    """
    return len(self.data)

def __getitem__(self, item) -> Union[Datapoint, List[Datapoint]]:
    """
    Gets one or more Datapoints via an index or slice.

    :param item: An index (int) or a slice object.
    :return: A Datapoint if an int is provided or a list of Datapoints
        if a slice is provided.
    """
    return self.data[item]

class MoleculeDataset(Dataset):
    """A MoleculeDataset contains a list of molecules and their associated
    features and targets."""

    def __init__(self, data: List[MoleculeDatapoint]):
        """
        Initializes a MoleculeDataset, which contains a list of
        MoleculeDatapoints (i.e. a list of molecules).

        :param data: A list of MoleculeDatapoints.
        """
        super(MoleculeDataset, self).__init__(data)

    def smiles(self) -> List[str]:
        """
        Returns the smiles strings associated with the molecules.

        :return: A list of smiles strings.

```

```

        """
        return [d.smiles for d in self.data]

def mols(self) -> List[Chem.Mol]:
    """
    Returns the RDKit molecules associated with the molecules.

    :return: A list of RDKit Mols.
    """
    return [d.mol for d in self.data]

class ReactionDataset(Dataset):
    """A ReactionDataset contains a list of reactions and their associated
    features and targets."""
def __init__(self, data: List[ReactionDatapoint]):
    """
    Initializes a ReactionDataset, which contains a list of
    ReactionDatapoints (i.e. a list of reactions).

    :param data: A list of ReactionDatapoints.
    """
    super(ReactionDataset, self).__init__(data)

def smiles(self) -> List[Tuple[str, str]]:
    """
    Returns the tuples of smiles strings associated with the reactions

    :return: A list of tuples of smiles strings.
    """
    return [(d.rsmiles, d.psmiles) for d in self.data]

def mols(self) -> List[Tuple[Chem.Mol, Chem.Mol]]:
    """
    Returns the tuples of RDKit molecules associated with the
    reactions.

    :return: A list of RDKit Mols.
    """
    return [(d.rmol, d.pmol) for d in self.data]

from typing import Any, List

import numpy as np

class StandardScaler:
    """A StandardScaler normalizes a dataset.

    When fit on a dataset, the StandardScaler learns the mean and standard
    deviation across the 0th axis.
    When transforming a dataset, the StandardScaler subtracts the means
    and divides by the standard deviations.
    """
def __init__(self, means: np.ndarray = None, stds: np.ndarray = None,
replace_nan_token: Any = None):
    """
    Initialize StandardScaler, optionally with means and standard
    deviations precomputed.

    :param means: An optional 1D numpy array of precomputed means.
    :param stds: An optional 1D numpy array of precomputed standard
    deviations.
    :param replace_nan_token: The token to use in place of nans.
    """

```

```

self.means = means
self.stds = stds
self.replace_nan_token = replace_nan_token

def fit(self, X: List[List[float]]) -> 'StandardScaler':
    """
    Learns means and standard deviations across the 0th axis.

    :param X: A list of lists of floats.
    :return: The fitted StandardScaler.
    """
    X = np.array(X).astype(float)
    self.means = np.nanmean(X, axis=0)
    self.stds = np.nanstd(X, axis=0)
    self.means = np.where(np.isnan(self.means), np.zeros(self.means.
        shape), self.means)
    self.stds = np.where(np.isnan(self.stds), np.ones(self.stds.shape)
        , self.stds)
    self.stds = np.where(self.stds == 0, np.ones(self.stds.shape),
        self.stds)

    return self

def transform(self, X: List[List[float]]):
    """
    Transforms the data by subtracting the means and dividing by the
    standard deviations.

    :param X: A list of lists of floats.
    :return: The transformed data.
    """
    X = np.array(X).astype(float)
    transformed_with_nan = (X - self.means) / self.stds
    transformed_with_none = np.where(np.isnan(transformed_with_nan),
        self.replace_nan_token, transformed_with_nan)

    return transformed_with_none

def inverse_transform(self, X: List[List[float]]):
    """
    Performs the inverse transformation by multiplying by the standard
    deviations and adding the means.

    :param X: A list of lists of floats.
    :return: The inverse transformed data.
    """
    X = np.array(X).astype(float)
    transformed_with_nan = X * self.stds + self.means
    transformed_with_none = np.where(np.isnan(transformed_with_nan),
        self.replace_nan_token, transformed_with_nan)

    return transformed_with_none

```

iii Інженерія ознак

```

from argparse import Namespace
from typing import List, Tuple, Union

from rdkit import Chem
import torch

from chemprop.mol_utils import str_to_mol

# Atom feature sizes
MAX_ATOMIC_NUM = 100
ATOM_FEATURES = {

```

```

'atomic_num': list(range(MAX_ATOMIC_NUM)),
'degree': [0, 1, 2, 3, 4, 5],
'formal_charge': [-1, -2, 1, 2, 0],
'chiral_tag': [0, 1, 2, 3],
'num_Hs': [0, 1, 2, 3, 4],
'hybridization': [
    Chem.rdchem.HybridizationType.SP,
    Chem.rdchem.HybridizationType.SP2,
    Chem.rdchem.HybridizationType.SP3,
    Chem.rdchem.HybridizationType.SP3D,
    Chem.rdchem.HybridizationType.SP3D2
],
}

# Distance feature sizes
PATH_DISTANCE_BINS = list(range(10))
THREE_D_DISTANCE_MAX = 20
THREE_D_DISTANCE_STEP = 1
THREE_D_DISTANCE_BINS = list(range(0, THREE_D_DISTANCE_MAX + 1,
    THREE_D_DISTANCE_STEP))

# len(choices) + 1 to include room for uncommon values; + 2 for IsAromatic
# and mass; +8 for ring membership
ATOM_FDIM = sum(len(choices) + 1 for choices in ATOM_FEATURES.values()) +
    2 + 8
BOND_FDIM = 14 + 8

# Memoization
SMILES_TO_GRAPH = {}

def clear_cache():
    """Clears featurization cache."""
    global SMILES_TO_GRAPH
    SMILES_TO_GRAPH = {}

def get_atom_fdim(args: Namespace) -> int:
    """
    Gets the dimensionality of atom features.

    :param: Arguments.
    """
    return ATOM_FDIM

def get_bond_fdim(args: Namespace) -> int:
    """
    Gets the dimensionality of bond features.

    :param: Arguments.
    """
    return BOND_FDIM

def onek_encoding_unk(value: int, choices: List[int]) -> List[int]:
    """
    Creates a one-hot encoding.

    :param value: The value for which the encoding should be one.
    :param choices: A list of possible values.
    :return: A one-hot encoding of the value in a list of length len(
        choices) + 1.
    If value is not in the list of choices, then the final element in the
    encoding is 1.
    """
    encoding = [0] * (len(choices) + 1)

```

```

index = choices.index(value) if value in choices else -1
encoding[index] = 1

return encoding

def atom_features(atom: Chem.rdchem.Atom, functional_groups: List[int] =
None) -> List[Union[bool, int, float]]:
    """
    Builds a feature vector for an atom.

    :param atom: An RDKit atom.
    :param functional_groups: A k-hot vector indicating the functional
        groups the atom belongs to.
    :return: A list containing the atom features.
    """
    features = onek_encoding_unk(atom.GetAtomicNum() - 1, ATOM_FEATURES['
        atomic_num']) + \
        onek_encoding_unk(atom.GetTotalDegree(), ATOM_FEATURES['degree'
        ]) + \
        onek_encoding_unk(atom.GetFormalCharge(), ATOM_FEATURES['
        formal_charge']) + \
        onek_encoding_unk(int(atom.GetChiralTag()), ATOM_FEATURES['
        chiral_tag']) + \
        onek_encoding_unk(int(atom.GetTotalNumHs()), ATOM_FEATURES['
        num_Hs']) + \
        onek_encoding_unk(int(atom.GetHybridization()), ATOM_FEATURES['
        hybridization']) + \
        [1 if atom.GetIsAromatic() else 0] + \
        [atom.GetMass() * 0.01] # scaled to about the same range as
        other features
    features += [
        atom.IsInRingSize(3),
        atom.IsInRingSize(4),
        atom.IsInRingSize(5),
        atom.IsInRingSize(6),
        atom.IsInRingSize(7),
        atom.IsInRingSize(8),
        atom.IsInRingSize(9),
        atom.IsInRingSize(10),
    ]
    if functional_groups is not None:
        features += functional_groups
    return features

def bond_features(bond: Chem.rdchem.Bond) -> List[Union[bool, int, float
]]:
    """
    Builds a feature vector for a bond.

    :param bond: A RDKit bond.
    :return: A list containing the bond features.
    """
    if bond is None:
        fbond = [1] + [0] * (BOND_FDIM - 1)
    else:
        bt = bond.GetBondType()
        fbond = [
            0, # bond is not None
            bt == Chem.rdchem.BondType.SINGLE,
            bt == Chem.rdchem.BondType.DOUBLE,
            bt == Chem.rdchem.BondType.TRIPLE,
            bt == Chem.rdchem.BondType.AROMATIC,
            (bond.GetIsConjugated() if bt is not None else 0),
            (bond.IsInRing() if bt is not None else 0),
            (bond.IsInRingSize(3) if bt is not None else 0),

```

```

        (bond.IsInRingSize(4) if bt is not None else 0),
        (bond.IsInRingSize(5) if bt is not None else 0),
        (bond.IsInRingSize(6) if bt is not None else 0),
        (bond.IsInRingSize(7) if bt is not None else 0),
        (bond.IsInRingSize(8) if bt is not None else 0),
        (bond.IsInRingSize(9) if bt is not None else 0),
        (bond.IsInRingSize(10) if bt is not None else 0),
    ]
    fbond += onek_encoding_unk(int(bond.GetStereo()), list(range(6)))
return fbond

```

```

class MolGraph:
    """
    A MolGraph represents the graph structure and featurization of a
    single molecule.

    A MolGraph computes the following attributes:
    - smiles: Smiles string.
    - n_atoms: The number of atoms in the molecule.
    - n_bonds: The number of bonds in the molecule.
    - f_atoms: A mapping from an atom index to a list atom features.
    - f_bonds: A mapping from a bond index to a list of bond features.
    - a2b: A mapping from an atom index to a list of incoming bond indices
    - b2a: A mapping from a bond index to the index of the atom the bond
    originates from.
    - b2revb: A mapping from a bond index to the index of the reverse bond
    """

    def __init__(self, smiles: str, args: Namespace):
        """
        Computes the graph structure and featurization of a molecule.

        :param smiles: A smiles string.
        :param args: Arguments.
        """
        self.smiles = smiles
        self.n_atoms = 0 # number of atoms
        self.n_bonds = 0 # number of bonds
        self.f_atoms = [] # mapping from atom index to atom features
        self.f_bonds = [] # mapping from bond index to concat(in_atom,
            bond) features
        self.a2b = [] # mapping from atom index to incoming bond indices
        self.b2a = [] # mapping from bond index to the index of the atom
            the bond is coming from
        self.b2revb = [] # mapping from bond index to the index of the
            reverse bond

        # Convert smiles to molecule
        mol = str_to_mol(smiles, explicit_hydrogens=args.
            explicit_hydrogens)

        # fake the number of "atoms" if we are collapsing substructures
        self.n_atoms = mol.GetNumAtoms()

        # Require atom map numbers when using reactions
        if args.reaction:
            if any(a.GetAtomMapNum() == 0 for a in mol.GetAtoms()):
                raise Exception(f'{smiles} is missing atom map numbers')

            # Ensure that atoms in reactant and product are sorted in the
            same way
            atoms = sorted(mol.GetAtoms(), key=lambda a: a.GetAtomMapNum()
                )
        else:

```

```

        atoms = mol.GetAtoms()

        # Get atom features
        for i, atom in enumerate(atoms):
            self.f_atoms.append(atom_features(atom))
        self.f_atoms = [self.f_atoms[i] for i in range(self.n_atoms)]

        for _ in range(self.n_atoms):
            self.a2b.append([])

        # Get bond features
        for a1 in range(self.n_atoms):
            for a2 in range(a1 + 1, self.n_atoms):
                rdkit_idx1 = atoms[a1].GetIdx()
                rdkit_idx2 = atoms[a2].GetIdx()
                bond = mol.GetBondBetweenAtoms(rdkit_idx1, rdkit_idx2)

                if bond is None:
                    continue

                f_bond = bond_features(bond)

                if args.atom_messages:
                    self.f_bonds.append(f_bond)
                    self.f_bonds.append(f_bond)
                else:
                    self.f_bonds.append(self.f_atoms[a1] + f_bond)
                    self.f_bonds.append(self.f_atoms[a2] + f_bond)

                # Update index mappings
                b1 = self.n_bonds
                b2 = b1 + 1
                self.a2b[a2].append(b1) # b1 = a1 --> a2
                self.b2a.append(a1)
                self.a2b[a1].append(b2) # b2 = a2 --> a1
                self.b2a.append(a2)
                self.b2revb.append(b2)
                self.b2revb.append(b1)
                self.n_bonds += 2

```

```
class BatchMolGraph:
```

```

    """
    A BatchMolGraph represents the graph structure and featurization of a
    batch of molecules.

    A BatchMolGraph contains the attributes of a MolGraph plus:
    - smiles_batch: A list of smiles strings.
    - n_mols: The number of molecules in the batch.
    - atom_fdim: The dimensionality of the atom features.
    - bond_fdim: The dimensionality of the bond features (technically the
    combined atom/bond features).
    - a_scope: A list of tuples indicating the start and end atom indices
    for each molecule.
    - b_scope: A list of tuples indicating the start and end bond indices
    for each molecule.
    - max_num_bonds: The maximum number of bonds neighboring an atom in
    this batch.
    - b2b: (Optional) A mapping from a bond index to incoming bond indices
    .
    - a2a: (Optional): A mapping from an atom index to neighboring atom
    indices.
    """

```

```

def __init__(self, mol_graphs: List[MolGraph], args: Namespace):
    self.smiles_batch = [mol_graph.smiles for mol_graph in mol_graphs]
    self.n_mols = len(self.smiles_batch)

```

```

self.atom_fdim = get_atom_fdim(args)
self.bond_fdim = get_bond_fdim(args) + (not args.atom_messages) *
    self.atom_fdim

# Start n_atoms and n_bonds at 1 b/c zero padding
self.n_atoms = 1 # number of atoms (start at 1 b/c need index 0
    as padding)
self.n_bonds = 1 # number of bonds (start at 1 b/c need index 0
    as padding)
self.a_scope = [] # list of tuples indicating (start_atom_index,
    num_atoms) for each molecule
self.b_scope = [] # list of tuples indicating (start_bond_index,
    num_bonds) for each molecule

# All start with zero padding so that indexing with zero padding
    returns zeros
f_atoms = [[0] * self.atom_fdim] # atom features
f_bonds = [[0] * self.bond_fdim] # combined atom/bond features
a2b = [[]] # mapping from atom index to incoming bond indices
b2a = [0] # mapping from bond index to the index of the atom the
    bond is coming from
b2revb = [0] # mapping from bond index to the index of the
    reverse bond
for mol_graph in mol_graphs:
    f_atoms.extend(mol_graph.f_atoms)
    f_bonds.extend(mol_graph.f_bonds)

    for a in range(mol_graph.n_atoms):
        a2b.append([b + self.n_bonds for b in mol_graph.a2b[a]])

    for b in range(mol_graph.n_bonds):
        b2a.append(self.n_atoms + mol_graph.b2a[b])
        b2revb.append(self.n_bonds + mol_graph.b2revb[b])

    self.a_scope.append((self.n_atoms, mol_graph.n_atoms))
    self.b_scope.append((self.n_bonds, mol_graph.n_bonds))
    self.n_atoms += mol_graph.n_atoms
    self.n_bonds += mol_graph.n_bonds

self.max_num_bonds = max(1, max(len(in_bonds) for in_bonds in a2b)
    ) # max with 1 to fix a crash in rare case of all single-heavy-
    atom mols

self.f_atoms = torch.FloatTensor(f_atoms)
self.f_bonds = torch.FloatTensor(f_bonds)
self.a2b = torch.LongTensor([a2b[a] + [0] * (self.max_num_bonds -
    len(a2b[a])) for a in range(self.n_atoms)])
self.b2a = torch.LongTensor(b2a)
self.b2revb = torch.LongTensor(b2revb)
self.b2b = None # try to avoid computing b2b b/c O(n_atoms^3)
self.a2a = None # only needed if using atom messages

def get_components(self) -> Tuple[torch.FloatTensor, torch.FloatTensor
,
    torch.LongTensor, torch.LongTensor,
    torch.LongTensor,
    List[Tuple[int, int]], List[Tuple[
        int, int]]]:
    """
    Returns the components of the BatchMolGraph.

    :return: A tuple containing PyTorch tensors with the atom features
        , bond features, and graph structure
    and two lists indicating the scope of the atoms and bonds (i.e.
        which molecules they belong to).
    """

```



```

        return self.f_atoms, self.f_bonds, self.a2b, self.b2a, self.b2revb
            , self.a_scope, self.b_scope

def get_b2b(self) -> torch.LongTensor:
    """
    Computes (if necessary) and returns a mapping from each bond index
        to all the incoming bond indices.

    :return: A PyTorch tensor containing the mapping from each bond
        index to all the incoming bond indices.
    """

    if self.b2b is None:
        b2b = self.a2b[self.b2a] # num_bonds x max_num_bonds
        # b2b includes reverse edge for each bond so need to mask out
        revmask = (b2b != self.b2revb.unsqueeze(1).repeat(1, b2b.size
            (1))).long() # num_bonds x max_num_bonds
        self.b2b = b2b * revmask

    return self.b2b

def get_a2a(self) -> torch.LongTensor:
    """
    Computes (if necessary) and returns a mapping from each atom index
        to all neighboring atom indices.

    :return: A PyTorch tensor containing the mapping from each bond
        index to all the incoming bond indices.
    """

    if self.a2a is None:
        # b = a1 --> a2
        # a2b maps a2 to all incoming bonds b
        # b2a maps each bond b to the atom it comes from a1
        # thus b2a[a2b] maps atom a2 to neighboring atoms a1
        self.a2a = self.b2a[self.a2b] # num_atoms x max_num_bonds

    return self.a2a

def mol2graph(smiles_batch: List[str],
              args: Namespace) -> BatchMolGraph:
    """
    Converts a list of SMILES strings to a BatchMolGraph containing the
        batch of molecular graphs.

    :param smiles_batch: A list of SMILES strings.
    :param args: Arguments.
    :return: A BatchMolGraph containing the combined molecular graph for
        the molecules
    """
    mol_graphs = []
    for smiles in smiles_batch:
        if smiles in SMILES_TO_GRAPH:
            mol_graph = SMILES_TO_GRAPH[smiles]
        else:
            mol_graph = MolGraph(smiles, args)
            if not args.no_cache:
                SMILES_TO_GRAPH[smiles] = mol_graph
            mol_graphs.append(mol_graph)

    return BatchMolGraph(mol_graphs, args)

from typing import Callable, List, Union

import numpy as np
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem

```

```

from chemprop.mol_utils import str_to_mol

Molecule = Union[str, Chem.Mol]
FeaturesGenerator = Callable[[Molecule], np.ndarray]

FEATURES_GENERATOR_REGISTRY = {}

def register_features_generator(features_generator_name: str) -> Callable
[[FeaturesGenerator], FeaturesGenerator]:
    """
    Registers a features generator.

    :param features_generator_name: The name to call the FeaturesGenerator
    :return: A decorator which will add a FeaturesGenerator to the
    registry using the specified name.
    """
    def decorator(features_generator: FeaturesGenerator) ->
    FeaturesGenerator:
        FEATURES_GENERATOR_REGISTRY[features_generator_name] =
        features_generator
        return features_generator

    return decorator

def get_features_generator(features_generator_name: str) ->
FeaturesGenerator:
    """
    Gets a registered FeaturesGenerator by name.

    :param features_generator_name: The name of the FeaturesGenerator.
    :return: The desired FeaturesGenerator.
    """
    if features_generator_name not in FEATURES_GENERATOR_REGISTRY:
        raise ValueError(f'Features generator "{features_generator_name}"
        could not be found. '
        f'If this generator relies on rdkit features, you
        may need to install descriptastorus.')

    return FEATURES_GENERATOR_REGISTRY[features_generator_name]

def get_available_features_generators() -> List[str]:
    """Returns the names of available features generators."""
    return list(FEATURES_GENERATOR_REGISTRY.keys())

MORGAN_RADIUS = 2
MORGAN_NUM_BITS = 2048

@register_features_generator('morgan')
def morgan_binary_features_generator(mol: Molecule,
    radius: int = MORGAN_RADIUS,
    num_bits: int = MORGAN_NUM_BITS) ->
    np.ndarray:
    """
    Generates a binary Morgan fingerprint for a molecule.

    :param mol: A molecule (i.e. either a SMILES string or an RDKit
    molecule).
    :param radius: Morgan fingerprint radius.

```

```

:param num_bits: Number of bits in Morgan fingerprint.
:return: A 1-D numpy array containing the binary Morgan fingerprint.
"""
mol = str_to_mol(mol) if type(mol) == str else mol
features_vec = AllChem.GetMorganFingerprintAsBitVect(mol, radius,
    nBits=num_bits)
features = np.zeros((1,))
DataStructs.ConvertToNumpyArray(features_vec, features)

return features

@register_features_generator('morgan_count')
def morgan_counts_features_generator(mol: Molecule,
    radius: int = MORGAN_RADIUS,
    num_bits: int = MORGAN_NUM_BITS) ->
    np.ndarray:
    """
    Generates a counts-based Morgan fingerprint for a molecule.

    :param mol: A molecule (i.e. either a SMILES string or an RDKit
        molecule).
    :param radius: Morgan fingerprint radius.
    :param num_bits: Number of bits in Morgan fingerprint.
    :return: A 1D numpy array containing the counts-based Morgan
        fingerprint.
    """
    mol = str_to_mol(mol) if type(mol) == str else mol
    features_vec = AllChem.GetHashedMorganFingerprint(mol, radius, nBits=
        num_bits)
    features = np.zeros((1,))
    DataStructs.ConvertToNumpyArray(features_vec, features)

    return features

try:
    from descriptastorus.descriptors import rdDescriptors,
        rdNormalizedDescriptors

@register_features_generator('rdkit_2d')
def rdkit_2d_features_generator(mol: Molecule) -> np.ndarray:
    """
    Generates RDKit 2D features for a molecule.

    :param mol: A molecule (i.e. either a SMILES string or an RDKit
        molecule).
    :return: A 1D numpy array containing the RDKit 2D features.
    """
    smiles = Chem.MolToSmiles(mol, isomericSmiles=True) if type(mol)
        != str else mol
    generator = rdDescriptors.RDKit2D()
    features = generator.process(smiles)[1:]

    return features

@register_features_generator('rdkit_2d_normalized')
def rdkit_2d_features_generator(mol: Molecule) -> np.ndarray:
    """
    Generates RDKit 2D normalized features for a molecule.

    :param mol: A molecule (i.e. either a SMILES string or an RDKit
        molecule).
    :return: A 1D numpy array containing the RDKit 2D normalized
        features.
    """
    smiles = Chem.MolToSmiles(mol, isomericSmiles=True) if type(mol)

```

```

        != str else mol
        generator = rdNormalizedDescriptors.RDKit2DNormalized()
        features = generator.process(smiles)[1:]

    return features
except ImportError:
    pass

"""
Custom features generator template.

Note: The name you use to register the features generator is the name
you will specify on the command line when using the --features_generator <
name> flag.
Ex. python train.py ... --features_generator custom ...

@register_features_generator('custom')
def custom_features_generator(mol: Molecule) -> np.ndarray:
    # If you want to use the SMILES string
    smiles = Chem.MolToSmiles(mol, isomericSmiles=True) if type(mol) !=
        str else mol

    # If you want to use the RDKit molecule
    mol = Chem.MolFromSmiles(mol) if type(mol) == str else mol

    # Replace this with code which generates features from the molecule
    features = np.array([0, 0, 1])

    return features
"""

```

iv Алгоритм тренування графової нейронної мережі

```

from argparse import Namespace
import logging
from typing import Callable, List, Union

from tensorboardX import SummaryWriter
import torch
import torch.nn as nn
from torch.optim import Optimizer
from torch.optim.lr_scheduler import _LRScheduler
from tqdm import trange

from chemprop.data import MoleculeDataset, ReactionDataset
from chemprop.nn_utils import compute_gnorm, compute_pnorm, NoamLR

def train(model: nn.Module,
          data: Union[MoleculeDataset, List[MoleculeDataset],
                     ReactionDataset, List[ReactionDataset]],
          loss_func: Callable,
          optimizer: Optimizer,
          scheduler: _LRScheduler,
          args: Namespace,
          n_iter: int = 0,
          logger: logging.Logger = None,
          writer: SummaryWriter = None) -> int:
    """
    Trains a model for an epoch.

    :param model: Model.
    :param data: A MoleculeDataset/ReactionDataset (or a list of
        MoleculeDatasets/ReactionDatasets if using moe).
    :param loss_func: Loss function.
    """

```

```

:param optimizer: An Optimizer.
:param scheduler: A learning rate scheduler.
:param args: Arguments.
:param n_iter: The number of iterations (training examples) trained on
    so far.
:param logger: A logger for printing intermediate results.
:param writer: A tensorboardX SummaryWriter.
:return: The total number of iterations (training examples) trained on
    so far.
"""
debug = logger.debug if logger is not None else print

model.train()

data.shuffle()

loss_sum, iter_count = 0, 0

num_iters = len(data) // args.batch_size * args.batch_size # don't
    use the last batch if it's small, for stability

iter_size = args.batch_size

for i in trange(0, num_iters, iter_size):
    # Prepare batch
    if i + args.batch_size > len(data):
        break
    data_batch = data[i:i + args.batch_size]
    mol_batch = ReactionDataset(data_batch) if args.reaction else
        MoleculeDataset(data_batch)
    smiles_batch, features_batch, target_batch = mol_batch.smiles(),
        mol_batch.features(), mol_batch.targets()
    batch = smiles_batch
    mask = torch.Tensor([[x is not None for x in tb] for tb in
        target_batch])
    targets = torch.Tensor([[0 if x is None else x for x in tb] for tb
        in target_batch])

    if next(model.parameters()).is_cuda:
        mask, targets = mask.cuda(), targets.cuda()

    class_weights = torch.ones(targets.shape)

    if args.cuda:
        class_weights = class_weights.cuda()

    # Run model
    model.zero_grad()
    if args.reaction:
        rbatch, pbatch = list(zip(*batch))
        preds, _, _, _ = model(rbatch, pbatch, features_batch)
    else:
        preds = model(batch, features_batch)

    if args.dataset_type == 'multiclass':
        targets = targets.long()
        loss = torch.cat([loss_func(preds[:, target_index, :], targets
           [:, target_index]).unsqueeze(1) for target_index in range(
                preds.size(1))], dim=1) * class_weights * mask
    else:
        loss = loss_func(preds, targets) * class_weights * mask

    loss = loss.sum() / mask.sum()

    loss_sum += loss.item()
    iter_count += len(mol_batch)

```

```

    loss.backward()
    optimizer.step()

    if isinstance(scheduler, NoamLR):
        scheduler.step()

    n_iter += len(mol_batch)

    # Log and/or add to tensorboard
    if (n_iter // args.batch_size) % args.log_frequency == 0:
        lrs = scheduler.get_lr()
        pnorm = compute_pnorm(model)
        gnorm = compute_gnorm(model)
        loss_avg = loss_sum / args.log_frequency
        loss_sum, iter_count = 0, 0

        lrs_str = ', '.join(f'lr_{i} = {lr:.4e}' for i, lr in
            enumerate(lrs))
        debug(f'Loss = {loss_avg:.4e}, PNorm = {pnorm:.4f}, GNorm = {
            gnorm:.4f}, {lrs_str}')

        if writer is not None:
            writer.add_scalar('train_loss', loss_avg, n_iter)
            writer.add_scalar('param_norm', pnorm, n_iter)
            writer.add_scalar('gradient_norm', gnorm, n_iter)
            for i, lr in enumerate(lrs):
                writer.add_scalar(f'learning_rate_{i}', lr, n_iter)

    return n_iter

from argparse import Namespace
import csv
from logging import Logger
import os
from pprint import pformat
from typing import List
import time

import numpy as np
from tensorboardX import SummaryWriter
import torch
from tqdm import trange
import pickle
from torch.optim.lr_scheduler import ExponentialLR

from .evaluate import evaluate, evaluate_predictions
from .predict import predict
from .train import train
from chemprop.data import StandardScaler
from chemprop.data.utils import get_class_sizes, get_data, get_task_names,
    split_data
from chemprop.models import build_model
from chemprop.nn_utils import param_count
from chemprop.utils import build_optimizer, build_lr_scheduler,
    get_loss_func, get_metric_func, load_checkpoint, \
    makedirs, save_checkpoint

def run_training(args: Namespace, logger: Logger = None) -> List[float]:
    """
    Trains a model and returns test scores on the model checkpoint with
    the highest validation score.

    :param args: Arguments.
    :param logger: Logger.
    :return: A list of ensemble scores for each task.
    """

```

```

if logger is not None:
    debug, info = logger.debug, logger.info
else:
    debug = info = print

# Set GPU
if args.gpu is not None:
    torch.cuda.set_device(args.gpu)

# Print args
debug(pformat(vars(args)))

# Get data
debug('Loading data')
args.task_names = get_task_names(args.data_path, reaction=args.
    reaction)
data = get_data(path=args.data_path, args=args, logger=logger)
args.num_tasks = data.num_tasks()
args.features_size = data.features_size()
debug(f'Number of tasks = {args.num_tasks}')

# Split data
debug(f'Splitting data with seed {args.seed}')
if args.separate_test_path:
    test_data = get_data(path=args.separate_test_path, args=args,
        features_path=args.separate_test_features_path, logger=logger)
if args.separate_val_path:
    val_data = get_data(path=args.separate_val_path, args=args,
        features_path=args.separate_val_features_path, logger=logger)

if args.separate_val_path and args.separate_test_path:
    train_data = data
elif args.separate_val_path:
    assert args.split_sizes[1] == 0.0
    train_data, _, test_data = split_data(data=data, split_type=args.
        split_type, sizes=args.split_sizes, seed=args.seed, args=args,
        logger=logger)
elif args.separate_test_path:
    assert args.split_sizes[2] == 0.0
    train_data, val_data, _ = split_data(data=data, split_type=args.
        split_type, sizes=args.split_sizes, seed=args.seed, args=args,
        logger=logger)
else:
    train_data, val_data, test_data = split_data(data=data, split_type
        =args.split_type, sizes=args.split_sizes, seed=args.seed, args=
        args, logger=logger)

if args.dataset_type == 'classification':
    class_sizes = get_class_sizes(data)
    debug('Class sizes')
    for i, task_class_sizes in enumerate(class_sizes):
        debug(f'{args.task_names[i]} '
            f'{"", ".join(f"{cls}: {size * 100:.2f}%" for cls, size
                in enumerate(task_class_sizes))}')

if args.save_smiles_splits:
    with open(args.data_path, 'r') as f:
        reader = csv.reader(f)
        header = next(reader)

        lines_by_smiles = {}
        indices_by_smiles = {}
        for i, line in enumerate(reader):
            smiles = (line[0], line[1]) if args.reaction else line[0]
            lines_by_smiles[smiles] = line
            indices_by_smiles[smiles] = i

```

```

all_split_indices = []
all_data = [train_data, val_data, test_data]
all_data_names = ['train', 'val', 'test']
for dataset, name, split_size in zip(all_data, all_data_names,
    args.split_sizes):
    if split_size > 0.0:
        with open(os.path.join(args.save_dir, name + '_smiles.csv'
            ), 'w') as f:
            writer = csv.writer(f)
            writer.writerow(['rsmiles', 'psmiles'] if args.
                reaction else ['smiles'])
            for smiles in dataset.smiles():
                writer.writerow(smiles if args.reaction else [
                    smiles])
        with open(os.path.join(args.save_dir, name + '_full.csv'),
            'w') as f:
            writer = csv.writer(f)
            writer.writerow(header)
            for smiles in dataset.smiles():
                writer.writerow(lines_by_smiles[smiles])
        split_indices = []
        for smiles in dataset.smiles():
            split_indices.append(indices_by_smiles[smiles])
        split_indices = sorted(split_indices)
        all_split_indices.append(split_indices)
with open(os.path.join(args.save_dir, 'split_indices.pckl'), 'wb')
    as f:
    pickle.dump(all_split_indices, f)

if args.features_scaling:
    features_scaler = train_data.normalize_features(replace_nan_token
        =0)
    val_data.normalize_features(features_scaler)
    test_data.normalize_features(features_scaler)
else:
    features_scaler = None

args.train_data_size = len(train_data)

debug(f'Total size = {len(train_data) + len(val_data) + len(test_data)
    :},} | '
    f'train size = {len(train_data):},} | val size = {len(val_data)
    :},} | test size = {len(test_data):},}')

# Initialize scaler and scale training targets by subtracting mean and
    dividing standard deviation (regression only)
if args.dataset_type == 'regression':
    debug('Fitting scaler')
    train_smiles, train_targets = train_data.smiles(), train_data.
        targets()
    scaler = StandardScaler().fit(train_targets)
    scaled_targets = scaler.transform(train_targets).tolist()
    train_data.set_targets(scaled_targets)
else:
    scaler = None

# Get loss and metric functions
loss_func = get_loss_func(args)
metric_func = get_metric_func(metric=args.metric)

# Set up test set evaluation
test_smiles, test_targets = test_data.smiles(), test_data.targets()
if args.dataset_type == 'multiclass':
    sum_test_preds = np.zeros((len(test_smiles), args.num_tasks, args.
        multiclass_num_classes))
else:
    sum_test_preds = np.zeros((len(test_smiles), args.num_tasks))

```



```

# Train ensemble of models
for model_idx in range(args.ensemble_size):
    # Tensorboard writer
    save_dir = os.path.join(args.save_dir, f'model_{model_idx}')
    makedirs(save_dir)
    try:
        twriter = SummaryWriter(log_dir=f'{save_dir}/train/')
        vwriter = SummaryWriter(log_dir=f'{save_dir}/test/')
    except:
        twriter = SummaryWriter(logdir=f'{save_dir}/train/')
        vwriter = SummaryWriter(logdir=f'{save_dir}/test/')
    # Load/build model
    if args.checkpoint_paths is not None:
        debug(f'Loading model {model_idx} from {args.checkpoint_paths[
            model_idx]}')
        model = load_checkpoint(args.checkpoint_paths[model_idx],
            current_args=args, logger=logger)
    else:
        debug(f'Building model {model_idx}')
        model = build_model(args)

    debug(model)
    debug(f'Number of parameters = {param_count(model):,}')
    if args.cuda:
        debug('Moving model to cuda')
        model = model.cuda()

    # Ensure that model is saved in correct location for evaluation if
    # 0 epochs
    save_checkpoint(os.path.join(save_dir, 'model.pt'), model, scaler,
        features_scaler, args)

    # Optimizers
    optimizer = build_optimizer(model, args)

    # Learning rate schedulers
    scheduler = build_lr_scheduler(optimizer, args)

    # Run training
    best_score = float('inf') if args.minimize_score else -float('inf'
        )
    best_epoch, n_iter = 0, 0
    for epoch in trange(args.epochs):
        debug(time.ctime(time.time()))
        debug(f'Epoch {epoch}')

        n_iter = train(
            model=model,
            data=train_data,
            loss_func=loss_func,
            optimizer=optimizer,
            scheduler=scheduler,
            args=args,
            n_iter=n_iter,
            logger=logger,
            writer=twriter
        )
        if isinstance(scheduler, ExponentialLR):
            scheduler.step()
        train_scores = evaluate(
            model=model,
            data=train_data,
            num_tasks=args.num_tasks,
            metric_func=metric_func,
            batch_size=args.batch_size,
            dataset_type=args.dataset_type,

```

```

        stage='train',
        scaler=scaler,
        logger=logger
    )
    val_scores = evaluate(
        model=model,
        data=val_data,
        num_tasks=args.num_tasks,
        metric_func=metric_func,
        batch_size=args.batch_size,
        dataset_type=args.dataset_type,
        stage='val',
        scaler=scaler,
        logger=logger
    )

    avg_tr_score = np.nanmean(train_scores)
    debug(f'Train {args.metric} = {avg_tr_score:.6f}')
    twriter.add_scalar(f'{args.metric}', avg_tr_score, n_iter)

    # Average validation score
    avg_val_score = np.nanmean(val_scores)
    debug(f'Validation {args.metric} = {avg_val_score:.6f}')
    vwriter.add_scalar(f'{args.metric}', avg_val_score, n_iter)

    if args.show_individual_scores:
        # Individual validation scores
        for task_name, val_score in zip(args.task_names,
            val_scores):
            debug(f'Validation {task_name} {args.metric} = {
                val_score:.6f}')
            vwriter.add_scalar(f'validation_{task_name}_{args.
                metric}', val_score, n_iter)

    # Save model checkpoint if improved validation score
    val_score = val_scores[0] if args.validate_on_first_target
        else avg_val_score
    if args.minimize_score and val_score < best_score or \
        not args.minimize_score and val_score > best_score:
        best_score, best_epoch = avg_val_score, epoch
        save_checkpoint(os.path.join(save_dir, 'model.pt'), model,
            scaler, features_scaler, args)

    # Evaluate on test set using model with best validation score
    debug(time.ctime(time.time()))

    info(f'Model {model_idx} best validation {args.metric} = {
        best_score:.6f} on epoch {best_epoch}')
    model = load_checkpoint(os.path.join(save_dir, 'model.pt'), cuda=
        args.cuda, logger=logger)

    test_preds, _, _, _ = predict(
        model=model,
        data=test_data,
        batch_size=args.batch_size,
        scaler=scaler
    )
    # test_preds = [test_preds[i] for i in samples_indices_to_eval]
    # test_targets = [test_targets[i] for i in samples_indices_to_eval
    ]

    test_scores = evaluate_predictions(
        preds=test_preds,
        targets=test_targets,
        num_tasks=args.num_tasks,
        metric_func=metric_func,

```

```

        dataset_type=args.dataset_type,
        logger=logger
    )

    if len(test_preds) != 0:
        sum_test_preds += np.array(test_preds)

    # Average test score
    avg_test_score = np.nanmean(test_scores)
    info(f'Model {model_idx} test {args.metric} = {avg_test_score:.6f}'
        ')
    vwriter.add_scalar(f'{args.metric}_test', avg_test_score, 0)

    if args.show_individual_scores:
        # Individual test scores
        for task_name, test_score in zip(args.task_names, test_scores)
            :
            info(f'Model {model_idx} test {task_name} {args.metric} =
                {test_score:.6f}')
            vwriter.add_scalar(f'test_{task_name}_{args.metric}',
                test_score, n_iter)

    # Evaluate ensemble on test set
    avg_test_preds = (sum_test_preds / args.ensemble_size).tolist()

    ensemble_scores = evaluate_predictions(
        preds=avg_test_preds,
        targets=test_targets,
        num_tasks=args.num_tasks,
        metric_func=metric_func,
        dataset_type=args.dataset_type,
        logger=logger
    )

    # Average ensemble score
    avg_ensemble_test_score = np.nanmean(ensemble_scores)
    info(f'Ensemble test {args.metric} = {avg_ensemble_test_score:.6f}')
    vwriter.add_scalar(f'ensemble_test_{args.metric}',
        avg_ensemble_test_score, 0)

    # Individual ensemble scores
    if args.show_individual_scores:
        for task_name, ensemble_score in zip(args.task_names,
            ensemble_scores):
            info(f'Ensemble test {task_name} {args.metric} = {
                ensemble_score:.6f}')

    return ensemble_scores

```

v Алгоритм перехресної оцінки графової нейронної мережі

```

import logging
from typing import Callable, List, Union

import matplotlib.pyplot as plt

import torch.nn as nn

from .predict import predict
from chemprop.data import MoleculeDataset, ReactionDataset, StandardScaler

def evaluate_predictions(preds: List[List[float]],

```

```

        targets: List[List[float]],
        num_tasks: int,
        metric_func: Callable,
        dataset_type: str,
        logger: logging.Logger = None) -> List[float]:
    """
    Evaluates predictions using a metric function and filtering out
    invalid targets.

    :param preds: A list of lists of shape (data_size, num_tasks) with
        model predictions.
    :param targets: A list of lists of shape (data_size, num_tasks) with
        targets.
    :param num_tasks: Number of tasks.
    :param metric_func: Metric function which takes in a list of targets
        and a list of predictions.
    :param dataset_type: Dataset type.
    :param logger: Logger.
    :return: A list with the score for each task based on 'metric_func'.
    """
    info = logger.info if logger is not None else print

    if len(preds) == 0:
        return [float('nan')] * num_tasks

    # Filter out empty targets
    # valid_preds and valid_targets have shape (num_tasks, data_size)
    valid_preds = [[] for _ in range(num_tasks)]
    valid_targets = [[] for _ in range(num_tasks)]
    for i in range(num_tasks):
        for j in range(len(preds)):
            if targets[j][i] is not None: # Skip those without targets
                valid_preds[i].append(preds[j][i])
                valid_targets[i].append(targets[j][i])

    # Compute metric
    results = []
    for i in range(num_tasks):
        # # Skip if all targets or preds are identical, otherwise we'll
        # crash during classification
        if dataset_type == 'classification':
            nan = False
            if all(target == 0 for target in valid_targets[i]) or all(
                target == 1 for target in valid_targets[i]):
                nan = True
                info('Warning: Found a task with targets all 0s or all 1s'
                    )
            if all(pred == 0 for pred in valid_preds[i]) or all(pred == 1
                for pred in valid_preds[i]):
                nan = True
                info('Warning: Found a task with predictions all 0s or all
                    1s')

            if nan:
                results.append(float('nan'))
                continue

        if len(valid_targets[i]) == 0:
            continue

        if dataset_type == 'multiclass':
            results.append(metric_func(valid_targets[i], valid_preds[i]))
        else:
            results.append(metric_func(valid_targets[i], valid_preds[i]))

    return results

```

```

def evaluate(model: nn.Module,
            data: Union[MoleculeDataset, ReactionDataset],
            num_tasks: int,
            metric_func: Callable,
            batch_size: int,
            dataset_type: str,
            stage: str,
            scaler: StandardScaler = None,
            logger: logging.Logger = None) -> List[float]:
    """
    Evaluates an ensemble of models on a dataset.

    :param model: A model.
    :param data: A MoleculeDataset or ReactionDataset.
    :param num_tasks: Number of tasks.
    :param metric_func: Metric function which takes in a list of targets
        and a list of predictions.
    :param batch_size: Batch size.
    :param dataset_type: Dataset type.
    :param scaler: A StandardScaler object fit on the training targets.
    :param logger: Logger.
    :return: A list with the score for each task based on 'metric_func'.
    """
    preds, _, _, _ = predict(
        model=model,
        data=data,
        batch_size=batch_size,
        scaler=scaler
    )

    targets = data.targets()
    if stage == 'train' and scaler != None:
        targets = scaler.inverse_transform(targets)

    results = evaluate_predictions(
        preds=preds,
        targets=targets,
        num_tasks=num_tasks,
        metric_func=metric_func,
        dataset_type=dataset_type,
        logger=logger
    )

    return results

from argparse import Namespace
from logging import Logger
import os
from typing import Tuple

import numpy as np

from .run_training import run_training
from chemprop.data.utils import get_task_names
from chemprop.utils import makedirs

def cross_validate(args: Namespace, logger: Logger = None) -> Tuple[float,
float]:
    """k-fold cross validation"""
    info = logger.info if logger is not None else print

    # Initialize relevant variables
    init_seed = args.seed
    save_dir = args.save_dir
    task_names = get_task_names(args.data_path, reaction=args.reaction)

```

```

# Run training on different random seeds for each fold
all_scores = []
for fold_num in range(args.num_folds):
    info(f'Fold {fold_num}')
    args.seed = init_seed + fold_num
    args.save_dir = os.path.join(save_dir, f'fold_{fold_num}')
    makedirs(args.save_dir)
    model_scores = run_training(args, logger)
    all_scores.append(model_scores)
all_scores = np.array(all_scores)

# Report results
info(f'{args.num_folds}-fold cross validation')

# Report scores for each fold
for fold_num, scores in enumerate(all_scores):
    info(f'Seed {init_seed + fold_num} ==> test {args.metric} = {np.
        nanmean(scores):.6f}')

    if args.show_individual_scores:
        for task_name, score in zip(task_names, scores):
            info(f'Seed {init_seed + fold_num} ==> test {task_name} {
                args.metric} = {score:.6f}')

# Report scores across models
avg_scores = np.nanmean(all_scores, axis=1) # average score for each
model across tasks
mean_score, std_score = np.nanmean(avg_scores), np.nanstd(avg_scores)
info(f'Overall test {args.metric} = {mean_score:.6f} +/- {std_score:.6
f}')

if args.show_individual_scores:
    for task_num, task_name in enumerate(task_names):
        info(f'Overall test {task_name} {args.metric} = '
            f'{np.nanmean(all_scores[:, task_num]):.6f} +/- {np.
                nanstd(all_scores[:, task_num]):.6f}')

return mean_score, std_score

```

vi Алгоритм прогнозування виходу продукту хімічної реакції

```

from typing import List, Union

import numpy as np
import torch
import torch.nn as nn
from tqdm import trange

from chemprop.data import MoleculeDataset, ReactionDataset, StandardScaler
import collections

def predict(model: nn.Module,
            data: Union[MoleculeDataset, ReactionDataset],
            batch_size: int,
            scaler: StandardScaler = None) -> List[List[float]]:
    """
    Makes predictions on a dataset using an ensemble of models.

    :param model: A model.
    :param data: A MoleculeDataset or ReactionDataset.
    :param batch_size: Batch size.

```

```

:param scaler: A StandardScaler object fit on the training targets.
:return: A list of lists of predictions. The outer list is examples
while the inner list is tasks.
"""
model.eval()

preds = []

last_layers = []
middle_layer_representations = []

num_changed_atoms = []

num_iters, iter_step = len(data), batch_size

for i in trange(0, num_iters, iter_step):
    # Prepare batch
    data_batch = data[i:i + batch_size]
    mol_batch = ReactionDataset(data_batch) if isinstance(data,
        ReactionDataset) else MoleculeDataset(data_batch)
    smiles_batch, features_batch = mol_batch.smiles(), mol_batch.
        features()

    # Run model
    batch = smiles_batch

    with torch.no_grad():
        if isinstance(data, ReactionDataset):
            rbatch, pbatch = list(zip(*batch))
            batch_preds, num_not_zero_diff,
                middle_layer_representation, last_layer = model(rbatch,
                    pbatch,

                    middle_layer_representation = middle_layer_representation.
                        tolist()
                    last_layer = last_layer.tolist()

            else:
                batch_preds = model(batch, features_batch)

    batch_preds = batch_preds.data.cpu().numpy()

    # Inverse scale if regression
    if scaler is not None:
        batch_preds = scaler.inverse_transform(batch_preds)

    # Collect vectors
    batch_preds = batch_preds.tolist()
    preds.extend(batch_preds)
    if isinstance(data, ReactionDataset):
        changed_atoms = np.array([c[0] for c in num_not_zero_diff])
        num_changed_atoms.extend(changed_atoms.tolist())
        middle_layer_representations.extend(
            middle_layer_representation)
        last_layers.extend(last_layer)

return preds, num_changed_atoms, middle_layer_representations,
    last_layers

from argparse import Namespace
import csv
from typing import List, Optional

```

```

import numpy as np
import torch
from tqdm import tqdm

from .predict import predict
from chemprop.data import MoleculeDataset, ReactionDataset
from chemprop.data.utils import get_data, get_data_from_smiles
from chemprop.utils import load_args, load_checkpoint, load_scalers

def make_predictions(args: Namespace, smiles: List[str] = None) -> List[
Optional[List[float]]]:
    """
    Makes predictions. If smiles is provided, makes predictions on smiles.
    Otherwise makes predictions on args.test_data.

    :param args: Arguments.
    :param smiles: Smiles to make predictions on.
    :return: A list of lists of target predictions.
    """
    if args.gpu is not None:
        torch.cuda.set_device(args.gpu)

    print('Loading training args')
    scaler, features_scaler = load_scalers(args.checkpoint_paths[0])
    train_args = load_args(args.checkpoint_paths[0])

    # Update args with training arguments
    for key, value in vars(train_args).items():
        if not hasattr(args, key):
            setattr(args, key, value)

    print('Loading data')
    if smiles is not None:
        test_data = get_data_from_smiles(smiles=smiles,
            skip_invalid_smiles=False, args=args)
    else:
        test_data = get_data(path=args.test_path, args=args,
            use_compound_names=args.use_compound_names, skip_invalid_smiles
            =False)

    print('Validating SMILES')
    if args.reaction:
        valid_indices = [i for i in range(len(test_data))
            if test_data[i].rmol is not None and test_data[i]
            ].pmol is not None]
    else:
        valid_indices = [i for i in range(len(test_data)) if test_data[i].
            mol is not None]
    full_data = test_data
    valid_data = [test_data[i] for i in valid_indices]
    test_data = ReactionDataset(valid_data) if args.reaction else
        MoleculeDataset(valid_data)

    # Edge case if empty list of smiles is provided
    if len(test_data) == 0:
        return [None] * len(full_data)

    if args.use_compound_names:
        compound_names = test_data.compound_names()
    print(f'Test size = {len(test_data):,}')

    features = full_data.features()

    # Normalize features
    if train_args.features_scaling:
        test_data.normalize_features(features_scaler)

```



```

# Predict with each model individually and sum predictions
if args.dataset_type == 'multiclass':
    sum_preds = np.zeros((len(test_data), args.num_tasks, args.
        multiclass_num_classes))
else:
    sum_preds = np.zeros((len(test_data), args.num_tasks))
print(f'Predicting with an ensemble of {len(args.checkpoint_paths)}
models')
for checkpoint_path in tqdm(args.checkpoint_paths, total=len(args.
checkpoint_paths)):
    # Load model
    model = load_checkpoint(checkpoint_path, cuda=args.cuda,
        current_args=args)
    model_preds, num_not_zero_atoms, middle_layer_representations,
        last_layer = predict(
        model=model,
        data=test_data,
        batch_size=args.batch_size,
        scaler=scaler
    )
    sum_preds += np.array(model_preds)

# Ensemble predictions
avg_preds = sum_preds / len(args.checkpoint_paths)
avg_preds = avg_preds.tolist()

# Save predictions
assert len(test_data) == len(avg_preds)
print(f'Saving predictions to {args.preds_path}')

# Put Nones for invalid smiles
full_preds = [None] * len(full_data)
for i, si in enumerate(valid_indices):
    full_preds[si] = avg_preds[i]
avg_preds = full_preds
test_smiles = full_data.smiles()
targets = full_data.targets()
# if scaler:
#     targets = scaler.inverse_transform(targets)
# Write predictions
with open(args.preds_path, 'w') as f:
    writer = csv.writer(f)

    header = []

    if args.use_compound_names:
        header.append('compound_names')

    if args.reaction:
        header.extend(['rsmiles', 'psmiles'])
    else:
        header.append('smiles')

    if args.dataset_type == 'multiclass':
        for name in args.task_names:
            for i in range(args.multiclass_num_classes):
                header.append(name + '_class' + str(i))
    else:
        header.extend(args.task_names)

    header.append('predicted')
    header.append('label')
    header.append('num not zero atoms')
    writer.writerow(header)

```

```

for i in range(len(avg_preds)):
    row = []

    if args.use_compound_names:
        row.append(compound_names[i])

    if args.reaction:
        row.extend(test_smiles[i])
    else:
        row.append(test_smiles[i])

    if avg_preds[i] is not None:
        if args.dataset_type == 'multiclass':
            for task_probs in avg_preds[i]:
                row.extend(task_probs)
        else:
            row.extend(avg_preds[i])
    else:
        if args.dataset_type == 'multiclass':
            row.extend([''] * args.num_tasks * args.multiclass_num_classes)
        else:
            row.extend([''] * args.num_tasks)
    if args.dataset_type == 'multiclass':
        row.append(np.argmax(avg_preds[i]))
    else:
        row.append((np.array((avg_preds[i])) > 0.5).astype(np.int8)
                    [0])
        row.append(targets[i][0])
        row.append(num_not_zero_atoms[i] if num_not_zero_atoms else
                    row.append(''))
        writer.writerow(row)

if args.middle_representation_path:
    with open(args.middle_representation_path, 'w') as f:
        writer = csv.writer(f)

        header = []

        if args.use_compound_names:
            header.append('compound_names')

        if args.reaction:
            header.extend(['rsmiles', 'psmiles'])
        else:
            header.append('smiles')

        for i in range(len(middle_layer_representations[0])):
            header.append('entry_no_' + str(i))

        writer.writerow(header)
        for i in range(len(middle_layer_representations)):
            row = []

            if args.use_compound_names:
                row.append(compound_names[i])

            if args.reaction:
                row.extend(test_smiles[i])
            else:
                row.append(test_smiles[i])

            row.extend(middle_layer_representations[i])

            writer.writerow(row)

```

```

if args.last_ffn_representation_path:
    with open(args.last_ffn_representation_path, 'w') as f:
        writer = csv.writer(f)

        header = []

        if args.use_compound_names:
            header.append('compound_names')

        if args.reaction:
            header.extend(['rsmiles', 'psmiles'])
        else:
            header.append('smiles')

        for i in range(len(last_layer[0])):
            header.append('entry_no_' + str(i))

        writer.writerow(header)
        for i in range(len(last_layer)):
            row = []

            if args.use_compound_names:
                row.append(compound_names[i])

            if args.reaction:
                row.extend(test_smiles[i])
            else:
                row.append(test_smiles[i])

            row.extend(last_layer[i])

            writer.writerow(row)

return avg_preds, middle_layer_representations, last_layer

```

IV Додаток Г. ПРОГРАМНИЙ КОД МЕТОДУ РОЗРОБКИ МОЛЕКУЛ- КАНДИДАТІВ У ЛІКАРСЬКІ РЕЧОВИНИ

i Підготовка даних

```
import torch
import torch.utils.data

class EmbDataset(torch.utils.data.Dataset):
    def __init__(self, config):
        self.config = config

        # load embeddings
        data_file = open(self.config['path_to_emb'], 'r')
        item = data_file.readline().split(',')
        irow = 1
        for _ in data_file:
            irow += 1
        number_of_columns = len(item)

        smiles = []
        representation_as_tensor = torch.zeros(irow, (number_of_columns -
            1))

        if self.config['path_to_labels']:
            # load labels
            labels_file = open(self.config['path_to_labels'], 'r')
            label0 = {}
            for line in labels_file:
                item = line.split(",")
                label0[item[0].strip()] = float(item[1].strip())
            labels_file.close()
            if self.config['labels_one_hot']:
                self.labels_as_tensor = torch.zeros(irow, 2)
            else:
                self.labels_as_tensor = torch.zeros(irow, 1)

        irow = 0
        data_file.seek(0, 0)
        for line in data_file:
            item = line.split(",")
            smiles.append(item[0])
            for icolumn in range(1, number_of_columns):
                representation_as_tensor[irow][(icolumn - 1)] = float(item
                    [icolumn])

            if self.config['path_to_labels']:
                if self.config['labels_one_hot']:
                    if label0[item[0]] == 0.:
                        self.labels_as_tensor[irow][0] = 1.0
                    else:
                        self.labels_as_tensor[irow][1] = 1.0
                else:
                    self.labels_as_tensor[irow] = label0[item[0]]
            irow += 1
```

```

        data_file.close()

        self.length = irow
        self.smiles_strings = smiles
        self.representation = representation_as_tensor

    def __len__(self):
        return self.length

    def __getitem__(self, index):
        return self.smiles_strings[index], self.representation[index],
            self.labels_as_tensor[index]

import torch
import torch.utils.data
import random
import re
from rdkit import Chem
from rdkit import rdBase

# class SMILESDataSet(torch.utils.data.Dataset):
#     def __init__(self, path_to_csv_file, size_of_smiles_dict):
#         data_file = open(path_to_csv_file, 'r')
#         item = data_file.readline().split(',')
#         irow = 0
#         icolumn = len(item)
#         position = data_file.tell()
#         for _ in data_file:
#             irow += 1
#
#         label = []
#         representation_as_tensor = torch.empty(irow, (icolumn - 1) *
# size_of_smiles_dict)
#
#         irow = 0
#         number_of_columns = icolumn
#         data_file.seek(position)
#         for line in data_file:
#             item = line.split(',')
#             lbl, representation_as_tensor[irow] = csv_to_onehot(item,
# size_of_smiles_dict)
#             label.append(lbl)
#             irow += 1
#         data_file.close()
#
#         self.length = irow
#         self.label = label
#         self.representation = representation_as_tensor
#         self.number_of_columns = number_of_columns - 1
#
#     def __len__(self):
#         return self.length
#
#     def __getitem__(self, index):
#         return self.label[index], self.representation[index]
#
#     def col_num(self):
#         return self.number_of_columns

class SMILESDataSet:
    def __init__(self, config):
        self.config = config

        with open(config['path_to_smiles_dict'], 'r') as f:
            self.char2indx = {l.strip().split('\t')[1]: int(l.strip())}

```

```

        split('\t')[0]) for l in f.readlines()}
self.indx2char = {v: k for k, v in self.char2indx.items()}
self.max_smiles_len = self.config['max_smiles_len']
self.pad_token = 0
self.sos_token = len(self.indx2char)
self.eos_token = len(self.indx2char) + 1

self.smiles = []
self.smiles_strings = []
self.labels = []

with open(self.config['path_to_datafile'], 'r') as f:
    lines = list(f.readlines())
    transformed = list(map(self.transform, lines))
    # self.smiles_strings.extend([tr[0] for tr in transformed if
    # tr])
    # self.smiles.extend([tr[1] for tr in transformed if tr])
    # if self.config['mode'] == 'predictor':
    #     self.labels.extend([tr[2] for tr in transformed if tr])
    for tr in transformed:
        if tr and tr not in self.smiles_strings:
            self.smiles_strings.append(tr[0])
            self.smiles.append(tr[1])
            if self.config['mode'] == 'predictor':
                self.labels.append(tr[2])

def __len__(self):
    return len(self.smiles)

def __getitem__(self, index):
    if self.config['mode'] == 'ae' or self.config['mode'] == 'seq2seq':
        :
        if self.config['rn_in_input']:
            return torch.FloatTensor(add_random_noise(self.smiles[
                index], self.config['max_smiles_len'], len(self.
                char2indx), self.eos_token)), \
                torch.FloatTensor(self.smiles[index])
        return torch.FloatTensor(self.smiles[index]), torch.
            FloatTensor(self.smiles[index])
    elif self.config['mode'] == 'predictor':
        return torch.Tensor(self.smiles[index]), torch.Tensor(self.
            labels[index])
    elif self.config['mode'] == 'denovo':
        return torch.Tensor(self.smiles[index])

def transform(self, l):
    sm_str = re.split('[,\t]', l.strip())[0]
    # sm_str = Chem.MolToSmiles(Chem.MolFromSmiles(sm_str)) # convert
    # to canonical
    sm = split_smiles_string(sm_str, self.char2indx)
    real_sm_len = len(sm)
    if len(sm) > self.config['max_smiles_len']:
        return []
    try:
        sm = [self.char2indx[ch] for ch in sm]
    except KeyError:
        print(sm_str)
        return []
    sm = pad_smile(sm, self.config['max_smiles_len'])

    if self.config['smiles_one_hot']:
        sm = [indx_to_onehot(indx, self.char2indx) for indx in sm]

    lbl = None
    if self.config['mode'] == 'predictor':
        lbl = float(l.strip().split(',')[1])
        if self.config['labels_one_hot']:

```

```

        lbl = indx_to_onehot(1, [0, 0])

    return sm_str, sm, lbl

def indx_to_onehot(indx, smiles_dict):
    vec = [0] * len(smiles_dict)
    vec[indx] = 1
    return vec

def pad_smile(smile_indices, max_smiles_len):
    while len(smile_indices) < max_smiles_len:
        smile_indices.append(0)
    return smile_indices

def split_smiles_string(smiles_string, smiles_dict):
    chars = []
    for ch in smiles_dict:
        if ch == '++':
            chars.append('\+\+')
        elif ch in ('$', '\\', '*', '[', ']', '(', ')', '.', '+'):
            chars.append '\\' + ch
        else:
            chars.append(ch)
    delimiter = '|'.join(chars)
    splited = re.split('(' + delimiter + ')', smiles_string)
    splited = list(filter(None, splited))
    return splited

def onehot_to_smiles(as_tensor, smiles_dict, number_of_columns):
    as_smiles = ''
    for i in as_tensor.reshape(number_of_columns, len(smiles_dict)).max(1)
        [1]:
        as_smiles += smiles_dict[i.item()]
    return as_smiles.lstrip('0')

def csv_to_onehot(csv_string, size_of_smiles_dict):
    representation_as_tensor = torch.zeros((len(csv_string) - 1) *
        size_of_smiles_dict)
    label = csv_string[0]
    for i in range(1, len(csv_string)):
        representation_as_tensor[(i - 1) * size_of_smiles_dict + int(
            csv_string[i])] = 1
    return label, representation_as_tensor

def is_valid_smiles(smiles_string):
    rdBase.DisableLog('rdApp.error')
    mol = Chem.MolFromSmiles(smiles_string)
    if mol is not None:
        valid = True
    else:
        valid = False
    return valid

def is_new_smiles(smiles_string, smiles_ds):
    return smiles_string in smiles_ds.smiles_strings

def add_random_noise(smile_lst, max_smiles_len, n_chars, eos_token):
    real_sm_len = smile_lst.index(eos_token)

```

```

num_edits = random.choices([0, 1, 2, 3], [0.1, 0.3, 0.3, 0.3], k=1)[0]
edit_indecs = [random.randint(0, real_sm_len - 1) for _ in range(
    num_edits)]

if real_sm_len >= (max_smiles_len - num_edits):
    edit_op = random.choices(['repl', 'del'], [0.95, 0.05], k=
        num_edits)
else:
    edit_op = random.choices(['repl', 'ins', 'del'], [0.9, 0.05,
        0.05], k=num_edits)
smile_with_rand = []
for i in range(real_sm_len):
    if i in edit_indecs:
        if edit_op[edit_indecs.index(i)] == 'repl':
            smile_with_rand.append(random.randint(1, n_chars - 1))
        elif edit_op[edit_indecs.index(i)] == 'ins':
            smile_with_rand.append(random.randint(1, n_chars - 1))
        else:
            smile_with_rand.append(smile_lst[i])
smile_with_rand.append(eos_token)
smile_with_rand = pad_smile(smile_with_rand, max_smiles_len + 1)
# if len(smile_with_rand) != len(smile_lst):
#     print(smile_lst)
#     print(smile_with_rand)
return smile_with_rand

def decode_smiles_from_seq2seq(decoder_output, smiles_ds):
    decoder_output = decoder_output.transpose(0, 1)
    decoded_smiles = []
    _, decoder_output_indices = decoder_output.topk(1)
    decoder_output_indices = decoder_output_indices.squeeze(2)
    for di in range(decoder_output_indices.size()[0]):
        decoded_smile = []
        for indx in decoder_output_indices[di]: #skip SOS token
            if indx.item() == smiles_ds.sos_token:
                continue
            elif indx.item() == smiles_ds.eos_token:
                decoded_smile.append('EOS')
                break
            decoded_smile.append(smiles_ds.indx2char[indx.item()])
        decoded_smiles.append(decoded_smile)
    return decoded_smiles

```

ii Архітектура автокодувальника

```

import torch
# from torch import nn, optim
from torch import nn
from torch.nn import functional as F

class AE(nn.Module):
    def __init__(self, config):
        super(AE, self).__init__()

        self.drop = nn.Dropout(p=0.0)

        # input Tensor - (N_batch, Channels = 1, Height = 58, Width = 60)
        self.conv0 = nn.Conv2d(1, 60, kernel_size=(58, 3), stride=(1, 1))
        # output Tensor - (N_batch, Channels = 60, Height = 1, Width = 58)

        # input Tensor - (N_batch, Channels = 60, Height = 1, Width = 58)
        self.conv1 = nn.Conv2d(60, 87, kernel_size=(1, 19), stride=(1, 1))
        # output Tensor - (N_batch, Channels = 87, Height = 1, Width = 40)

```



```

# input Tensor - (N_batch, Channels = 87, Height = 1, Width = 40)
self.conv2 = nn.Conv2d(87, 116, kernel_size=(1, 11), stride=(1, 1)
)
# output Tensor - (N_batch, Channels = 116, Height = 1, Width =
30)

# input Tensor - (N_batch, Channels = 116, Height = 1, Width = 30)
self.conv3 = nn.Conv2d(116, 120, kernel_size=(1, 2), stride=(1, 1)
)
# output Tensor - (N_batch, Channels = 120, Height = 1, Width =
29)

self.efc0 = nn.Linear(3480, 512)

self.dfc0 = nn.Linear(512, 3480)

# input Tensor - (N_batch, Channels = 120, Height = 1, Width = 29)
self.deconv0 = nn.ConvTranspose2d(120, 116, kernel_size=(1, 2),
stride=(1, 1))
# output Tensor - (N_batch, Channels = 116, Height = 1, Width =
30)

# input Tensor - (N_batch, Channels = 116, Height = 1, Width = 30)
self.deconv1 = nn.ConvTranspose2d(116, 87, kernel_size=(1, 11),
stride=(1, 1))
# output Tensor - (N_batch, Channels = 87, Height = 1, Width = 40)

# input Tensor - (N_batch, Channels = 87, Height = 1, Width = 40)
self.deconv2 = nn.ConvTranspose2d(87, 60, kernel_size=(1, 19),
stride=(1, 1))
# output Tensor - (N_batch, Channels = 60, Height = 1, Width = 58)

# input Tensor - (N_batch, Channels = 60, Height = 1, Width = 58)
self.deconv3 = nn.ConvTranspose2d(60, 1, kernel_size=(58, 3),
stride=(1, 1))
# output Tensor - (N_batch, Channels = 1, Height = 58, Width = 60)

def encode(self, x):
h0 = F.relu(self.drop(self.conv0(x.view(-1, 1, 58, 60))))
h1 = F.relu(self.drop(self.conv1(h0) + h0.view(-1, 87, 1, 40)))
h2 = F.relu(self.drop(self.conv2(h1) + h0.view(-1, 116, 1, 30) +
h1.view(-1, 116, 1, 30)))
h3 = F.relu(self.drop(self.conv3(h2) + h0.view(-1, 120, 1, 29) +
h1.view(-1, 120, 1, 29) +
h2.view(-1, 120, 1, 29)))
h4 = F.relu(self.drop(self.efc0(h3.view(-1, 3480))))
return h4

def decode(self, z):
h0 = F.relu(self.drop(self.drop(self.dfc0(z))))
h1 = F.relu(self.drop(self.deconv0(h0.view(-1, 120, 1, 29))))
h2 = F.relu(self.drop(self.deconv1(h1) + h1.view(-1, 87, 1, 40)))
h3 = F.relu(self.drop(self.deconv2(h2) + h1.view(-1, 60, 1, 58) +
h2.view(-1, 60, 1, 58)))
h4 = self.deconv3(h3)
h = torch.sigmoid(h4.view(-1, 3480))
return h

def forward(self, x):
h = self.encode(x)
x = self.decode(h)
return x

```

iii Налаштування автокодувальника

```

# hyperparams for training Autoencoder generative model

path_to_datafile_train: 'data/SMILES/smiles.raw.train'
path_to_datafile_test: 'data/SMILES/smiles.raw.test'
path_to_smiles_dict: 'data/SMILES/Dictionary'
mode: 'ae'
rn_in_input: False
smiles_one_hot: True

model: 'ae'
max_smiles_len: 60
smiles_dict_size: 58

path_to_save_model: 'ae/ae.pth'
save_model_every: 10

batch_size: 64
l_r: 0.001
decay_lr_every: 40
gamma: 0.1
num_epochs: 50

```

iv Алгоритм тренування автокодувальника

```

import torch
from torch.nn import functional as F

def loss_function(recon_x, x):
    bce = F.binary_cross_entropy(recon_x, x, reduction='sum')
    return bce

def train(epoch, model, optimizer, device, train_loader):
    model.train()
    train_loss = 0
    for batch_idx, (_, data) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch = model(data)
        loss = loss_function(recon_batch, data)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

        print('Train Epoch: {} [{} / {} ( {:.0f} % )] \t Loss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader),
            loss.item() / len(data)))
    print('====> Epoch: {} Average loss on train set: {:.6f}'.format(
        epoch, train_loss / len(train_loader.dataset)))

def test(epoch, model, device, test_loader):
    model.eval()

    test_loss = 0
    with torch.no_grad():
        for i, (_, data) in enumerate(test_loader):
            data = data.to(device)
            recon_batch = model(data)
            test_loss += loss_function(recon_batch, data).item()
    test_loss /= len(test_loader.dataset)

```

```

print('====> Epoch: {} Average loss on test set: {:.6f}'.format(epoch
, test_loss))

# esol_loss = 0
# with torch.no_grad():
#     for i, (_, data) in enumerate(esol_loader):
#         data = data.to(device)
#         recon_batch = model(data)
#         esol_loss += loss_function(recon_batch, data).item()
# esol_loss /= len(esol_loader.dataset)
# print('====> Epoch: {} Average loss on ESol set: {:.6f}'.format(
epoch, esol_loss))

#
# esol_test_loss = 0
# with torch.no_grad():
#     for i, (_, data) in enumerate(esol_test_loader):
#         data = data.to(device)
#         recon_batch = model(data)
#         esol_test_loss += loss_function(recon_batch, data).item()
# esol_test_loss /= len(esol_test_loader.dataset)
# print('====> Epoch: {} Average loss on ESol test set: {:.6f}'.
format(epoch, esol_test_loss))

```

v Архітектура класифікаційної моделі

```

import torch
from torch import nn, optim
from torch.nn import functional as F

class C2F(nn.Module):
    def __init__(self, config):
        super(C2F, self).__init__()

        self.drop = nn.Dropout(p=0.0)

        self.fc0 = nn.Linear(512, 32)
        self.fc1 = nn.Linear(32, 2)

    def forward(self, x):

        h0 = x.view(-1, 512)

        h0 = torch.sigmoid(self.drop(self.fc0(h0))) - 0.5
        h0 = torch.sigmoid(self.drop(self.fc1(h0))) - 0.5

        h0 = F.softmax(h0, dim=1)

        return h0

```

vi Налаштування класифікаційної моделі

```

# hyperparams for training classifier on embeddings from AE latent space

path_to_embd_dataset_train: 'data/ames/ames_embds_train.csv'
path_to_embd_dataset_test: 'data/ames/ames_embds_test.csv'
path_to_labels_train: 'data/ames/ames_train_raw.csv'
path_to_labels_test: 'data/ames/ames_test_raw.csv'

labels_one_hot: True

model: 'c2f'
mode: 'predictor'
path_to_save_model: 'c2f/c2f_ames.pth'

```

```
save_model_every: 50
```

```
batch_size: 64  
l_r: 0.001  
decay_lr_every: 100  
gamma: 0.1  
num_epochs: 40
```

vii Алгоритм тренування класифікаційної моделі

```
import torch  
from torch import nn, optim  
from torch.nn import functional as F  
  
def loss_function(recon_x, x):  
    loss = F.binary_cross_entropy(recon_x, x, reduction='sum')  
    #loss = F.mse_loss(recon_x, x, reduction='sum')  
    return loss  
  
def train(epoch, model, optimizer, device, train_loader):  
    model.train()  
    train_loss = 0  
    correct = 0  
    for batch_idx, (label, data, feature) in enumerate(train_loader):  
        data = data.to(device)  
        feature = feature.to(device)  
        optimizer.zero_grad()  
        recon_feature = model(data)  
        #loss = loss_function(recon_feature, feature.view(-1, 1))  
        loss = loss_function(recon_feature, feature.view(-1, 2))  
        loss.backward()  
        train_loss += loss.item()  
        optimizer.step()  
        correct += (recon_feature.topk(1)[1] == feature.topk(1)[1]).sum().  
            item()  
    print('====> Epoch: {} Average loss on train set: {:.6f}'.format(  
        epoch, train_loss / len(train_loader.dataset)))  
    print('====> Epoch: {} Accuracy on train set: {:.2f}'.format(  
        epoch, correct / len(train_loader.dataset)))  
  
def test(epoch, model, device, test_loader):  
    model.eval()  
    test_loss = 0  
    correct = 0  
    with torch.no_grad():  
        for batch_idx, (label, data, feature) in enumerate(test_loader):  
            data = data.to(device)  
            feature = feature.to(device)  
            recon_feature = model(data)  
            #loss = loss_function(recon_feature, feature.view(-1, 1))  
            loss = loss_function(recon_feature, feature.view(-1, 2))  
            test_loss += loss.item()  
            correct += (recon_feature.topk(1)[1] == feature.topk(1)[1]).  
                sum().item()  
    print('====> Epoch: {} Average loss on test set : {:.6f}'.format(  
        epoch, test_loss / len(test_loader.dataset)))  
    print('====> Epoch: {} Accuracy on test set: {:.2f}'.format(  
        epoch, correct / len(test_loader.dataset)))
```

viii Архітектура регресійної моделі

```
import torch
from torch import nn, optim
from torch.nn import functional as F

class E2F(nn.Module):
    def __init__(self, config):
        super(E2F, self).__init__()

        self.fc00 = nn.Linear(512, 512)
        self.fc01 = nn.Linear(512, 256)

        self.fc10 = nn.Linear(256, 256)
        self.fc11 = nn.Linear(256, 128)

        self.fc20 = nn.Linear(128, 128)
        self.fc21 = nn.Linear(128, 64)

        self.fc30 = nn.Linear(64, 64)
        self.fc31 = nn.Linear(64, 32)

        self.fc4 = nn.Linear(32, 8)

        self.fc5 = nn.Linear(8, 4)

        self.fc6 = nn.Linear(4, 2)

        self.fc7 = nn.Linear(2, 1)

        self.prelu = nn.PReLU()

    def forward(self, x):

        h0 = x.view(-1, 512)

        h1 = F.relu(self.fc00(h0) + h0)
        h2 = F.relu(self.fc00(h1) + h1 + h0)
        h3 = F.relu(self.fc00(h2) + h2 + h1 + h0)
        h0 = F.relu(self.fc01(h3))

        h1 = F.relu(self.fc10(h0) + h0)
        h2 = F.relu(self.fc10(h1) + h1 + h0)
        h3 = F.relu(self.fc10(h2) + h2 + h1 + h0)
        h4 = F.relu(self.fc10(h3) + h3 + h2 + h1 + h0)
        h0 = F.relu(self.fc11(h4))

        h1 = F.relu(self.fc20(h0) + h0)
        h2 = F.relu(self.fc20(h1) + h1 + h0)
        h3 = F.relu(self.fc20(h2) + h2 + h1 + h0)
        h4 = F.relu(self.fc20(h3) + h3 + h2 + h1 + h0)
        h0 = F.relu(self.fc21(h4))

        h1 = F.relu(self.fc30(h0) + h0)
        h2 = F.relu(self.fc30(h1) + h1 + h0)
        h3 = F.relu(self.fc30(h2) + h2 + h1 + h0)
        h4 = F.relu(self.fc30(h3) + h3 + h2 + h1 + h0)
        h0 = F.relu(self.fc31(h4))

        h0 = F.relu(self.fc4(h0))
        h0 = F.relu(self.fc5(h0))
        h2 = self.prelu(self.fc6(h0))
        h1 = self.fc7(h2)

        return h1 #h2
```

ix Налаштування регресійної моделі

```
# hyperparams for training regressor model on embeddings from AE latent
space
path_to_embd_dataset_train: 'data/esol/esol_embds_train.csv'
path_to_embd_dataset_test: 'data/esol/esol_embds_test.csv'
path_to_labels_train: 'data/esol/esol_train_raw.csv'
path_to_labels_test: 'data/esol/esol_test_raw.csv'

labels_one_hot: False

model: 'e2f'
mode: 'predictor'
path_to_save_model: 'e2f/e2f_sol.pth'
save_model_every: 5

batch_size: 64
l_r: 0.001
decay_lr_every: 10
gamma: 0.1
num_epochs: 10
```

x Алгоритм тренування регресійних моделей

```
import torch
from torch import nn, optim
from torch.nn import functional as F

def loss_function(recon_x, x):
    return F.mse_loss(recon_x, x, reduction='sum')

def train(epoch, model, optimizer, device, train_loader):
    model.train()
    train_loss = 0
    for batch_idx, (label, data, feature) in enumerate(train_loader):
        data = data.to(device)
        feature = feature.to(device)
        optimizer.zero_grad()
        recon_feature = model(data)
        loss = loss_function(recon_feature, feature.view(-1, 1))
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
    print('====> Epoch: {} Average loss on train set: {:.6f}'.format(
        epoch, train_loss / len(train_loader.dataset)))

def test(epoch, model, device, test_loader):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for batch_idx, (label, data, feature) in enumerate(test_loader):
            data = data.to(device)
            feature = feature.to(device)
            recon_feature = model(data)
            loss = loss_function(recon_feature, feature.view(-1, 1))
            test_loss += loss.item()
    print('====> Epoch: {} Average loss on test set : {:.6f}'.format(
        epoch, test_loss / len(test_loader.dataset)))
```

xi Архітектура моделі виправлення помилок

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Seq2Seq(nn.Module):
    def __init__(self, config):
        super(Seq2Seq, self).__init__()
        self.config = config
        self.config['max_smiles_len'] += 1
        self.config['smiles_dict_size'] += 2
        self.encoder = EncoderRNN(config['smiles_dict_size'], config['cell_type'], config['hidden_size'],
                                  config['num_layers'], config['encoder_dropout_p'])
        self.decoder = AttnDecoderRNN(config['smiles_dict_size'], config['cell_type'], config['hidden_size'],
                                       config['num_layers'], config['max_smiles_len'], config['decoder_dropout_p'])
        self.curr_teacher_forcing_p = self.config['start_teacher_forcing_p']
        self.teacher_forcing_decay = self.config['teacher_forcing_decay']

    def forward(self, input, target):
        encoder_outputs, encoder_hidden = self.encode(input)
        decoder_outputs = self.decode(target, encoder_hidden,
                                       encoder_outputs)
        return decoder_outputs

    def encode(self, input_batch):
        batch_size = input_batch.size()[0]
        encoder_hidden = self.encoder.init_hidden(self.config['device'],
                                                  batch_size=batch_size)
        input_batch = input_batch.transpose(0, 1)
        input_length = input_batch.size()[0]
        encoder_outputs = torch.zeros(self.config['max_smiles_len'],
                                       batch_size, self.encoder.hidden_size, device=self.config['device'])
        for ei in range(input_length):
            encoder_output, encoder_hidden = self.encoder(
                input_batch[ei], encoder_hidden, batch_size=batch_size)
            # print(encoder_output.size())
            encoder_outputs[ei] = encoder_output[0]
        return encoder_outputs, encoder_hidden

    def decode(self, target_batch, encoder_hidden, encoder_outputs):
        batch_size = target_batch.size()[0]
        target_batch = target_batch.transpose(0, 1)
        target_length = target_batch.size()[0]

        decoder_input = torch.tensor([58] * batch_size, device=self.config['device']) # todo: add sos token as param

        decoder_hidden = encoder_hidden

        use_teacher_forcing = True if torch.rand(1).item() < self.curr_teacher_forcing_p else False # ToDo: add decay

        decoder_outputs = torch.zeros(target_length, batch_size, self.config['smiles_dict_size'],
                                       device=self.config['device'])

        for di in range(self.config['max_smiles_len']):
            decoder_output, decoder_hidden, attention_weights = self.
```

```

        decoder(
            decoder_input, decoder_hidden, encoder_outputs, batch_size
            =batch_size)
    decoder_outputs[di] = decoder_output
    if use_teacher_forcing:
        decoder_input = target_batch[di] # Teacher forcing
    else:
        topv, topi = decoder_output.data.topk(1)
        decoder_input = torch.squeeze(topi).detach()

    return decoder_outputs

class EncoderRNN(nn.Module):
    def __init__(self, input_size, cell, hidden_size, n_layers, dropout_p)
    : # input_size - vocabulary size
    super(EncoderRNN, self).__init__()
    self.cell = cell
    self.hidden_size = hidden_size
    self.n_layers = n_layers

    self.embedding = nn.Embedding(input_size, hidden_size) #
        embedding layer is trained also
    if self.cell == 'GRU':
        self.recurrent = nn.GRU(hidden_size, hidden_size, n_layers,
            dropout=dropout_p)
    elif self.cell == 'LSTM':
        self.recurrent = nn.LSTM(hidden_size, hidden_size, n_layers,
            dropout=dropout_p)

    def forward(self, input, hidden, batch_size):
        embedded = self.embedding(input).view(1, batch_size, -1)
        output = embedded
        output, hidden = self.recurrent(output, hidden)
        return output, hidden

    def init_hidden(self, device, batch_size=1):
        if self.cell == 'LSTM':
            return (torch.zeros(self.n_layers, batch_size, self.
                hidden_size, device=device),
                torch.zeros(self.n_layers, batch_size, self.
                    hidden_size, device=device)
            )
        return torch.zeros(self.n_layers, batch_size, self.hidden_size,
            device=device)

    def get_num_params(self):
        model_parameters = filter(lambda p: p.requires_grad, self.
            parameters())
        return sum([torch.prod(torch.Tensor(list(p.size()))) for p in
            model_parameters]).item()

class DecoderRNN(nn.Module):
    def __init__(self, cell, hidden_size, output_size, n_layers):
    super(DecoderRNN, self).__init__()
    self.cell = cell
    self.hidden_size = hidden_size
    self.n_layers = n_layers

    self.embedding = nn.Embedding(output_size, hidden_size)
    if self.cell == 'GRU':
        self.recurrent = nn.GRU(hidden_size, hidden_size, n_layers)
    elif self.cell == 'LSTM':
        self.recurrent = nn.LSTM(hidden_size, hidden_size, n_layers)
    self.out = nn.Linear(hidden_size, output_size)
    self.softmax = nn.LogSoftmax(dim=1)

```



```

def forward(self, input, hidden, batch_size=1):
    output = self.embedding(input).view(1, batch_size, -1)
    output = F.relu(output)
    output, hidden = self.recurrent(output, hidden)
    output = self.softmax(self.out(output[0]))
    return output, hidden

def init_hidden(self, device, batch_size=1):
    if self.cell == 'LSTM':
        return (torch.zeros(self.n_layers, batch_size, self.
            hidden_size, device=device),
            torch.zeros(self.n_layers, batch_size, self.
            hidden_size, device=device)
        )
    return torch.zeros(self.n_layers, batch_size, self.hidden_size,
        device=device)

def get_num_params(self):
    model_parameters = filter(lambda p: p.requires_grad, self.
        parameters())
    return sum([torch.prod(torch.Tensor(list(p.size()))) for p in
        model_parameters]).item()

class AttnDecoderRNN(nn.Module):
    def __init__(self, output_size, cell, hidden_size, n_layers,
        max_length, dropout_emb=0.5):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_emb = dropout_emb
        # self.dropout_r = dropout_r
        self.max_length = max_length
        self.cell = cell
        self.n_layers = n_layers

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.
            hidden_size)
        self.dropout = nn.Dropout(self.dropout_emb)
        if self.cell == 'GRU':
            self.recurrent = nn.GRU(hidden_size, hidden_size, n_layers,)
        elif self.cell == 'LSTM':
            self.recurrent = nn.LSTM(hidden_size, hidden_size, n_layers,)

        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs, batch_size):
        embedded = self.embedding(input).view(1, batch_size, -1)
        embedded = self.dropout(embedded)

        if self.cell == 'LSTM':
            cell_state = hidden[0]
        else:
            cell_state = hidden
        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], cell_state[0]), 1)), dim=1)
            #instead of embedded - encoder outputs

        # print(encoder_outputs.shape)
        # print(embedded.shape)
        #
        # attn_weights = F.softmax(
        #     self.attn(torch.cat((encoder_outputs, cell_state[0].repeat(
        #         self.max_length, 1, 1))), 1)), dim=1)

```

```

        attn_applied = torch.bmm(attn_weights.unsqueeze(1),
                                  encoder_outputs.view(batch_size,
                                                         encoder_outputs.shape[0], self.
                                                         hidden_size))

        output = torch.cat((embedded[0], attn_applied.view(1, batch_size,
                                                            self.hidden_size)[0]), 1)

        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, hidden = self.recurrent(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

def init_hidden(self, device, batch_size=1):
    if self.cell == 'LSTM':
        return (torch.zeros(self.n_layers, batch_size, self.
                              hidden_size, device=device),
                torch.zeros(self.n_layers, batch_size, self.
                              hidden_size, device=device)
                )
    return torch.zeros(self.n_layers, batch_size, self.hidden_size,
                        device=device)

def get_num_params(self):
    model_parameters = filter(lambda p: p.requires_grad, self.
                               parameters())
    return sum([torch.prod(torch.Tensor(list(p.size()))) for p in
                model_parameters]).item()

```

xii Налаштування моделі виправлення помилок

```
# hyperparams for training seq2seq SMILES spellchecking model
```

```

path_to_datafile_train: 'data/SMILES/smiles.raw.train300k'
path_to_datafile_test: 'data/SMILES/smiles.raw.test10k'
path_to_smiles_dict: 'data/SMILES/Dictionary'
mode: 'seq2seq'
rn_in_input: True
smiles_one_hot: False

```

```

model: 'seq2seq'
hidden_size: 512
embd_size: 64
cell_type: 'LSTM'
num_layers: 2
start_teacher_forcing_p: 1.0
teacher_forcing_decay: 0.99
max_smiles_len: 60
smiles_dict_size: 58
encoder_dropout_p: 0.2
decoder_dropout_p: 0.2

```

```

path_to_save_model: 'seqtoseq/seqtoseq_rn_64emb.pth'
save_model_every: 50

```

```

batch_size: 128
l_r: 0.01
decay_lr_every: 500
gamma: 0.1
num_epochs: 200

```

xiii Тренування моделі виправлення помилок

```
# -*- coding: utf-8 -*-
import torch
import torch.nn as nn
import torch.utils.data
from data_processing.smiles import decode_smiles_from_seq2seq

def train(epoch, model, optimizer, device, train_loader):
    criterion = nn.NLLLoss(ignore_index=train_loader.dataset.pad_token)
    train_loss = 0

    for input_batch, trg_batch in train_loader:
        input_batch = input_batch.to(device)
        trg_batch = trg_batch.to(device)
        decoder_outputs = model(input_batch, trg_batch)
        optimizer.zero_grad()
        loss = 0
        for p,t in zip(decoder_outputs, trg_batch.transpose(0,1)):
            loss += criterion(p, t)
        loss.backward()
        optimizer.step()
        train_loss += loss
    model.curr_teacher_forcing_p *= model.teacher_forcing_decay
    print('====> Epoch: {} Average loss(avg over sum of losses from each
        batch) on train set: {:.6f}'.format(
        epoch, train_loss / len(train_loader.dataset)))

def test(epoch, model, device, test_loader):
    test_loss = 0
    criterion = nn.NLLLoss(ignore_index=test_loader.dataset.pad_token)

    with open('seq2seq_test_results', 'w') as file:
        with torch.no_grad():
            for input_batch, trg_batch in test_loader:
                input_batch = input_batch.to(device)
                trg_batch = trg_batch.to(device)

                decoder_outputs = model(input_batch, trg_batch)
                loss = 0
                for p, t in zip(decoder_outputs, trg_batch.transpose(0, 1)
                    ):
                    loss += criterion(p, t)
                test_loss += loss

            decoded_smiles = decode_smiles_from_seq2seq(
                decoder_outputs.cpu(), test_loader.dataset)

            for i in range(input_batch.size()[0]):
                input_sm = ''.join([test_loader.dataset.indx2char[indx
                    .item()] if indx != test_loader.dataset.eos_token
                    else ' '
                    for indx in input_batch[i]]).split(
                    ' ')[0]
                target_sm = ''.join([test_loader.dataset.indx2char[
                    indx.item()] if indx != test_loader.dataset.
                    eos_token else ' '
                    for indx in trg_batch[i]]).split(
                    ' ')[0]
                pred = ''.join(decoded_smiles[i]).split('EOS')[0]
                if i == 10:
                    print(f'With errors: {input_sm}')
                    print(f'Correct: {target_sm}')
                    print(f'Predicted: {pred}')
                    print('')
```

```

file.write(f'With errors: {input_sm}\n')
file.write(f'Correct:      {target_sm}\n')
file.write(f'Predicted:    {pred}\n')
file.write('\n')

# utils.plot_attention([smiles_lang.indx2char[indx.
#                       item()] for indx in input_batch[i]],
#                       decoded_smiles[i],
#                       decoder attentions[:, i, :], f"{config['exp_path']}
#                       attentions/attn_{i}.png", log=False)
# utils.plot_attention(input_sm, pred,
#                       decoder attentions[:, i, :],
#                       f"{config['exp_path']}
#                       attentions/attn_log_{j}.png")
print('====> Epoch: {} Average loss(avg over sum of losses from each
batch) on test set: {:.6f}'.format(
epoch, test_loss / len(test_loader.dataset)))

```

xiv Каскад моделей

```

from aifordrugdiscovery.data_processing.smiles import SMILESDataSet,
onehot_to_smiles, is_valid_smiles, is_new_smiles,
decode_smiles_from_seq2seq
from aifordrugdiscovery.ae.autoencoder import AE
from aifordrugdiscovery.seqtoseq.seqtoseq import Seq2Seq
from aifordrugdiscovery.c2f.classifier import C2F
from aifordrugdiscovery.e2f.regressor import E2F

import torch
from imblearn.combine import SMOTETomek

from copy import deepcopy

class Pipeline():
    def __init__(self, config, smiles_ds=None):
        self.config = config
        if smiles_ds:
            self.SMILES_ds = smiles_ds
        else:
            self.SMILES_ds = SMILESDataSet(config)

        self.ae = AE(config).to(config['device'])
        self.ae.load_state_dict(torch.load(config['path_to_ae'],
map_location=config['device']))

        self.predictors = []
        for path in config['paths_to_predictors']:
            if 'c2f' in path.split('/'):
                p = C2F(config).to(config['device'])
            elif 'e2f' in path.split('/'):
                p = E2F(config).to(config['device'])
            # p.load_state_dict(torch.load(path, map_location=config['
            device']))
            self.predictors.append(p)

        self.seq2seq = Seq2Seq(deepcopy(config)).to(config['device'])
        # self.seq2seq.load_state_dictionary(config['path_to_seq2seq'])

        self.smt = SMOTETomek(random_state=42)

    def smiles_to_embeddings(self):
        SMILES_dl = torch.utils.data.DataLoader(self.SMILES_ds, batch_size
        =self.config['batch_size'], shuffle=False)

```

```

all_embds = torch.zeros((len(self.SMILES_ds), self.config['
    embd_size']), dtype=torch.float32)

with torch.no_grad():
    for batch_indx, data in enumerate(SMILES_dl):
        data = data.to(self.config['device'])
        embds = self.ae.encode(data)
        all_embds[batch_indx * self.config['batch_size']:
            batch_indx * self.config['batch_size'] + embds.shape
            [0]] = embds.cpu()
return all_embds

def generate_new_embeddings(self, ref_embds):
    embds_for_smote = torch.cat((ref_embds, torch.ones((len(self.
        SMILES_ds) * 2, self.config['embd_size']))))
    new_embds, _ = self.smt.fit_sample(embds_for_smote, torch.cat((
        torch.ones((len(self.SMILES_ds), 1)),
                                                    torch.zeros((
                len(self.
                SMILES_ds)
                * 2, 1))))))

    return new_embds[-len(self.SMILES_ds):]

def predict_properties(self, embds):
    embd_dl = torch.utils.data.DataLoader(embds, batch_size=self.
        config['batch_size'], shuffle=False)
    predictions = {i: [] for i in range(len(self.predictors))}
    with torch.no_grad():
        for batch_indx, data in enumerate(embd_dl):
            data = data.to(self.config['device'])
            for i, prdct in enumerate(self.predictors):
                pred = prdct(data).cpu()
                predictions[i].extend(pred.topk(1)[1].squeeze(1).
                    tolist() if pred.shape[-1] > 1
                    else list(map(lambda x: round(x,
                        3), pred.squeeze(1).tolist()
                    )))

    return predictions

def embeddings_to_smiles(self, embds):
    embd_dl = torch.utils.data.DataLoader(embds, batch_size=self.
        config['batch_size'], shuffle=False)
    all_smiles_onehot = torch.zeros((len(embds), self.config['
        max_smiles_len'], len(self.SMILES_ds.char2indx)), dtype=torch.
        float32)
    for batch_indx, data in enumerate(embd_dl):
        smiles = self.ae.decode(data).cpu().view(-1, self.config['
            max_smiles_len'], len(self.SMILES_ds.char2indx))
        all_smiles_onehot[batch_indx * self.config['batch_size']:
            batch_indx * self.config['batch_size'] + smiles.shape[0]] =
            smiles
    return all_smiles_onehot

def correct_smiles(self, smiles_onehot):

    smiles = [onehot_to_smiles(onehot, self.SMILES_ds.indx2char, self.
        config['max_smiles_len']) for onehot in
        smiles_onehot]
    smiles2indx = {sm: i for i, sm in enumerate(smiles)}

    valid, invalid = [], []

    for sm in smiles:
        if is_valid_smiles(sm):
            valid.append(sm)
        else:
            invalid.append(sm)

```

```

invalid_as_tensor = torch.zeros((len(invalid), self.config['
    max_smiles_len'] + 1), dtype=torch.int64)
invalid_as_tensor[:, -1] = torch.full((1, len(invalid)), self.
    SMILES_ds.eos_token).squeeze()
for i, sm in enumerate(invalid):
    indx = smiles_onehot[smiles2indx[sm]].topk(1)[1]
    invalid_as_tensor[i, :-1] = indx.squeeze(1)

invalid_smiles_dl = torch.utils.data.DataLoader(invalid_as_tensor,
    batch_size=self.config['batch_size'], shuffle=False)
attempted_to_correct = []
for batch_indx, data in enumerate(invalid_smiles_dl):
    data = data.to(torch.int64).to(self.config['device'])
    decoder_outputs = self.seq2seq(data, data)
    attempted_to_correct.extend([''.join(sm).split('EOS')[0] for
        sm in
                                decode_smiles_from_seq2seq(
                                    decoder_outputs.cpu(), self.
                                    SMILES_ds)])

corrected = []
for i, sm in enumerate(attempted_to_correct):
    if is_valid_smiles(sm):
        corrected.append(sm)
        smiles2indx[sm] = smiles2indx[invalid[i]]
        del smiles2indx[invalid[i]]
unique_valid = [sm for sm in valid if is_new_smiles(sm, self.
    SMILES_ds)]
unique_corrected = [sm for sm in corrected if is_new_smiles(sm,
    self.SMILES_ds) and sm not in unique_valid] # TODO: add chem
DBs check if unique

all_new_smiles = deepcopy(unique_valid)
all_new_smiles.extend(unique_corrected)

return all_new_smiles, smiles2indx

def write_results_to_file(self, smiles, predictions, smiles2indx):
    with open('smiles_with_predicted_properties.csv', 'w') as f:
        f.write(','.join(['SMILES', ', '.join(['preds_{}'.format(i) for
            i in range(len(self.predictors))])]))
        f.write('\n')
        for i, sm in enumerate(smiles):
            if smiles2indx:
                preds = [predictions[j][smiles2indx[sm]] for j in
                    range(len(self.predictors))]
            else:
                preds = [predictions[j][i] for j in range(len(self.
                    predictors))]

            f.write(','.join([sm, ', '.join(map(str, preds))]))
            f.write('\n')

def write_embeddings_to_file(self, smiles, embds, filepath):
    with open(filepath, 'w') as f:
        for sm, embd in zip(smiles, embds):
            f.write(','.join([sm, ', '.join(map(lambda x: str(x.item()),
                embd))]))
            f.write('\n')

```

xv Розробка молекул-кандидатів у лікарські речовини

```

from pipeline.pipeline import Pipeline

import torch

```

```

import yaml
import warnings
import time

warnings.filterwarnings('ignore')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

if __name__ == '__main__':
    st = time.time()
    with open('pipeline/pipeline_config.yaml') as f:
        config = yaml.load(f)
        config['device'] = device

    p = Pipeline(config)

    embds = p.smiles_to_embeddings()

    new_embds = p.generate_new_embeddings(embds)
    properties = p.predict_properties(new_embds)

    new_smiles_one_hot = p.embeddings_to_smiles(new_embds)

    new_correct_smiles, smiles2indx = p.correct_smiles(
        new_smiles_one_hot)

    p.write_results_to_file(new_correct_smiles, properties,
        smiles2indx)

    end = time.time() - st
    print('Generated {} new SMILES in {:.2f}s'.format(len(
        new_correct_smiles), end))

```

xvi Прогнозування властивостей молекул-кандидатів у лікарські речовини

```

from pipeline.pipeline import Pipeline

import torch

import yaml
import warnings
import time

warnings.filterwarnings('ignore')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

if __name__ == '__main__':
    st = time.time()
    with open('pipeline/pipeline_config.yaml') as f:
        config = yaml.load(f)
        config['device'] = device

    p = Pipeline(config)

    embds = p.smiles_to_embeddings()

    properties = p.predict_properties(embds)

    p.write_results_to_file(p.SMILES_ds.smiles_strings, properties,
        {})

```

```
end = time.time() - st
print('Predicted properties for {} structures in {:.2f}s'.format(
    len(embds), end))
```