OUKA
The University of Osaka
Institutional Knowledge Archive

Title	Enhancing QSAR-based Predictions of Natural Product Bioactivity through Multi-Task Learning with Protein Evolutionary-Relatedness Metrics
Author(s)	Donny, Ramadhan
Citation	大阪大学, 2026, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/105652
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

Enhancing QSAR-based Predictions of Natural Product Bioactivity through Multi-Task Learning with Protein Evolutionary-Relatedness Metrics

A Doctoral Thesis Submitted for the Degree of Doctor of Philosophy

Donny Ramadhan

Department of Chemistry

Graduate School of Science

The University of Osaka

2026

Abstract

Natural products possess diverse and generally three-dimensional structures that may play crucial roles in their interactions with drug targets. Considering the scarcity of natural product bioactivity data in public databases, multi-task learning (MTL) emerges as a valuable approach to enhance quantitative structure-activity relationship (QSAR) prediction performance. This study aimed to enhance MTL by incorporating protein evolutionary-relatedness metrics to improve the natural product bioactivity prediction when data are limited, while also determining the conditions in which MTL achieves optimal performance. Using binary classification filtering, a curated dataset of predicted natural products with bioactivity toward enzymes from ChEMBL was constructed. Single-task learning (STL) was used as the baseline model, while feature-based MTL (FBMTL) was trained on all proteins within each group. Instance-based MTL (IBMTL), which extends FBMTL, incorporated protein evolutionary-relatedness metrics as additional features. The results indicated that IBMTL consistently surpassed STL and FBMTL in most protein groups, highlighting the benefit of including protein evolutionary relatedness. Significant improvements were observed in the kinase and cytochrome P450 groups, where proteins are categorized into specific levels within ChEMBL's six-level protein classification. Within the kinase group, IBMTL achieved its best performance at the target parent level, suggesting a trade-off between dataset size and protein evolutionary relatedness. Overall, these findings demonstrate that integrating protein evolutionary information into MTL is a valuable strategy for natural-product drug discovery, even under data-limited conditions.

Keywords: protein evolutionary relatedness, feature-based multi-task learning (FBMTL), instance-based multi-task learning (IBMTL), single-task learning (STL)

Table of Contents

Abstract 1

Table of Contents 2

List of Figures 4

List of Tables 6

List of Abbreviations 7

Originality 9

Chapter 1 Introduction 10

1.1. The Importance of Natural Products in Drug Discovery 10

1.2. Computer-Aided Drug Design (CADD) in Natural Product Drug Discovery 12

1.3. Learning Paradigms: Single-Task and Multi-Task Learning 15

1.4. Protein Kinase B Beta (AKT2) 18

1.5. Research Objectives 19

1.6. References 21

Chapter 2 Methods 25

2.1. Construction of Predicted Natural Product Datasets 25

2.2. Dataset Curation and Preprocessing 29

2.3. Single-Task and Multi-Task Regression Models 31

2.4. Model Construction, Hyperparameter Optimization, and Evaluation 34

2.5. Statistical Analysis 37

2.6. Studies of MTL Models over Three Hierarchical Protein Group Levels 38

2.7. Identification of Potential AKT2 Inhibitors through Virtual Screening of Natural Product Databases 39

2.8. Comparative Evaluation of Virtual Screening Results with Protein-Ligand Docking 40
2.9. References 42
Chapter 3 Results and Discussion 46
3.1. Machine Learning-Based Identification of Natural Products and Dataset Characterization 46
3.2. Bioactivity Predictive Performance Comparison of STL and MTL across Protein Groups in the Enzyme Family 51
3.3. Effect of Three Hierarchical Protein Group Levels on MTL Performance in Kinase and Protease Groups 54
3.4. Effects of Protein Similarity, Protein Count, and Bioactivity Data Availability on MTL Performance at the Target Parent Level in the Kinase Group 57
3.5. Virtual Screening of Natural Products Against AKT2 Using the Best-Performing Predictive Model and Docking-Based Comparative Analysis 59
3.6. References 64
Chapter 4 Conclusion 65
Limitations of the Study 67
Acknowledgements 68
List of Publications 69
Supplementary Materials 71

List of Figures

Figure 1.1. Proportion of small-molecule approved drugs by class between 1 January 1981 and 30 September 2019 (n = 1394).....11

Figure 1.2. Distribution of synthetic compounds, natural products, and hybrid compounds across clinical trial phases.....12

Figure 1.3. Classification of computer-aided drug design based on ligand and protein structure availability.....13

Figure 1.4. Schematic illustration of single-task learning (STL) (a) and multi-task learning (MTL) (b).....16

Figure 1.5. Distribution of global cancer mortality by cancer type in 2019. 18

Figure 1.6. Five-year relative survival rates by cancer type in the United States between 2013 and 2019.....19

Figure 2.1. (a) Overview of data acquisition and preprocessing workflow. (b) Illustration of STL and MTL models applied to protein groups within the enzyme family. (c) Illustration of MTL models across three hierarchical protein group levels: protein class, target parent, and superclass. (d) Overview of the virtual screening workflow for natural products from public databases using the best-performing AKT2 model to estimate bioactivities.....26

Figure 2.2. Five-fold cross-validation for model training and evaluation. 28

Figure 2.3. Six-level hierarchical classification of proteins in the ChEMBL database. 30

Figure 2.4. Overview of data frame formulations used in single-task learning (a), conventional multi-task learning (b), feature-based multi-task learning (c), and instance-based multi-task learning (d).....32

Figure 2.5. Nested cross-validation scheme for hyperparameter optimization and performance evaluation.....38

Figure 3.1. Evaluation of machine learning models for compound classification: (a) accuracy of models combining various molecular fingerprints and machine learning algorithms, (b) sensitivity of the best-performing classification model in identifying natural products across internal and external validation datasets. 47

Figure 3.2. Comparison of the chemical space of natural compounds, synthetic compounds, and predicted natural compounds based on t-SNE analysis. 48

Figure 3.3. Protein distribution across all protein groups within the enzyme family, before and after data preprocessing. 50

Figure 3.4. Protein-to-bioactivity data distribution per protein across all protein groups within the enzyme family after data preprocessing.....50

Figure 3.5. Bioactivity data distribution per protein group within the enzyme family after data preprocessing. 51

Figure 3.6. Evaluation of three regression learning algorithms (ANN, SVM, and RF) under STL and MTL settings for proteins in the kinase group. 52

Figure 3.7. Evaluation of STL, FBMTL, IBMTL AA-GSS, IBMTL AA-LSS, and BMTL predictive performance across all protein groups under the enzyme family.....53

Figure 3.8. Evaluation of STL, FBMTL, IBMTL AA-GSS, and IBMTL AA-LSS predictive performance across superclass, target parent, and protein class levels in the kinase group (a) and protease group (b). 55

Figure 3.9. Evaluation of STL, FBMTL, and IBMTL AA-LSS predictive performance at the target parent level in the kinase group, evaluated by (a) average protein local sequence similarity, (b) protein count, and (c) compound bioactivity data availability. 58

Figure 3.10. Overlay of the crystallographic pose (blue) and the best-ranked redocked pose (green) of the co-crystallized ligand (GSK690693). Redocking using (a) AutoDock Vina resulted in an RMSD of 0.22 Å, while (b) MOE produced an RMSD of 0.978 Å.....61

List of Tables

Table 2.1. Molecular fingerprints generated using RDKit and CDK.....27

Table 2.2. Hyperparameter settings applied in nested cross-validation for STL, FBMTL, and IBMTL.....36

Table 2.3. Hyperparameter settings applied in nested cross-validation for BMTL.....37

Table 3.1. Evaluation of STL, FBMTL, IBMTL AA-GSS, and IBMTL AA-LSS predictive performance for AKT2 across two hierarchical protein group levels.....60

Table 3.2. Top ten natural products with the lowest predicted IC_{50} values targeting AKT2 screened using the best-performing model, together with reference compounds and their protein-ligand docking binding affinities.....63

List of Abbreviations

1-NN	:	1-nearest neighbor
AA-GSS	:	amino acid global sequence similarity
AA-LSS	:	amino acid local sequence similarity
AI	:	artificial intelligence
AKT2	:	AKT serine/threonine kinase 2
ANN	:	artificial neural network
BMTL	:	baseline multi-task learning
CADD	:	computer-aided drug design
CDK	:	chemistry development kit
CMNPD	:	comprehensive marine natural products database
FBMTL	:	feature-based multi-task learning
GBVI/WSA	:	generalized-born volume integral/weighted surface area
IBMTL	:	instance-based multi-task learning
IC_{50}	:	50% inhibitory concentration
KNIME	:	Konstanz information miner
LBDD	:	ligand-based drug design
MFP	:	molecular fingerprint
MOE	:	molecular operating environment
MOPAC	:	molecular orbital package
MSE	:	mean squared error
MTL	:	multi-task learning
NP	:	natural product

NPASS	:	natural product activity and species source
PDB	:	protein data bank
PLB	:	propensity for ligand binding
QSAR	:	quantitative structure-activity relationship
RF	:	random forest
RMSD	:	root mean square deviation
RMSE	:	root mean squared error
SBDD	:	structure-based drug design
SMILES	:	simplified molecular input line entry system
STL	:	single-task learning
SVM	:	support vector machine
TPE	:	tree-structured Parzen estimator
t-SNE	:	t-distributed stochastic neighbor embedding

Originality

The content of this doctoral thesis is derived from the findings and descriptions reported in the following scientific article:

Ramadhan, D., Watanabe, R., & Mizuguchi, K. (2025). Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. *ACS Omega*, *10*(37), 42776–42790. https://doi.org/10.1021/acsomega.5c05094

Chapter 1

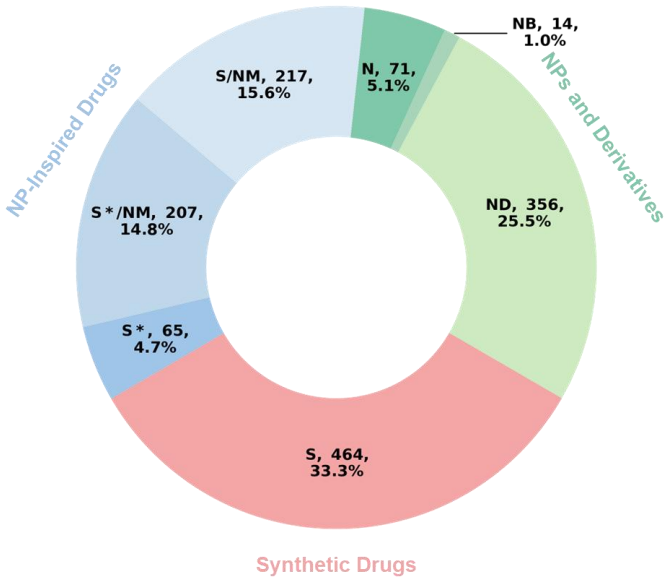
Introduction

1.1. The Importance of Natural Products in Drug Discovery

Natural products refer to metabolites originating from living organisms, including plants, animals, microorganisms, and marine species (Abdel-Razek et al., 2020). Plant-derived natural products, in particular, have been utilized by humans since ancient times to treat various illnesses and diseases (Li et al., 2022). The earliest evidence of the medicinal use of natural products appears in cuneiform clay tablets from Mesopotamia (2600 B.C.), which describe oils derived from *Cupressus sempervirens* (cypress) and *Commiphora* species (myrrh), substances that are still used today to treat coughs, colds, and inflammation (Dias et al., 2012). Well-known drugs derived from natural products include morphine from the opium poppy (*Papaver somniferum* L.) and salicylic acid from the willow tree (*Salix alba*), which led to the development of two major drug classes: analgesics and antipyretics (Shilpi & Uddin, 2020).

Natural products continue to be a valuable source for drug discovery due to their high degree of three-dimensionality compared to synthetic compounds, which may be crucial for interacting with challenging drug targets (Mullowney et al., 2023). Their relatively greater rigidity can also be advantageous for effectively modulating protein-protein interactions. Moreover, natural products exhibit exceptional scaffold diversity and structural complexity, resulting in libraries enriched with bioactive compounds that occupy a broader chemical space than conventional synthetic collections (Atanasov et al., 2021). In addition, as naturally occurring metabolites, they are likely to serve as substrates for transporter systems, thereby facilitating access to their biological targets (Mullowney et al., 2023).

Figure 1.1 shows the distribution of all small-molecule approved drugs from 1981 to 2019. Although the proportion of small-molecule approved drugs derived from unaltered natural products is relatively small (5.1%), more than half (66.7%) are linked to natural products, highlighting their potential in drug discovery (Newman & Cragg, 2020).



Code	Definition
N	Unaltered Natural Product
NB	Botanical Drug (Defined Mixture)
ND	Natural Product Derivative
S	Synthetic Drug
S*	Synthetic Drug (NP Pharmacophore)
/NM	Mimic of Natural Product

Figure 1.1. Proportion of small-molecule approved drugs by class between 1 January 1981 and 30 September 2019 (n = 1394) (Newman & Cragg, 2020).

Domingo-Fernández et al. reported that compounds derived from natural products (NP-likeness > 0.6) and hybrid molecules (0 ≤ NP-likeness ≤ 0.6) become increasingly prevalent as candidates progress through successive clinical trial phases, whereas the relative contribution of synthetic compounds (NP-likeness < 0) declines consistently, as shown in Figure 1.2. Specifically, synthetic molecules accounted for 65% (3085/4749) of compounds entering phase I, but this proportion fell to 55.5% (1863/3356) by phase III. In contrast, the share of natural products rose from approximately 20% (940/4749) in phase I to around 26% (860/3356) in phase III, while hybrid compounds increased from roughly 15% (724/4749) to 19% (632/3356)

over the same progression. Collectively, natural products and hybrids represented about 45% of phase III compounds, a distribution comparable to that observed among approved drugs. This pattern may be attributed to the lower observed and predicted toxicity of natural products compared to synthetic molecules, indicating that emphasizing natural products could lead to higher success rates in clinical trials (Domingo-Fernández et al., 2024).

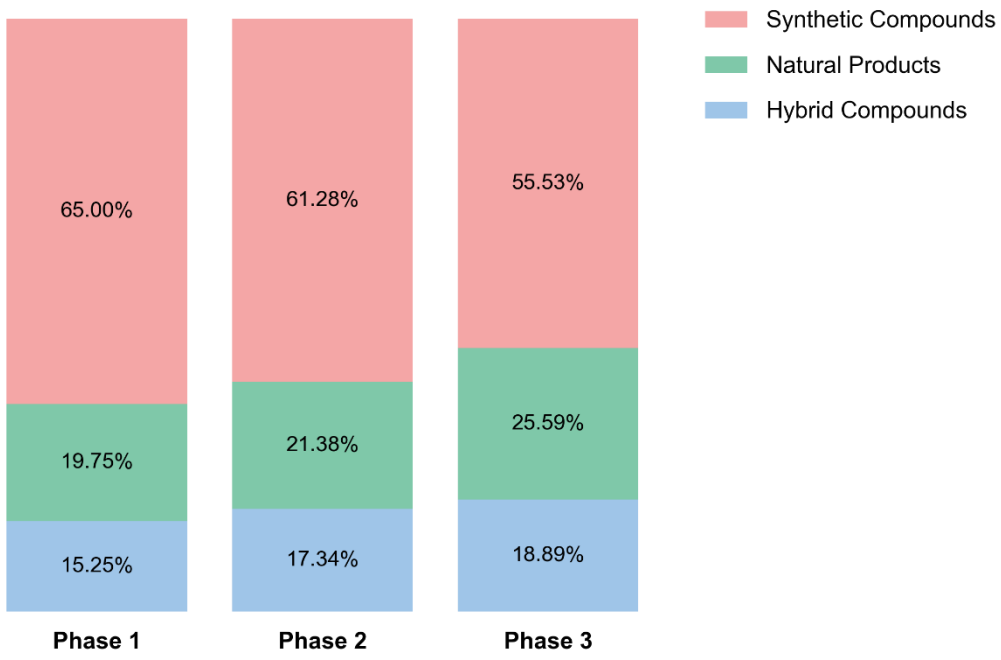


Figure 1.2. Distribution of synthetic compounds, natural products, and hybrid compounds across clinical trial phases (Domingo-Fernández et al., 2024).

1.2. Computer-Aided Drug Design (CADD) in Natural Product Drug Discovery

Currently, more than 250000 known natural product structures are accessible through public databases (Chen et al., 2018). Experimentally evaluating such a vast number of compounds against specific diseases in wet laboratories requires substantial resources, high costs, and considerable time (Wei & McCammon, 2024). Computer-aided drug design (CADD) addresses these limitations by prioritizing the most promising candidate compounds, thereby

reducing time and cost in chemical and biological testing. CADD encompasses a broad range of theoretical and computational methods used in modern drug discovery, including molecular modeling, bioinformatics, cheminformatics, and related fields (Medina-Franco et al., 2022; Sharma et al., 2023).

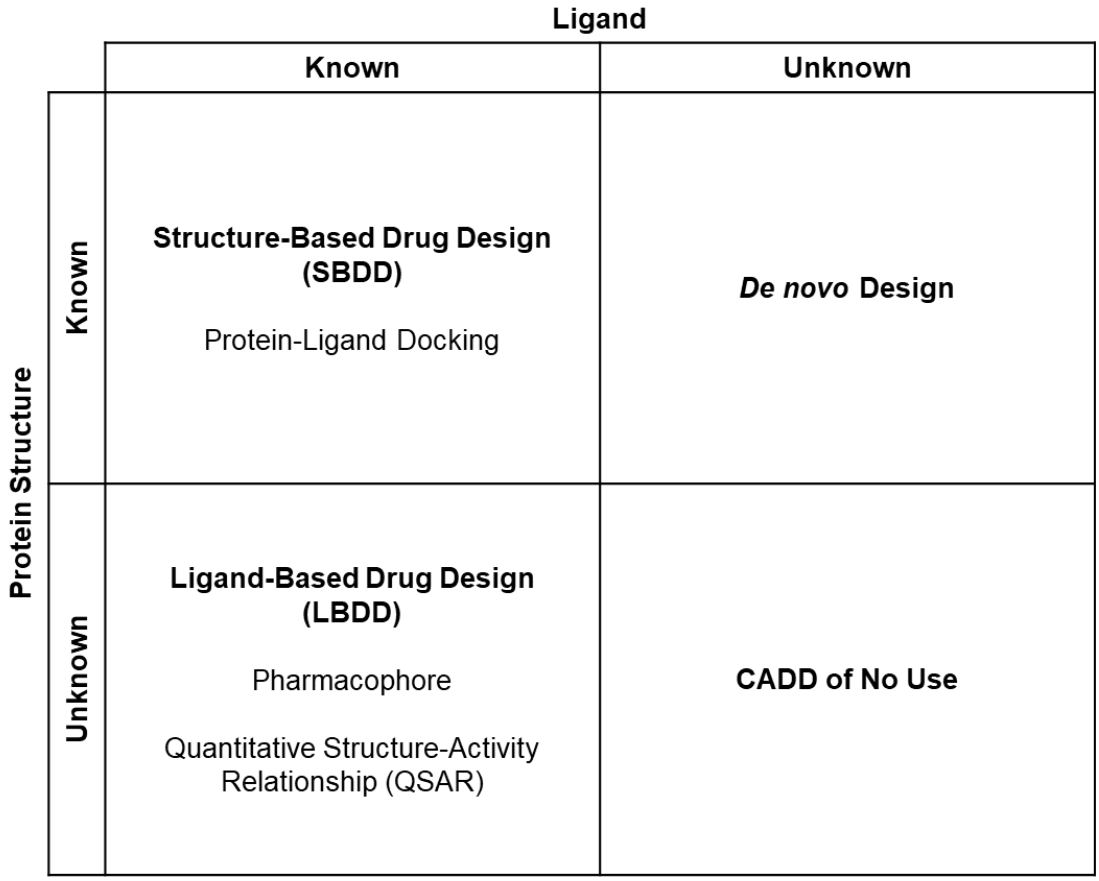


Figure 1.3. Classification of computer-aided drug design based on ligand and protein structure availability (Arora & Malik, 2021).

Structure-based drug design (SBDD) and ligand-based drug design (LBDD) are the most frequently used techniques in CADD. As depicted in Figure 1.3, SBDD relies on the availability of three-dimensional structures of the target proteins, whereas LBDD is applied when such structural information is not available (Wei & McCammon, 2024). Among CADD techniques,

protein-ligand docking has gained broad popularity and refers to a computational method that predicts how small molecules (ligands) interact with a target protein (Chen et al., 2023).

Although protein-ligand docking is widely used in virtual drug screening, it has substantial limitations, including dependence on high-quality protein structures, limited treatment of protein flexibility and solvation effects, inability to properly account for induced-fit effects, and the poor ranking performance of scoring functions. In addition, its speed remains computationally inefficient for screening million-scale compound libraries. In contrast, LBDD overcomes these challenges by operating independently of target protein structures and instead exploiting known active ligands to identify key structural features responsible for biological activity. As a result, LBDD methods, including pharmacophore modeling and quantitative structure-activity relationship (QSAR), remain broadly applicable and effective even when target protein structures are unavailable (Mao et al., 2021).

QSAR is a computational approach that develops mathematical models to identify statistically meaningful relationships between chemical structures and their biological activities using chemometric methods (Vasilev & Atanasova, 2025). Its paradigm can be expressed as:

$$A = f(x_1, x_2, \dots, x_P) \tag{1.1}$$

where A denotes physicochemical property or bioactivity of a compound, modeled as a function (f) of its structural features represented by P molecular descriptors (Valsecchi et al., 2023).

QSAR enables large compound libraries to be analyzed rapidly and efficiently while maintaining high predictive accuracy (Mao et al., 2021). In recent years, artificial intelligence (AI) has significantly transformed CADD by enabling more efficient and accurate predictive modeling, virtual screening, and molecular optimization compared with traditional approaches

(Koirala et al., 2025). Traditional QSAR methods, which were historically based on linear and statistical techniques, have been substantially enhanced by machine learning and deep learning that can model complex, nonlinear relationships between molecular structures and biological responses with greater fidelity (Pathak et al., 2025).

However, the predictive performance and accuracy of QSAR models are strongly influenced by the size of the dataset and the quality of the underlying chemical and biological data (Mansouri et al., 2024; Rácz et al., 2021). Currently, a real-world challenge in applying QSAR to natural product bioactivity prediction is that only a few public databases provide biological data for natural products targeting specific proteins, and the available datasets are generally insufficient to support machine learning-based QSAR modeling.

1.3. Learning Paradigms: Single-Task and Multi-Task Learning

One of the most significant challenges in developing predictive models of chemical effects is the scarcity of available data. Model performance and generalization ability can be enhanced by incorporating data from relevant tasks and learning them simultaneously, a strategy referred to as multi-task learning (MTL) (Lin et al., 2024). As illustrated in Figure 1.4, unlike conventional single-task learning (STL), which trains one task independently and treats different tasks as unrelated, MTL exploits knowledge shared across related tasks, making it effective for data-scarce problems (Zhao et al., 2020).

For a regression problem, a single-task dataset comprises a fixed number of N objects. Each object is described by P-descriptor vector x_i and an associated continuous response for the t-th task to be estimated (y_i^t). Its dataset for the t-th task can be expressed as:

$$S = \{(x_i, y_i^t)\}_{i=1}^{N} \tag{1.2}$$

with $x_i \in R^P$ and $y_i \in R$.

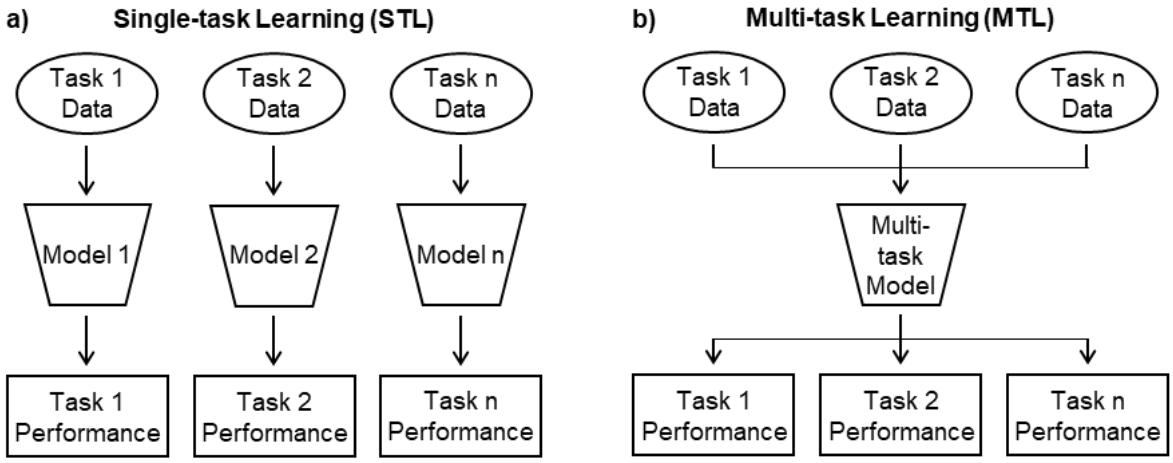


Figure 1.4. Schematic illustration of single-task learning (STL) (a) and multi-task learning (MTL) (b).

On the other hand, a multi-task dataset comprises a fixed number of N objects, each described by P variables and associated with a continuous response vector across T tasks. Its dataset can be expressed as:

$$S = \{S_1, S_2, \dots, S_T\} \tag{1.3}$$

where the dataset for the t-th task is:

$$S_t = \{(x_i, y_i^t)\}_{i=1}^{n_t} \tag{1.4}$$

with $x_i \in R^P$ and $y_i^t \in R$, whereas t denotes the task index with $t \in [1, \dots, T]$, and n_t denotes the number of objects labeled for the task t. Under this formulation, the input space at $(N \times P)$

and output space at $(N \times T)$ are assumed, with all tasks sharing the same variable space and allowing missing annotations (Valsecchi et al., 2023).

The underlying idea in MTL is that combining data from multiple tasks increases the dataset size for each task, thereby improving learning performance. However, the naive inclusion of all source tasks can result in worse performance than STL when some source tasks are unrelated to the target task, a phenomenon known as negative transfer (Li et al., 2023). Consequently, task-relatedness is a fundamental requirement for MTL, as knowledge learned from unrelated tasks cannot be expected to benefit another task if there is no meaningful overlap or shared characteristics between them (Ben-David & Schuller, 2003; Wenzel et al., 2019).

MTL can be categorized into traditional MTL, feature-based MTL (FBMTL), and instance-based MTL (IBMTL). In traditional MTL, data are organized in a response matrix where rows correspond to objects and columns to tasks, resulting in missing values when objects lack annotations for certain tasks. The setting prevents object duplication and maintains balanced data representation, but typically requires learning algorithms, such as artificial neural networks (ANN), that can accommodate missing values during model training. FBMTL transforms the problem by introducing a task identifier as an additional feature, allowing the use of a single shared response without missing values. IBMTL further extends this framework by integrating task-similarity measures as extra features, thereby quantitatively encoding relationships among tasks. Both FBMTL and IBMTL replicate objects across tasks for which they are annotated, making them compatible with conventional single-output algorithms such as random forest (RF) and support vector machines (SVM) (Sadawi et al., 2019; Valsecchi et al., 2023).

Sadawi et al. reported that IBMTL, which incorporates protein evolutionary relatedness as features, achieved superior performance compared with FBMTL and STL in QSAR studies

(Sadawi et al., 2019). However, neither this study nor previous work has systematically examined the conditions, particularly protein evolutionary relatedness, the number of proteins, and the availability of compound bioactivity data, that favor FBMTL and IBMTL performance, especially in natural product bioactivity prediction.

1.4. Protein Kinase B Beta (AKT2)

To evaluate the best-performing model in this study, protein kinase B beta (AKT2) is selected as a case study to assess whether the model could identify potential compounds from a large natural product library and whether its predictions align with results obtained using the widely used virtual screening approach, protein-ligand docking. AKT2 was chosen because it is frequently overexpressed in pancreatic cancer, which is the fifth leading cause of cancer-related mortality (2019) (Global Change Data Lab, 2025), and has a poor 5-year relative survival rate of only 12.5% (2013-2019) (National Cancer Institute, 2025), as shown in Figures 1.5 and 1.6, respectively, making it one of the most lethal cancer types (He et al., 2021).

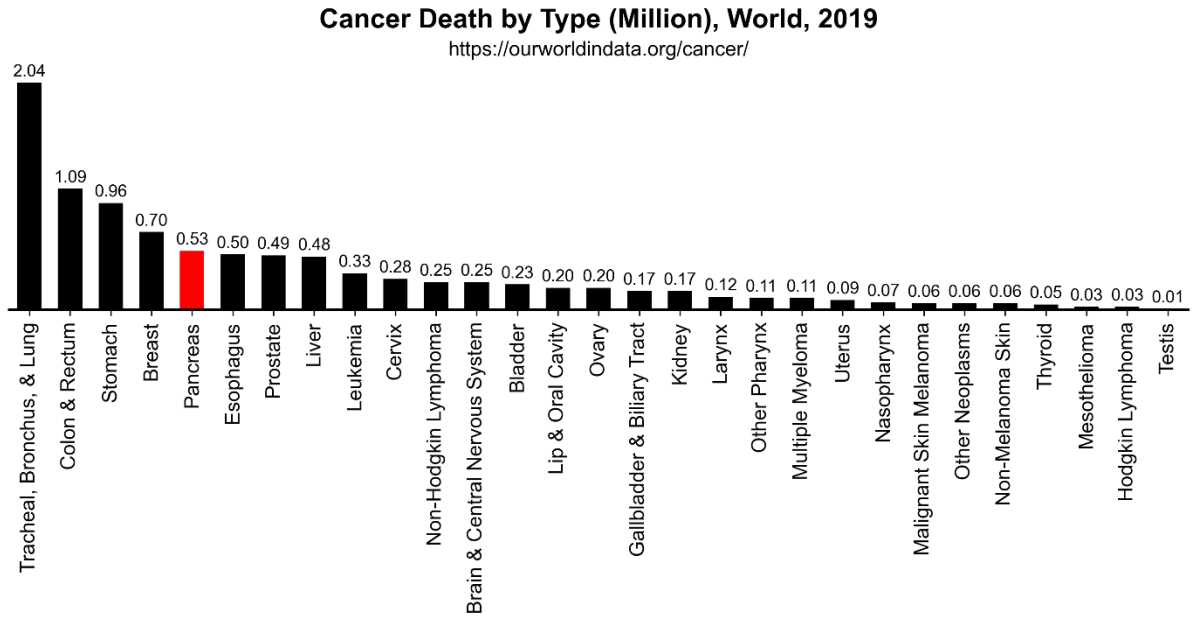


Figure 1.5. Distribution of global cancer mortality by cancer type in 2019.

Accordingly, the AKT signaling pathway may be a promising therapeutic target for pancreatic cancer. In addition, AKT2 is a member of the kinase group, for which IBMTL exhibited significantly better performance than STL, supporting its selection as a representative target for evaluation.

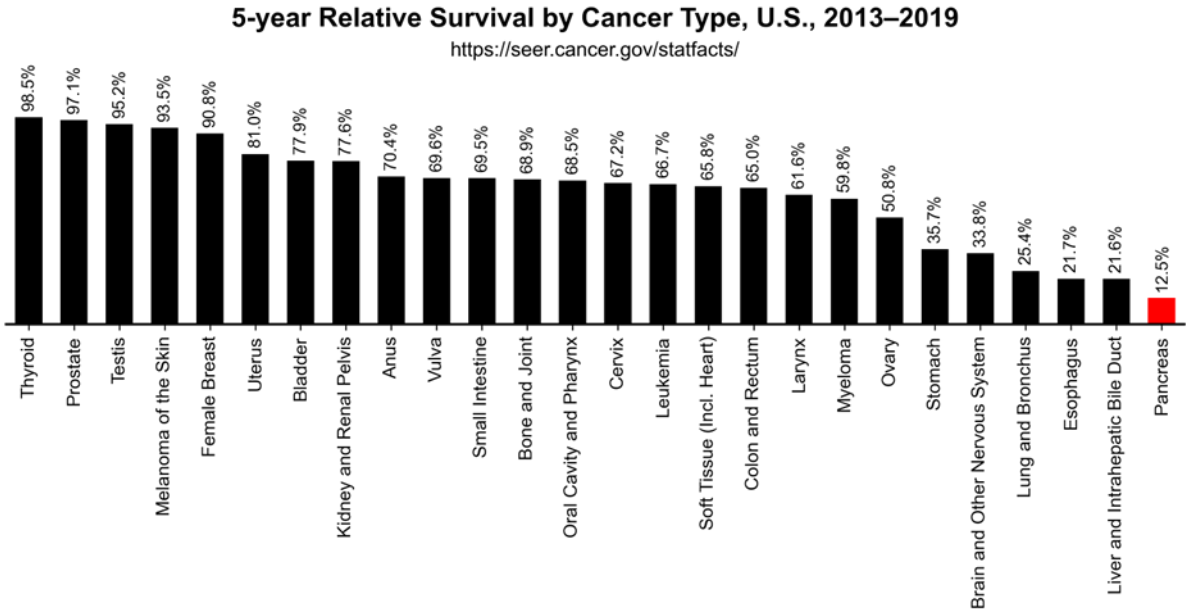


Figure 1.6. Five-year relative survival rates by cancer type in the United States between 2013 and 2019.

1.5. Research Objectives

Based on the background and motivation described above, this study is conducted with the following objectives:

1. Optimizing MTL with protein evolutionary relatedness metrics to enhance the prediction of natural product activity against target proteins, particularly when working with limited datasets.
2. Investigating the conditions (evolutionary relatedness of target proteins, number of proteins, and amount of bioactivity data) that are favorable for MTL models.

3. Applying the best-performing model to predict the activity of natural products against AKT2, a pancreatic cancer-associated protein.

1.6. References

Abdel-Razek, A. S., El-Naggar, M. E., Allam, A., Morsy, O. M., & Othman, S. I. (2020). Microbial Natural Products in Drug Discovery. *Processes*, *8*(4). https://doi.org/10.3390/pr8040470

Arora, T., & Malik, A. A. (2021). Chapter 14 - An introduction to BLAST: applications for computer-aided drug design and development. In N. Sharma, H. Ojha, P. K. Raghav, & R. k. Goyal (Eds.), *Chemoinformatics and Bioinformatics in the Pharmaceutical Sciences* (pp. 423–453). Academic Press. https://doi.org/10.1016/B978-0-12-821748-1.00015-4

Atanasov, A. G., Zotchev, S. B., Dirsch, V. M., Orhan, I. E., Banach, M., Rollinger, J. M., Barreca, D., Weckwerth, W., Bauer, R., Bayer, E. A., Majeed, M., Bishayee, A., Bochkov, V., Bonn, G. K., Braidy, N., Bucar, F., Cifuentes, A., D'Onofrio, G., Bodkin, M., … Supuran, C. T. (2021). Natural products in drug discovery: advances and opportunities. In *Nature Reviews Drug Discovery* (Vol. 20, Issue 3, pp. 200–216). Nature Research. https://doi.org/10.1038/s41573-020-00114-z

Ben-David, S., & Schuller, R. (2003). Exploiting Task Relatedness for Multiple Task Learning. In B. Schölkopf & M. K. Warmuth (Eds.), *Learning Theory and Kernel Machines* (pp. 567–580). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-45167-9_41

Chen, T., Shu, X., Zhou, H., Beckford, F. A., & Misir, M. (2023). Algorithm selection for protein–ligand docking: strategies and analysis on ACE. *Scientific Reports*, *13*(1), 8219. https://doi.org/10.1038/s41598-023-35132-5

Chen, Y., Garcia de Lomana, M., Friedrich, N.-O., & Kirchmair, J. (2018). Characterization of the Chemical Space of Known and Readily Obtainable Natural Products. *Journal of Chemical Information and Modeling*, *58*(8), 1518–1532. https://doi.org/10.1021/acs.jcim.8b00302

Dias, D. A., Urban, S., & Roessner, U. (2012). A Historical Overview of Natural Products in Drug Discovery. *Metabolites*, *2*(2), 303–336. https://doi.org/10.3390/metabo2020303

Domingo-Fernández, D., Gadiya, Y., Preto, A. J., Krettler, C. A., Mubeen, S., Allen, A., Healey, D., & Colluru, V. (2024). Natural Products Have Increased Rates of Clinical Trial Success throughout the Drug Development Process. *Journal of Natural Products*, *87*(7), 1844–1851. https://doi.org/10.1021/acs.jnatprod.4c00581

Global Change Data Lab. (2025). *Cancer Death by Type, World, 2019*. https://ourworldindata.org/cancer/

He, Y., Sun, M. M., Zhang, G. G., Yang, J., Chen, K. S., Xu, W. W., & Li, B. (2021). Targeting PI3K/Akt signal transduction for cancer therapy. *Signal Transduction and Targeted Therapy*, *6*(1), 425. https://doi.org/10.1038/s41392-021-00828-5

Koirala, M., Yan, L., Mohamed, Z., & DiPaola, M. (2025). AI-Integrated QSAR Modeling for Enhanced Drug Discovery: From Classical Approaches to Deep Learning and Structural Insight. *International Journal of Molecular Sciences*, *26*(19). https://doi.org/10.3390/ijms26199384

Li, D., Nguyen, H. L., & Zhang, H. R. (2023). Identification of negative transfers in multitask learning using surrogate models. *ArXiv Preprint ArXiv:2303.14582*. https://doi.org/10.48550/arXiv.2303.14582

Li, Z., Chen, K., Rose, P., & Zhu, Y. Z. (2022). Natural products in drug discovery and development: Synthesis and medicinal perspective of leonurine. *Frontiers in Chemistry*, *Volume 10-2022*. https://doi.org/10.3389/fchem.2022.1036329

Lin, R.-H., Lin, P., Wang, C.-C., & Tung, C.-W. (2024). A novel multitask learning algorithm for tasks with distinct chemical space: zebrafish toxicity prediction as an example. *Journal of Cheminformatics*, *16*(1), 91. https://doi.org/10.1186/s13321-024-00891-4

Mansouri, K., Moreira-Filho, J. T., Lowe, C. N., Charest, N., Martin, T., Tkachenko, V., Judson, R., Conway, M., Kleinstreuer, N. C., & Williams, A. J. (2024). Free and open-source QSAR-ready workflow for automated standardization of chemical structures in support of QSAR modeling. *Journal of Cheminformatics*, *16*(1), 19. https://doi.org/10.1186/s13321-024-00814-3

Mao, J., Akhtar, J., Zhang, X., Sun, L., Guan, S., Li, X., Chen, G., Liu, J., Jeon, H.-N., Kim, M. S., No, K. T., & Wang, G. (2021). Comprehensive strategies of machine-learning-based quantitative structure-activity relationship models. *IScience*, *24*(9), 103052. https://doi.org/10.1016/j.isci.2021.103052

Medina-Franco, J. L., Flores-Padilla, E. A., & Chávez-Hernández, A. L. (2022). Chapter 23 - Discovery and development of lead compounds from natural sources using computational approaches. In P. K. Mukherjee (Ed.), *Evidence-Based Validation of Herbal Medicine*

(Second Edition) (pp. 539–560). Elsevier. https://doi.org/10.1016/B978-0-323-85542-6.00009-3

Mullowney, M. W., Duncan, K. R., Elsayed, S. S., Garg, N., van der Hooft, J. J. J., Martin, N. I., Meijer, D., Terlouw, B. R., Biermann, F., Blin, K., Durairaj, J., Gorostiola González, M., Helfrich, E. J. N., Huber, F., Leopold-Messer, S., Rajan, K., de Rond, T., van Santen, J. A., Sorokina, M., … Medema, M. H. (2023). Artificial intelligence for natural product drug discovery. In *Nature Reviews Drug Discovery* (Vol. 22, Issue 11, pp. 895–916). Nature Research. https://doi.org/10.1038/s41573-023-00774-7

National Cancer Institute. (2025). *Cancer Stat Facts*. https://seer.cancer.gov/statfacts/

Newman, D. J., & Cragg, G. M. (2020). Natural Products as Sources of New Drugs over the Nearly Four Decades from 01/1981 to 09/2019. In *Journal of Natural Products* (Vol. 83, Issue 3, pp. 770–803). American Chemical Society. https://doi.org/10.1021/acs.jnatprod.9b01285

Pathak, A., Theagarajan, R., Rizqi, M. M., Nugraha, A. S., Boruah, T., Pratiksha, Kumar, H., Naik, B., Yadav, S., Jha, A. K., Trivedi, A., Gupta, A. K., & Kumar, P. (2025). AI-enabled drug and molecular discovery: computational methods, platforms, and translational horizons. *Discover Molecules*, *2*(1), 32. https://doi.org/10.1007/s44345-025-00037-5

Rácz, A., Bajusz, D., & Héberger, K. (2021). Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification. *Molecules*, *26*(4). https://doi.org/10.3390/molecules26041111

Sadawi, N., Olier, I., Vanschoren, J., Van Rijn, J. N., Besnard, J., Bickerton, R., Grosan, C., Soldatova, L., & King, R. D. (2019). Multi-task learning with a natural metric for quantitative structure activity relationship learning. *Journal of Cheminformatics*, *11*(1). https://doi.org/10.1186/s13321-019-0392-1

Sharma, P., Sharma, K., & Nandave, M. (2023). Chapter 2 - Computational approaches in drug discovery and design. In R. K. Gautam, M. A. Kamal, & P. Mittal (Eds.), *Computational Approaches in Drug Discovery, Development and Systems Pharmacology* (pp. 53–93). Academic Press. https://doi.org/10.1016/B978-0-323-99137-7.00009-5

Shilpi, J. A., & Uddin, S. J. (2020). Chapter Fourteen - Analgesic and antipyretic natural products. In S. D. Sarker & L. Nahar (Eds.), *Annual Reports in Medicinal Chemistry* (Vol. 55, pp. 435–458). Academic Press. https://doi.org/10.1016/bs.armc.2020.03.003

Valsecchi, C., Grisoni, F., Consonni, V., Ballabio, D., & Todeschini, R. (2023). Multitask Learning for Quantitative Structure–Activity Relationships: A Tutorial. In H. Huixiao (Ed.), *Machine Learning and Deep Learning in Computational Toxicology*. https://doi.org/10.1007/978-3-031-20730-3_8

Vasilev, B., & Atanasova, M. (2025). A (Comprehensive) Review of the Application of Quantitative Structure–Activity Relationship (QSAR) in the Prediction of New Compounds with Anti-Breast Cancer Activity. *Applied Sciences*, *15*(3). https://doi.org/10.3390/app15031206

Wei, H., & McCammon, J. A. (2024). Structure and dynamics in drug discovery. *Npj Drug Discovery*, *1*(1), 1. https://doi.org/10.1038/s44386-024-00001-2

Wenzel, J., Matter, H., & Schmidt, F. (2019). Predictive Multitask Deep Neural Network Models for ADME-Tox Properties: Learning from Large Data Sets. *Journal of Chemical Information and Modeling*, *59*(3), 1253–1268. https://doi.org/10.1021/acs.jcim.8b00785

Zhao, Z., Qin, J., Gou, Z., Zhang, Y., & Yang, Y. (2020). Multi-task learning models for predicting active compounds. *Journal of Biomedical Informatics*, *108*, 103484. https://doi.org/10.1016/j.jbi.2020.103484

Chapter 2

Methods

The study followed a four-stage workflow, as illustrated in Figure 2.1. Initially, an optimized natural-product filter was constructed using binary classification models and employed to predict natural products and their associated bioactivity data from the ChEMBL database, followed by comprehensive data preprocessing. The predictive performance of STL and MTL regression models was subsequently assessed across protein groups within the ChEMBL enzyme family. Next, the MTL models were systematically evaluated across three hierarchical protein-group levels: superclass, target parent, and protein class. Finally, the best-performing model was applied to a virtual screening task targeting AKT2 to predict the bioactivities of natural products obtained from public databases.

2.1. Construction of Predicted Natural Product Datasets

Because public databases providing bioactivity data on natural products against specific proteins are limited, the ChEMBL 35 database was selected as the primary data source for this study, as it contains numerous bioactivity records, despite including both natural and synthetic compounds. A natural product filter was developed to identify compounds with a high probability of being natural products within the ChEMBL 35 database. The filter was trained using binary classification studies based on a balanced dataset comprising 10000 natural products and 10000 synthetic compounds, which were randomly sampled from the InterBioScreen database (InterBioScreen Ltd., 2025; Seo et al., 2020).

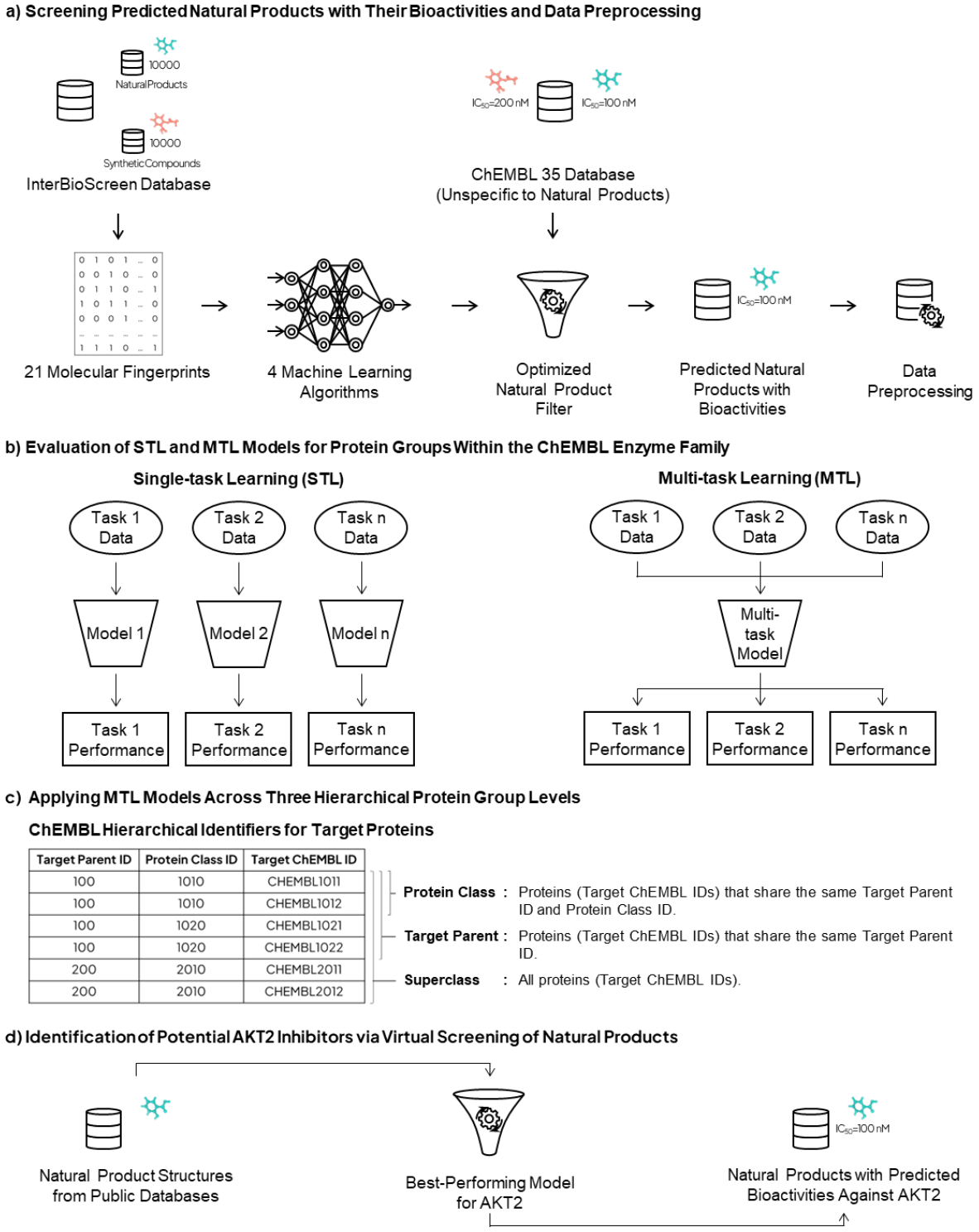


Figure 2.1. (a) Overview of data acquisition and preprocessing workflow. (b) Illustration of STL and MTL models applied to protein groups within the enzyme family. (c) Illustration of MTL models across three hierarchical protein group levels: protein class, target parent, and superclass. (d) Overview of the virtual screening workflow for natural products from public databases using the best-performing AKT2 model to estimate bioactivities.

Binary classification studies were performed using a diverse set of 21 molecular fingerprints generated with RDKit and Chemistry Development Kit (CDK), as listed in Table 2.1. Model training and evaluation were carried out using four machine learning algorithms: artificial neural network (ANN), support vector machine (SVM), random forest (RF), and 1-nearest neighbor (1-NN). To ensure reliable performance assessment and preserve class balance, stratified five-fold cross-validation was applied across all datasets, as illustrated in Figure 2.2.

Table 2.1. Molecular fingerprints generated using RDKit and CDK

Fingerprints from RDKit	Fingerprints from CDK
Morgan	Standard
FeatMorgan	Extended
AtomPair	EState
Torsion	Pubchem
RDKit	ECFP0
Avalon	ECFP2
Layered	ECFP4
MACCS	ECFP6
Pattern	FCFP0
	FCFP2
	FCFP4
	FCFP6

The molecular fingerprint-machine learning algorithm pair with the highest accuracy in discriminating between natural products and synthetic compounds was further assessed through external validation. Independent test sets comprising natural products from four databases: Natural Product Activity and Species Source Database (NPASS) (Zhao et al., 2023),

Natural Product Atlas (NP Atlas) (Poynton et al., 2025), Comprehensive Marine Natural Products Database (CMNPD) (Lyu et al., 2021), and MolPort (Molport, 2025), were used for this external validation. For each database, external validation was performed five times by randomly sampling 2000 natural products per run. Because these databases do not provide synthetic compounds as decoys, unlike the InterBioScreen dataset, sensitivity (%) was used as the evaluation metric rather than accuracy (%) to ensure a fair comparison between internal and external validations. The best-performing molecular fingerprint-machine learning algorithm pair was selected as the optimized natural-product filter and then applied to the ChEMBL 35 database to predict natural products and their associated bioactivity data against target proteins.

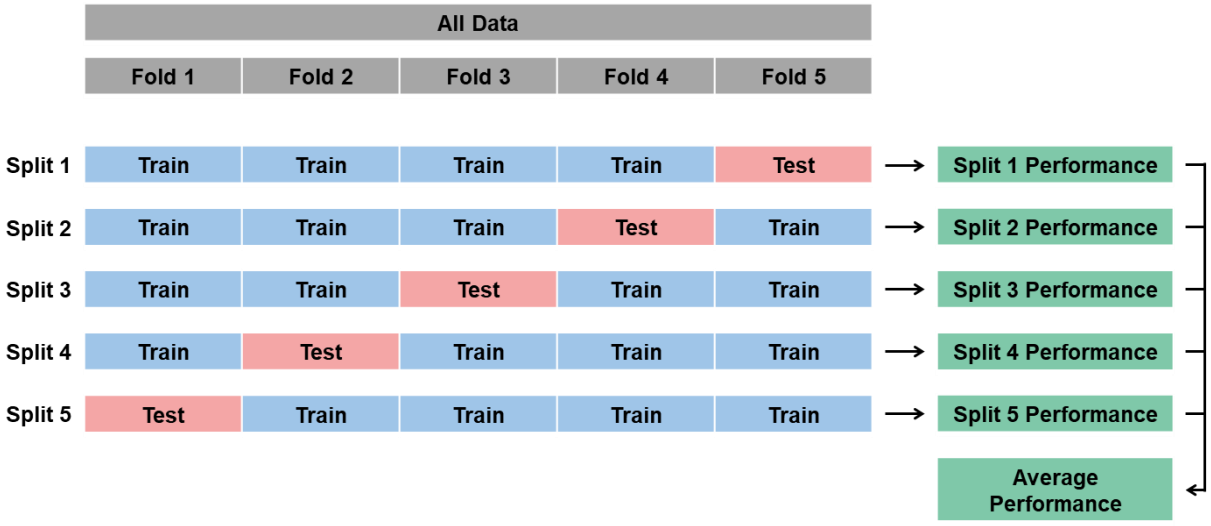


Figure 2.2. Five-fold cross-validation for model training and evaluation.

Binary classification and external validation tasks were conducted through Konstanz Information Miner (KNIME) Analytics Platform 5.2.0 (Berthold et al., 2008). In this software, Open Babel (O'Boyle et al., 2011) was utilized to convert compound file formats, whereas RDKit (RDKit, 2024) was used for multiple preprocessing steps, including format conversion,

removal of salts and small fragments, and molecular fingerprint generation. CDK (Beisken et al., 2013) was also employed for molecular fingerprint generation from compound structures.

2.2. Dataset Curation and Preprocessing

Only proteins assigned to protein groups at level 2 and lower (levels 3-6), within the enzyme family (level 1), according to ChEMBL's six-level hierarchical protein classification (Figure 2.3), were considered in this study. In addition, the dataset was restricted to single-protein targets. Compound records were further filtered to retain only those with biological activity reported as half-maximal inhibitory concentration (IC_{50}). IC_{50} values expressed in nanomolar units were directly retained, whereas values reported in other units were converted to nanomolar. Entries with zero or negative IC_{50} values, missing unit information, or relational operators other than "=" were excluded. The nanomolar IC_{50} values for each compound were subsequently converted to their logarithmic form (pIC_{50}).

At the molecular preprocessing stage, ionized molecules were uncharged using RDKit's non-forced uncharging approach (RDKit, 2025). Under this procedure, charges were removed whenever possible, and if not feasible, the opposite charge was retained to preserve overall molecular neutrality. Salt counterions that are irrelevant to the main structure associated with biological activity were excluded. Small molecular fragments were removed, keeping only the largest fragment per molecule to ensure that the modeling was based on the primary chemical structure. Simplified molecular input line entry system (SMILES) normalization was applied by transforming each structure into canonical SMILES, thereby providing uniform atom ordering and aromaticity representation (Hermansyah et al., 2021).

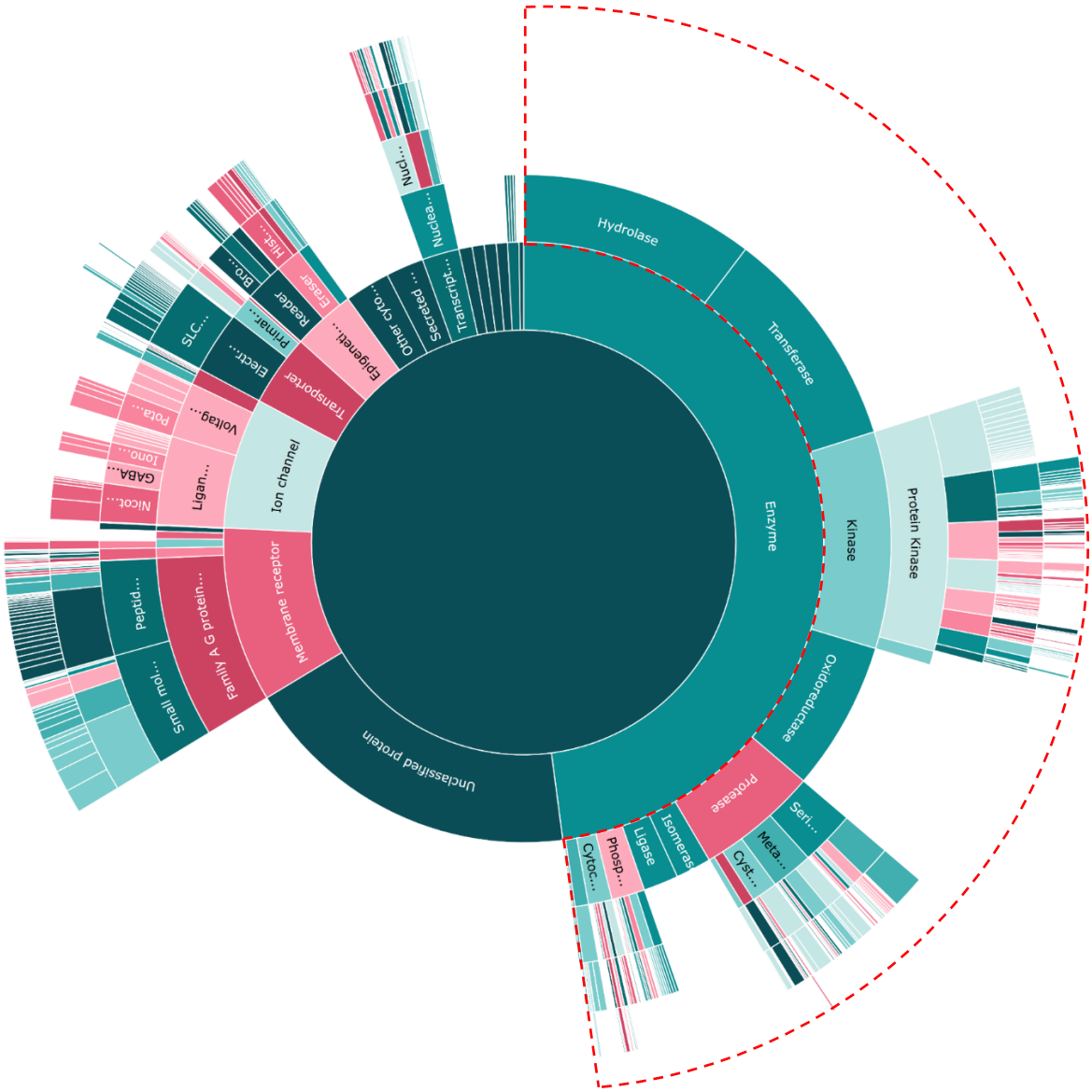


Figure 2.3. Six-level hierarchical classification of proteins in the ChEMBL database. The red dashed line indicates the region corresponding to protein groups within the enzyme family used in this study. The figure was adapted from ChEMBL (https://www.ebi.ac.uk/chembl/visualise).

When multiple IC_{50} measurements were available for the same compound-protein pair, only the most recent entry based on the reporting year was retained (Kausar & Falcao, 2018). The dataset was further filtered to include only protein groups consisting of at least two proteins, with each protein associated with a minimum of 50 predicted natural products for

which IC_{50} values are available (Zhao et al., 2020). The predicted natural products, represented as SMILES (Weininger, 1988), were subsequently transformed into molecular fingerprints using the fingerprint type that showed the best performance in the preceding binary classification task.

Data preprocessing was performed using Python 3.10.13. Pandas 1.5.3 was utilized for tabular data handling, including data import, filtering, removal of duplicate records, and sorting operations. NumPy 1.26.2 was used to carry out numerical transformations, such as converting IC_{50} values into logarithmic form. Molecular structure processing was conducted with RDKit, which supported compound neutralization, removal of salts and small fragments, SMILES standardization, and generation of molecular fingerprints.

Following preprocessing, the predicted natural products, along with the natural products and synthetic compounds used to construct the binary classification model, were projected into chemical space using t-distributed stochastic neighbor embedding (t-SNE). For this analysis, 2500 representative compounds were randomly selected from each group to assess whether the predicted natural products employed in this study adequately capture the chemical characteristics of known natural products.

2.3. Single-Task and Multi-Task Regression Models

This study applied STL and three MTL approaches: FBMTL, IBMTL, and a baseline MTL (BMTL). STL and BMTL were used as reference models to evaluate the performance of FBMTL and IBMTL. As shown in Figure 2.4.a, STL operates on a tabular data frame in which each row corresponds to a compound represented by molecular descriptors, specifically molecular fingerprints (MFPs) in this study, and labeled with a pIC_{50} value for a given protein. In this setting, an independent model is trained for each protein within a protein group, using

only the molecular fingerprints of individual compounds as input features to predict pIC_{50} values.

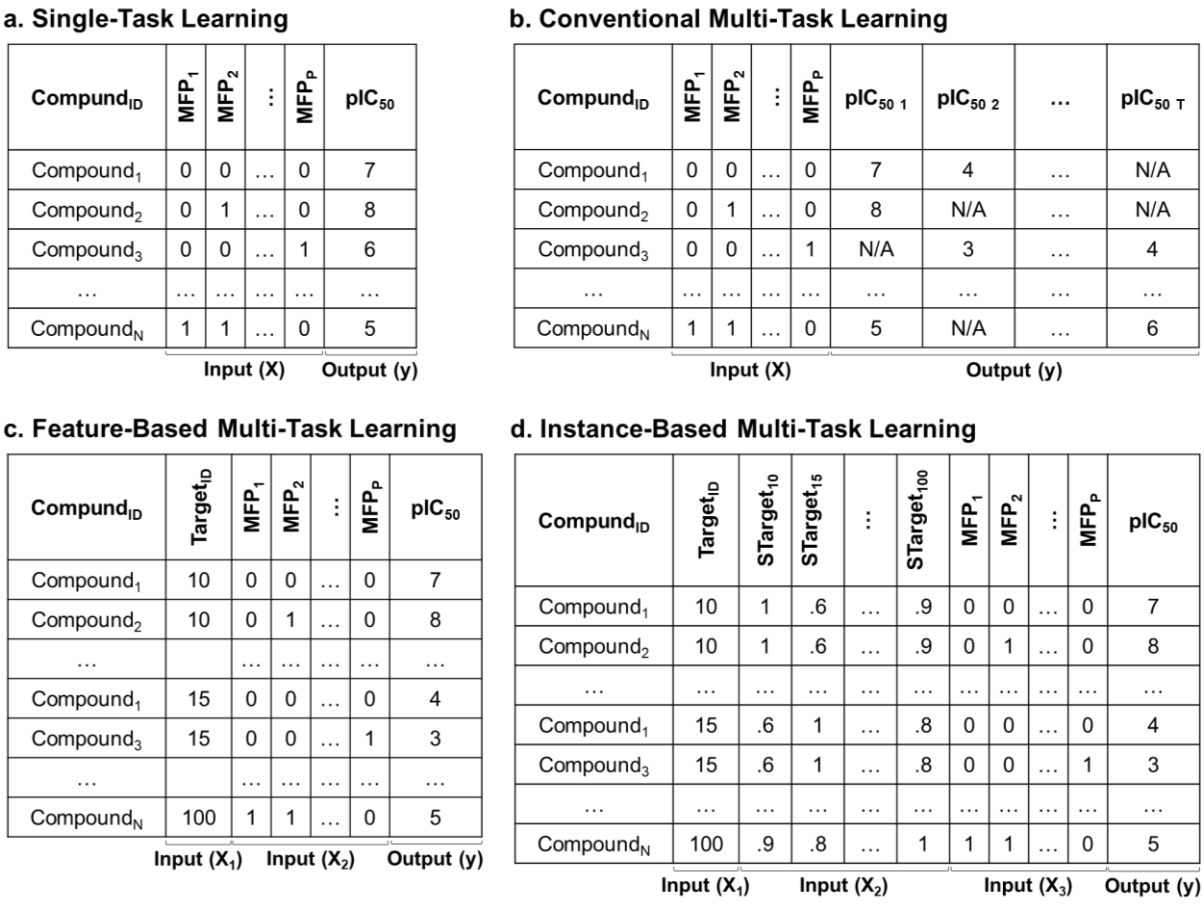


Figure 2.4. Overview of data frame formulations used in single-task learning (a), conventional multi-task learning (b), feature-based multi-task learning (c), and instance-based multi-task learning (d).

Figure 2.4.b illustrates the conventional MTL data frame, in which a dataset of N compounds is linked to a pIC_{50} matrix comprising T columns, each corresponding to a distinct target protein. Since bioactivity annotations are not available for every compound-target protein pair, this matrix can contain missing entries (Valsecchi et al., 2023). In this setting, a single model is trained using the molecular fingerprints of each compound as input and

simultaneously outputs pIC_{50} values for all target proteins. BMTL was implemented under this MTL setting.

As illustrated in Figure 2.4.c, FBMTL incorporates a task identifier, $Target_{ID}$, into the data frame. The $Target_{ID}$ is represented through one-hot encoding and included as an input feature in combination with the molecular fingerprints. Data from all proteins belonging to the same protein group are merged into a single data frame. This setting enables the use of a shared pIC_{50} response vector across all proteins in the group without introducing missing values. Each compound is duplicated k times, where k denotes the number of proteins for which annotation data are available for that compound. This unified dataset serves as the basis for training a single model, in which molecular fingerprints, together with a $Target_{ID}$, are provided as inputs to predict pIC_{50} values for compound-target pairs. FBMTL therefore implements an MTL paradigm that incorporates a task identifier into the feature set, enabling information sharing across tasks (Sadawi et al., 2019; Valsecchi et al., 2023).

IBMTL extends the FBMTL data frame by incorporating quantitative measures of protein similarity as instance weights, thereby treating these weights as additional features in the model. Figure 2.4.d shows that the $STarget_{ID}$ columns represent pairwise similarity levels among proteins within the concatenated dataset. These values are defined as the proportion of amino acid matches with similar properties relative to the aligned length, reported as a percentage. In this study, two sequence-based similarity measures were used: amino acid global sequence similarity (AA-GSS) and amino acid local sequence similarity (AA-LSS), evaluated using the Needleman-Wunsch (Needleman & Wunsch, 1970) and Smith-Waterman (Smith & Waterman, 1981) algorithms, respectively. As an illustration, in the row describing the activity of $Compound_1$ against $Target_{15}$, $STarget_{10}$ captures the similarity between $Target_{10}$ and $Target_{15}$, while $STarget_{15}$ takes a value of 1 because the target is compared to itself. Thus, IBMTL is an

MTL framework that extends FBMTL by utilizing data instances from all tasks to build a model via instance weighting (Sadawi et al., 2019; Valsecchi et al., 2023).

Before performing the MTL analyses, a preliminary evaluation was conducted to determine the most suitable machine learning algorithm for regression. The kinase group and its subgroups were used as representative examples to assess three machine learning algorithms: ANN, SVM, and RF. Following algorithm selection, MTL experiments were performed using all datasets.

2.4. Model Construction, Hyperparameter Optimization, and Evaluation

Model development for STL and MTL was implemented in Python 3.10.13. TensorFlow 2.10.0 was employed to construct models based on ANN, whereas Scikit-learn 1.0.2 was employed for SVM and RF. Model performance was assessed using the root mean squared error (RMSE), defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.1}$$

where N represents the number of compound-target protein pairs in the test set, y_i denotes the actual pIC_{50} value of the i-th compound-target protein pair, and $\hat{y}_i$ denotes the corresponding pIC_{50} value predicted by the model. Lower RMSE values indicate better predictive performance, and this evaluation metric was applied uniformly to both STL and MTL models to ensure a fair comparison.

In the STL setting, an independent model was constructed for each protein within a protein group. Model evaluation was conducted using nested cross-validation, with a five-fold outer loop for performance assessment and a five-fold inner loop for hyperparameter

optimization. The trained model predicted pIC_{50} values of compounds in the outer test folds, and the RMSE was calculated separately for each protein.

For FBMTL, nested stratified cross-validation with 5-fold outer and inner loops was employed to predict pIC_{50} values across all proteins using a single machine learning model. Stratification by $Target_{ID}$ ensured proportional representation of compounds associated with each protein in the training and test sets of every fold. After generating predictions for all compounds in the outer test folds, the results were reassigned to individual proteins according to $Target_{ID}$, enabling RMSE calculation for each protein. IBMTL followed the same model construction and evaluation procedure as FBMTL, with the addition of protein similarity features. Amino acid global sequence similarity (AA-GSS) and amino acid local sequence similarity (AA-LSS) were incorporated as additional features and were computed using Biopython 1.83 (Cock et al., 2009).

BMTL, as a widely adopted MTL setting, was also applied in this study as a comparative MTL approach. BMTL followed the conventional MTL data frame setting and was implemented as a deep learning model using a graph convolutional network. The BMTL models were developed with kMoL 1.1.9.3 (Cozac et al., 2025).

Hyperparameter optimization for each model was carried out using Bayesian optimization with a tree-structured Parzen estimator (TPE) (Bergstra et al., 2011), implemented in Optuna (Akiba et al., 2019). Model performance was assessed using a nested cross-validation scheme to obtain robust and unbiased estimates. For each outer training set, an inner five-fold cross-validation was conducted for hyperparameter tuning. During this inner loop, models were trained on the inner training folds and evaluated on the inner validation folds to identify the optimal hyperparameter configuration. The hyperparameter set yielding the lowest mean squared error (MSE) was selected as the optimal hyperparameter and used to train a

model on the full outer training set, which was then evaluated on the outer test set. MSE is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.2}$$

This nested cross-validation scheme is illustrated in Figure 2.5. By employing this technique, the test data are strictly excluded from model training and hyperparameter optimization, ensuring an unbiased evaluation of performance. The hyperparameters used for STL, FBMTL, and IBMTL are summarized in Table 2.2, while those used for BMTL are summarized in Table 2.3.

Table 2.2. Hyperparameter settings applied in nested cross-validation for STL, FBMTL, and IBMTL

Learning Algorithm	Hyperparameter	Number of Trial
Artificial Neural Networks	Number of Units per Layer = 16-128 Number of Layers = 1-3 Learning Rate = 0.00001-0.01 Number of Epochs = 50-200 Batch Size = 16-128	20
Support Vector Machine	Regularization Parameter (C) = 0.01-100 Epsilon (ε) = 0.001-1 Gamma (γ) = 0.00001-10	20
Random Forest	Number of Tress = 50-200 Maximum Depth = 3-25	20

Table 2.3. Hyperparameter settings applied in nested cross-validation for BMTL

Algorithm	Hyperparameters	Number of Trials
Graph Convolutional	Hidden Features = 32-256 Dropout = 0.0-0.7 Layer Type = torch_geometric.nn.GCNConv, torch_geometric.nn.GraphConv, torch_geometric.nn.GENConv, torch_geometric.nn.LEConv, torch_geometric.nn.SAGEConv, kmol.model.layers.GINConvolution Layers Count = 2-8 Batch Size = 64-256	20

2.5. Statistical Analysis

The predictive performance of all models, including STL, FBMTL, IBMTL AA-GSS, IBMTL AA-LSS, and BMTL, was assessed using the average RMSE, defined as the mean RMSE across all proteins within each protein group. The performance of each MTL model was then compared with that of the STL model. Statistical significance was evaluated using a one-tailed paired t-test (right-tailed), based on the alternative hypothesis that the average RMSE obtained with the STL model is higher than that of the corresponding MTL model. Statistical significance thresholds were set at $p < 0.05$, $p < 0.01$, and $p < 0.001$. All statistical analyses were conducted using SciPy 1.13.0 (Virtanen et al., 2020).

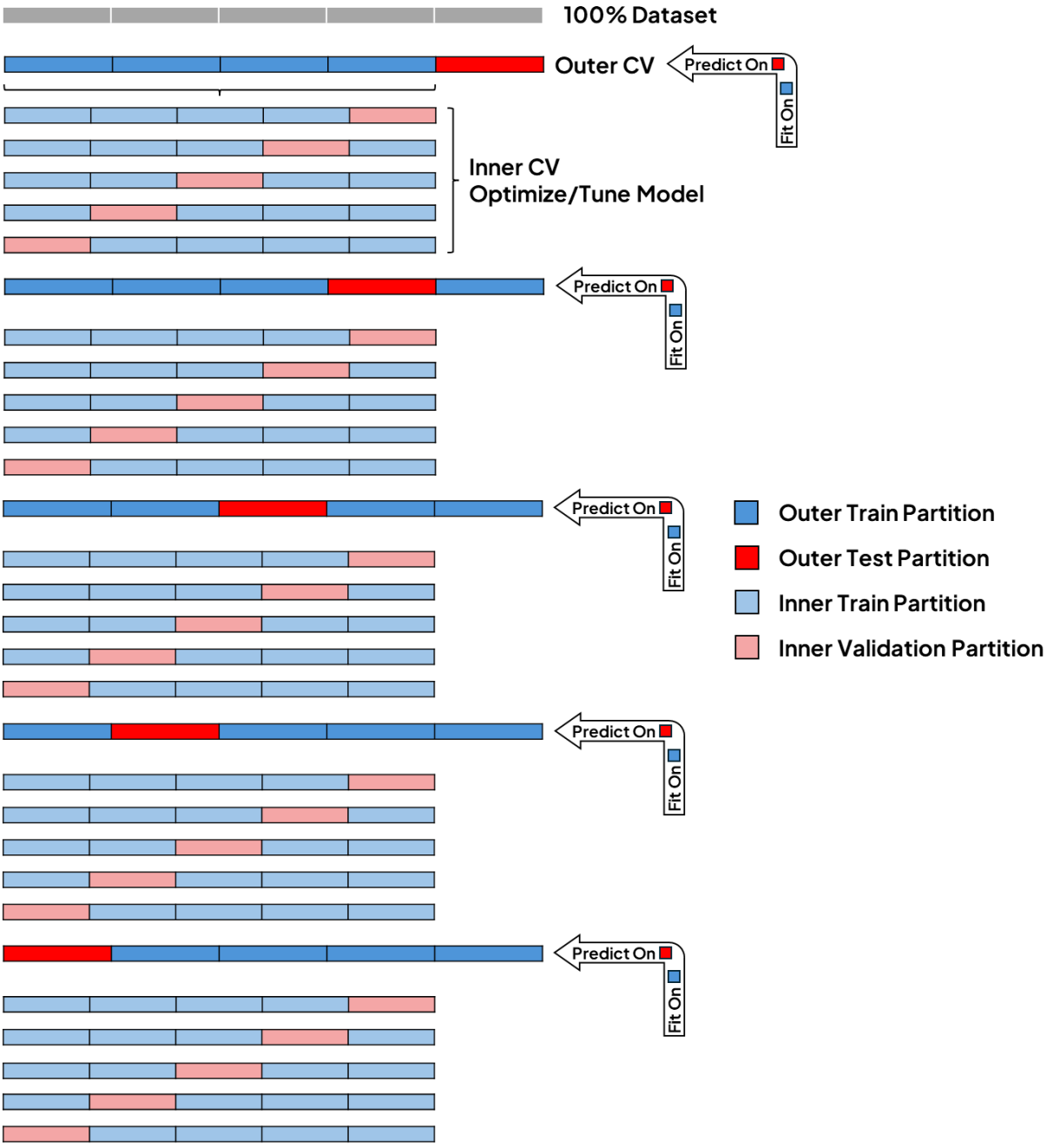


Figure 2.5. Nested cross-validation scheme for hyperparameter optimization and performance evaluation (Lavasa et al., 2021).

2.6. Studies of MTL Models over Three Hierarchical Protein Group Levels

After conducting MTL analyses using all compound pIC_{50} data for proteins classified at level 2 and lower within the enzyme group, the investigation was further extended to more specific hierarchical levels. The previously analyzed levels are collectively referred to as the

superclass level. At the next level, termed the target parent level, the analysis was restricted to proteins within the enzyme group classified at level 2 and lower that share the same Target Parent ID. Subsequently, at the protein class level, the analysis was conducted on proteins in the enzyme group classified as level 2 and lower, restricted to those sharing identical Target Parent ID and Protein Class ID.

According to the ChEMBL 35 database, the Protein Class ID uniquely defines each protein family, whereas the Target Parent ID uniquely identifies the corresponding parent family. To maintain the feasibility of MTL analyses at the lowest level, an additional constraint was used, requiring that each Protein Class ID comprises at least two proteins.

2.7. Identification of Potential AKT2 Inhibitors through Virtual Screening of Natural Product Databases

Virtual screening to identify potential AKT2 inhibitors was carried out on several natural product databases: NPASS, NP Atlas, CMNPD, InterBioScreen, and MolPort, consisting of 243547 natural product structures in total. All structures were first converted to their uncharged forms with RDKit's non-forced uncharger to match the uncharged-state treatment used in developing the machine learning model. After removing salt counterions and small fragments, SMILES normalization was applied, and duplicate SMILES in the merged dataset were removed, ensuring that each compound was represented only once.

All compounds were transformed into molecular fingerprints consistent with those used in previous single and multi-task regression studies. In addition, Target ID and Similarity features derived from AKT2 protein data were incorporated. The best-performing MTL model, along with its optimized hyperparameters, was subsequently used to estimate pIC_{50} values for the natural product dataset. The predicted values were ranked to identify the top ten candidates, and their pIC_{50} values were converted to predicted IC_{50} values. These predictions were

evaluated by comparison with experimental IC_{50} values for reference compounds, including representative predicted natural products in ChEMBL 35 covering a range of AKT2 activities from low to high and the established synthetic AKT2 inhibitor, GSK690693.

2.8. Comparative Evaluation of Virtual Screening Results with Protein-Ligand Docking

The results of virtual screening were compared with protein-ligand docking results, followed by a comparison of the docking results with those of reference compounds. For molecules available only as 2D structures or SMILES, 3D structures were generated using Avogadro 1.2.0 (Hanwell et al., 2012). Semi-empirical PM6 calculations (Stewart, 2007) were used to generate energy-minimized structures for the ten selected natural products and the negative control, implemented in Molecular Orbital Package (MOPAC) (Stewart, 2016).

The human AKT2 crystal structure in complex with GSK690693 (PDB code: 3D0E) (Heerding et al., 2008) was retrieved from the Protein Data Bank (Berman et al., 2000) and prepared with AutoDockTools 1.5.7 (Morris et al., 2009). Preparation steps included removal of water molecules, extraction of the co-crystallized ligand (GSK690693), addition of polar hydrogens, merging of nonpolar hydrogens, and assignment of Gasteiger charges to the protein.

Two software programs, including AutoDock Vina 1.1.2 (Trott & Olson, 2010) and Molecular Operating Environment (MOE) 2024.0601 were used to perform protein-ligand docking. Before the primary docking analysis, protein-ligand docking validation was performed by redocking the co-crystallized ligand (GSK690693) into the human AKT2 protein. This validation aimed to assess whether each docking algorithm could accurately reproduce the experimentally observed binding pose and rank it as a top-scoring conformation.

Docking simulations using AutoDock Vina were executed through PyRx 1.1 (Dallakyan & Olson, 2015), in which the docking grid box was centered on the co-crystallized ligand at coordinates x = 22.521, y = -19.611, and z = 7.41, with a cubic grid box of 26.25 Å in each

dimension. Docking was performed with an exhaustiveness of 64, and a total of 27 docking poses were generated.

In MOE, the binding site was determined with the Site Finder tool and selected according to the highest propensity for ligand binding (PLB) score. Ligand placement was performed using the Triangle Matcher, followed by refinement with Rigid Receptor. During the placement step, 60 docking poses were generated using the London dG scoring function, followed by refinement with the generalized born volume integral/weighted surface area (GBVI/WSA) dG scoring function, yielding 10 final poses.

2.9. References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2623–2631. https://doi.org/10.1145/3292500.3330701

Beisken, S., Meinl, T., Wiswedel, B., Figueiredo, L. F. De, & Berthold, M. (2013). KNIME-CDK : Workflow-driven cheminformatics. *BMC Informatics*, *14*(257), 2–5. https://doi.org/10.1186/1471-2105-14-257

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2546–2554. https://dl.acm.org/doi/10.5555/2986459.2986743

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., & Bourne, P. E. (2000). The protein data bank. *Nucleic Acids Research*, *28*(1), 235–242. https://doi.org/10.1093/nar/28.1.235

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., & Wiswedel, B. (2008). *KNIME: The Konstanz Information Miner BT - Data Analysis, Machine Learning and Applications* (C. Preisach, H. Burkhardt, L. Schmidt-Thieme, & R. Decker, Eds.; pp. 319–326). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-78246-9_38

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., & de Hoon, M. J. L. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, *25*(11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163

Cozac, R., Hasic, H., Choong, J. J., Richard, V., Beheshti, L., Froehlich, C., Koyama, T., Matsumoto, S., Kojima, R., Iwata, H., Hasegawa, A., Otsuka, T., & Okuno, Y. (2025). kMoL: an open-source machine and federated learning library for drug discovery. *Journal of Cheminformatics*, *17*(1), 22. https://doi.org/10.1186/s13321-025-00967-9

Dallakyan, S., & Olson, A. J. (2015). Small-Molecule Library Screening by Docking with PyRx. In J. E. Hempel, C. H. Williams, & C. C. Hong (Eds.), *Chemical Biology: Methods and Protocols* (pp. 243–250). Springer New York. https://doi.org/10.1007/978-1-4939-2269-7_19

Hanwell, M. D., Curtis, D. E., Lonie, D. C., Vandermeersch, T., Zurek, E., & Hutchison, G. R. (2012). Avogadro: an advanced semantic chemical editor, visualization, and analysis platform. *Journal of Cheminformatics*, *4*(1), 17. https://doi.org/10.1186/1758-2946-4-17

Heerding, D. A., Rhodes, N., Leber, J. D., Clark, T. J., Keenan, R. M., Lafrance, L. V, Li, M., Safonov, I. G., Takata, D. T., Venslavsky, J. W., Yamashita, D. S., Choudhry, A. E., Copeland, R. A., Lai, Z., Schaber, M. D., Tummino, P. J., Strum, S. L., Wood, E. R., Duckett, D. R., … Kumar, R. (2008). Identification of 4-(2-(4-Amino-1,2,5-oxadiazol-3-yl)-1-ethyl-7-{[(3S)-3-piperidinylmethyl]oxy}-1H-imidazo[4,5-c]pyridin-4-yl)-2-methyl-3-butyn-2-ol (GSK690693), a Novel Inhibitor of AKT Kinase. *Journal of Medicinal Chemistry*, *51*(18), 5663–5679. https://doi.org/10.1021/jm8004527

Hermansyah, O., Bustamam, A., & Yanuar, A. (2021). Virtual screening of dipeptidyl peptidase-4 inhibitors using quantitative structure–activity relationship-based artificial intelligence and molecular docking of hit compounds. *Computational Biology and Chemistry*, *95*. https://doi.org/10.1016/j.compbiolchem.2021.107597

InterBioScreen Ltd. (2025). *Natural Compounds*. https://www.ibscreen.com/natural-compounds

Kausar, S., & Falcao, A. O. (2018). An automated framework for QSAR model building. *Journal of Cheminformatics*, *10*(1), 1. https://doi.org/10.1186/s13321-017-0256-5

Lavasa, E., Giannopoulos, G., Papaioannou, A., Anastasiadis, A., Daglis, I. A., Aran, A., Pacheco, D., & Sanahuja, B. (2021). Assessing the Predictability of Solar Energetic Particles with the Use of Machine Learning Techniques. *Solar Physics*, *296*(7), 107. https://doi.org/10.1007/s11207-021-01837-x

Lyu, C., Chen, T., Qiang, B., Liu, N., Wang, H., Zhang, L., & Liu, Z. (2021). CMNPD: a comprehensive marine natural products database towards facilitating drug discovery from the ocean. *Nucleic Acids Research*, *49*(D1), D509–D515. https://doi.org/10.1093/nar/gkaa763

Molport. (2025). *Natural Product and NP-Like Compound Library*. https://www.molport.com/shop/natural-compound-database

Morris, G. M., Huey, R., Lindstrom, W., Sanner, M. F., Belew, R. K., Goodsell, D. S., & Olson, A. J. (2009). AutoDock4 and AutoDockTools4: Automated docking with selective

receptor flexibility. *Journal of Computational Chemistry*, *30*(16), 2785–2791. https://doi.org/10.1002/jcc.21256

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. https://doi.org/10.1016/0022-2836(70)90057-4

O'Boyle, N. M., Banck, M., James, C. A., Morley, C., Vandermeersch, T., & Hutchison, G. R. (2011). Open Babel: An open chemical toolbox. *Journal of Cheminformatics*, *3*(1), 33. https://doi.org/10.1186/1758-2946-3-33

Poynton, E. F., van Santen, J. A., Pin, M., Contreras, M. M., McMann, E., Parra, J., Showalter, B., Zaroubi, L., Duncan, K. R., & Linington, R. G. (2025). The Natural Products Atlas 3.0: extending the database of microbially derived natural products. *Nucleic Acids Research*, *53*(D1), D691–D699. https://doi.org/10.1093/nar/gkae1093

RDKit. (2025). *RDKit Mol Standardize Uncharger Class Reference*. https://www.rdkit.org/docs/cppapi/classRDKit_1_1MolStandardize_1_1Uncharger.html

RDKit. (2024). *RDKit: Open-source cheminformatics.* (2024.03.1). https://www.rdkit.org

Sadawi, N., Olier, I., Vanschoren, J., Van Rijn, J. N., Besnard, J., Bickerton, R., Grosan, C., Soldatova, L., & King, R. D. (2019). Multi-task learning with a natural metric for quantitative structure activity relationship learning. *Journal of Cheminformatics*, *11*(1). https://doi.org/10.1186/s13321-019-0392-1

Seo, M., Shin, H. K., Myung, Y., Hwang, S., & No, K. T. (2020). Development of Natural Compound Molecular Fingerprint (NC-MFP) with the Dictionary of Natural Products (DNP) for natural product-based drug development. *Journal of Cheminformatics*, *12*(1). https://doi.org/10.1186/s13321-020-0410-3

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*(1), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

Stewart, J. J. P. (2007). Optimization of parameters for semiempirical methods V: Modification of NDDO approximations and application to 70 elements. *Journal of Molecular Modeling*, *13*(12), 1173–1213. https://doi.org/10.1007/s00894-007-0233-4

Stewart, J. J. P. (2016). *MOPAC2016; Stewart Computational Chemistry: Colorado Springs, Colorado*. http://openmopac.net/

Trott, O., & Olson, A. J. (2010). AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, *31*(2), 455–461. https://doi.org/10.1002/jcc.21334

Valsecchi, C., Grisoni, F., Consonni, V., Ballabio, D., & Todeschini, R. (2023). Multitask Learning for Quantitative Structure–Activity Relationships: A Tutorial. In H. Huixiao (Ed.), *Machine Learning and Deep Learning in Computational Toxicology*. https://doi.org/10.1007/978-3-031-20730-3_8

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Contributors, S. 1. 0. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, *28*(1), 31–36. https://doi.org/10.1021/ci00057a005

Zhao, H., Yang, Y., Wang, S., Yang, X., Zhou, K., Xu, C., Zhang, X., Fan, J., Hou, D., Li, X., Lin, H., Tan, Y., Wang, S., Chu, X.-Y., Zhuoma, D., Zhang, F., Ju, D., Zeng, X., & Chen, Y. Z. (2023). NPASS database update 2023: quantitative natural product activity and species source database for biomedical research. *Nucleic Acids Research*, *51*(D1), D621–D628. https://doi.org/10.1093/nar/gkac1069

Zhao, Z., Qin, J., Gou, Z., Zhang, Y., & Yang, Y. (2020). Multi-task learning models for predicting active compounds. *Journal of Biomedical Informatics*, *108*, 103484. https://doi.org/10.1016/j.jbi.2020.103484

Chapter 3

Results and Discussion

3.1. Machine Learning-Based Identification of Natural Products and Dataset Characterization

Given the scarcity of public databases providing biological activity data for natural products against target proteins, such as NPASS, the ChEMBL 35 database was used as the primary data source. To identify compounds likely to be natural products in the ChEMBL 35 database, a machine learning filtering model was constructed. This model was developed via binary classification studies on a dataset comprising natural products and synthetic compounds obtained from the InterBioScreen database.

Figure 3.1.a summarizes the classification accuracy of models built from pairs of 21 molecular fingerprints and four machine learning algorithms for discriminating natural products from synthetic compounds. The majority of models achieved strong classification performance, with accuracy greater than 90%, indicating robust discriminative capability. The highest accuracy was achieved with the combination of the Avalon fingerprint and the RF algorithm (95.75%).

In order to assess the robustness of the best-performing classification model, external validation was conducted using independent test sets of natural products from the Molport, NPASS, NP Atlas, and CMNPD databases. As shown in Figure 3.1.b, the external validation results indicate that the model used to generate natural product labels maintains high sensitivity (exceeding 80%), demonstrating strong generalizability to independent datasets. Among the evaluated databases, NP Atlas and CMNPD exhibited the lowest sensitivity values. This reduced performance may be attributed to the specialized nature of these databases, which

primarily contain marine- and microbially-derived natural products, respectively, thereby limiting the model's generalizability.

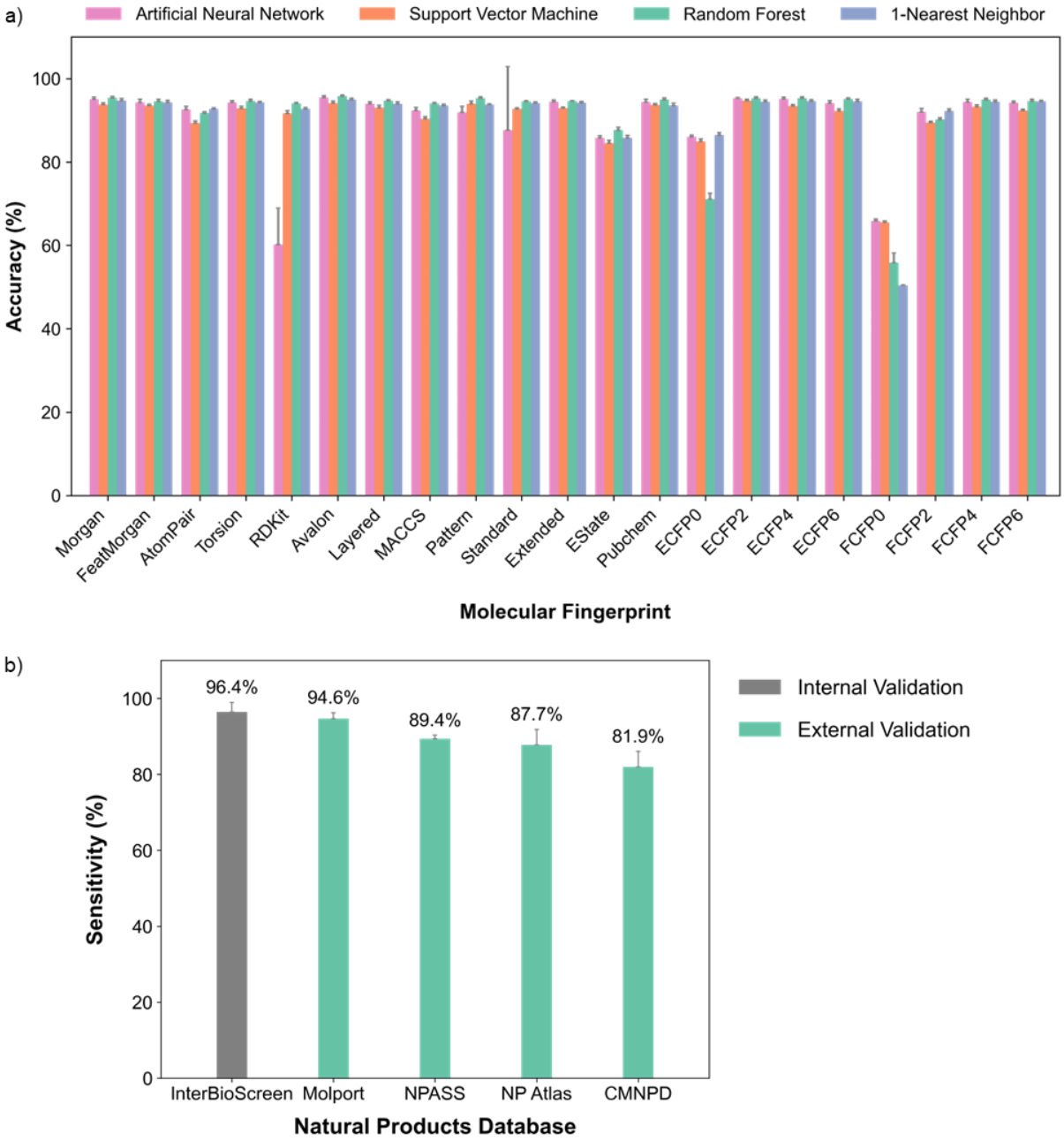


Figure 3.1. Evaluation of machine learning models for compound classification: (a) accuracy of models combining various molecular fingerprints and machine learning algorithms, (b) sensitivity of the best-performing classification model in identifying natural products across internal and external validation datasets.

Using the optimized model, the ChEMBL 35 database was filtered to extract predicted natural products and their associated bioactivity data for proteins belonging to the enzyme family (level 1) and classified at level 2 and lower (levels 3-6) under ChEMBL's six-level hierarchical protein classification (Figure 2.3). After filtering and data preprocessing, a final dataset comprising 83641 unique predicted natural products and 123248 bioactivity data was obtained.

After identifying the predicted natural products, their chemical space was projected using t-SNE together with the chemical spaces of the natural products and synthetic compounds utilized to construct the binary classification model. For this comparison, representative compounds from each of the three groups were selected, as presented in Figure 3.2.

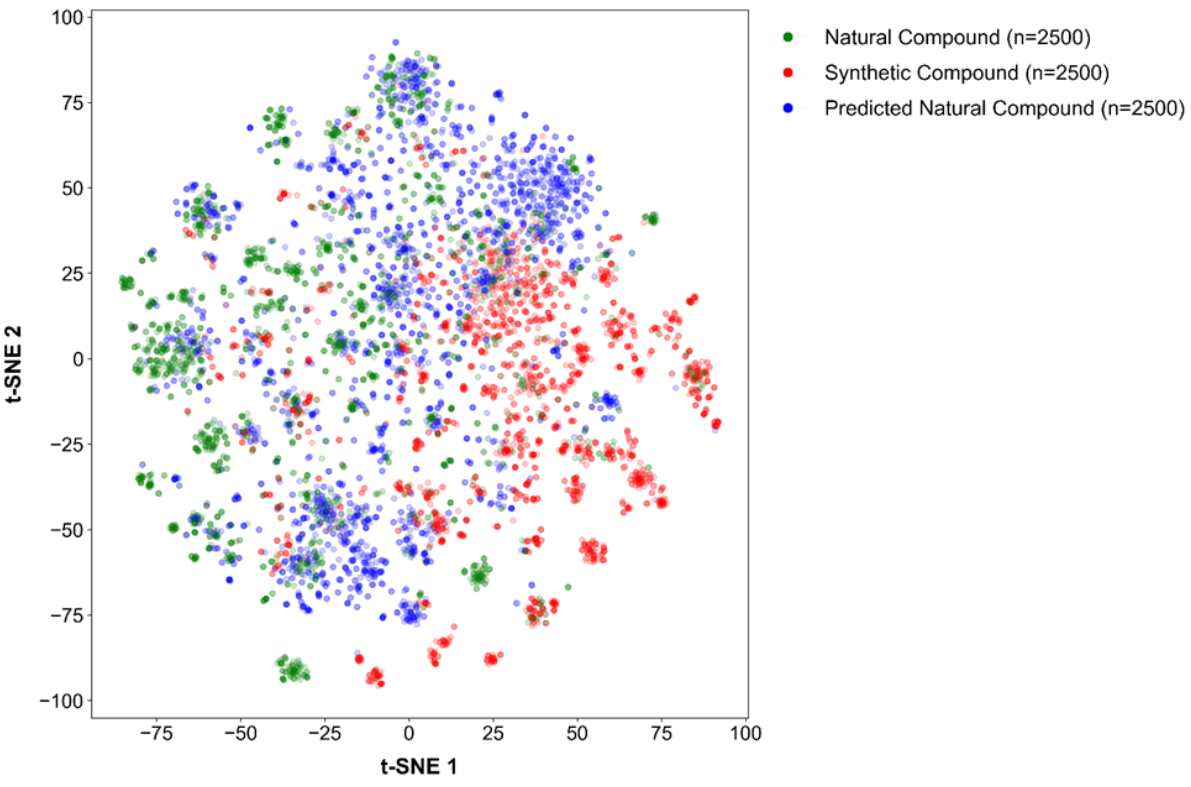


Figure 3.2. Comparison of the chemical space of natural compounds, synthetic compounds, and predicted natural compounds based on t-SNE analysis.

The overlap observed in Figure 3.2 between natural products and synthetic compounds highlights the ambiguity of chemical space, in which structural similarities exist between the two groups. For instance, salicylic acid, although classified as a natural product, shares similar structural features with aspirin, a synthetic compound developed from it. Despite this overlap, most natural products are distributed across distinct regions, indicating structural differences relative to synthetic compounds. In addition, the chemical space of natural products is broader than that of synthetic compounds, underscoring their greater structural diversity. Most predicted natural products are located within the natural product region, although a small number overlap with the region occupied by synthetic compounds. Overall, these observations support that the predicted natural products employed in this study are reasonably representative of natural products.

After data preprocessing, the distribution of protein counts is substantially reduced, as shown in Figure 3.3. Similarly, most proteins are associated with a small number of compounds in the 50-250 range, followed by the 251-500 range, as illustrated in Figure 3.4. These patterns indicate that the study is based on relatively small datasets, making MTL an appropriate approach, as it can improve learning performance by exploiting shared information among related tasks (Simões et al., 2018). The distribution of compound bioactivity data per protein group after data preprocessing is shown in Figure 3.5.

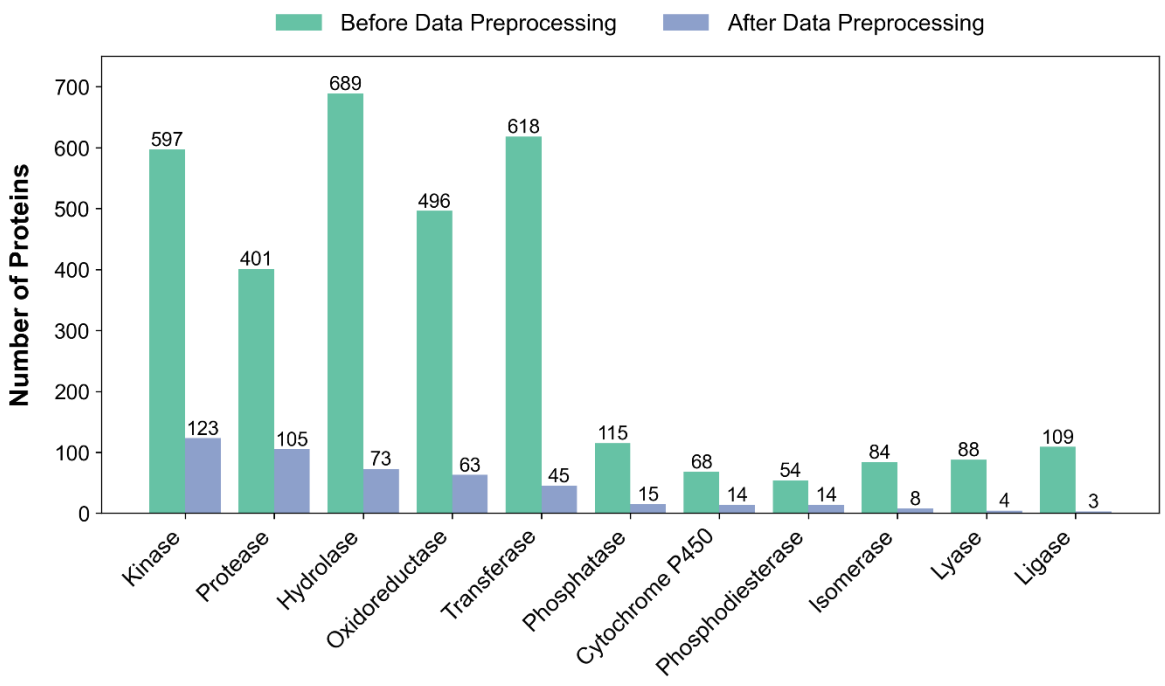


Figure 3.3. Protein distribution across all protein groups within the enzyme family, before and after data preprocessing.

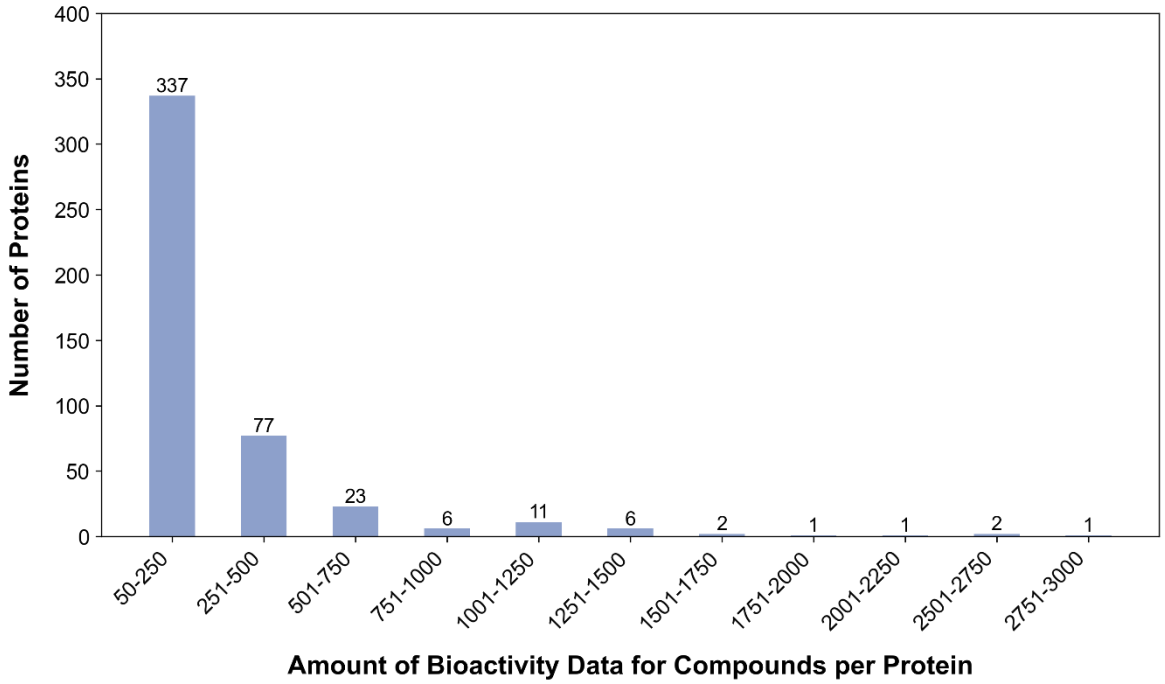


Figure 3.4. Protein-to-bioactivity data distribution per protein across all protein groups within the enzyme family after data preprocessing.

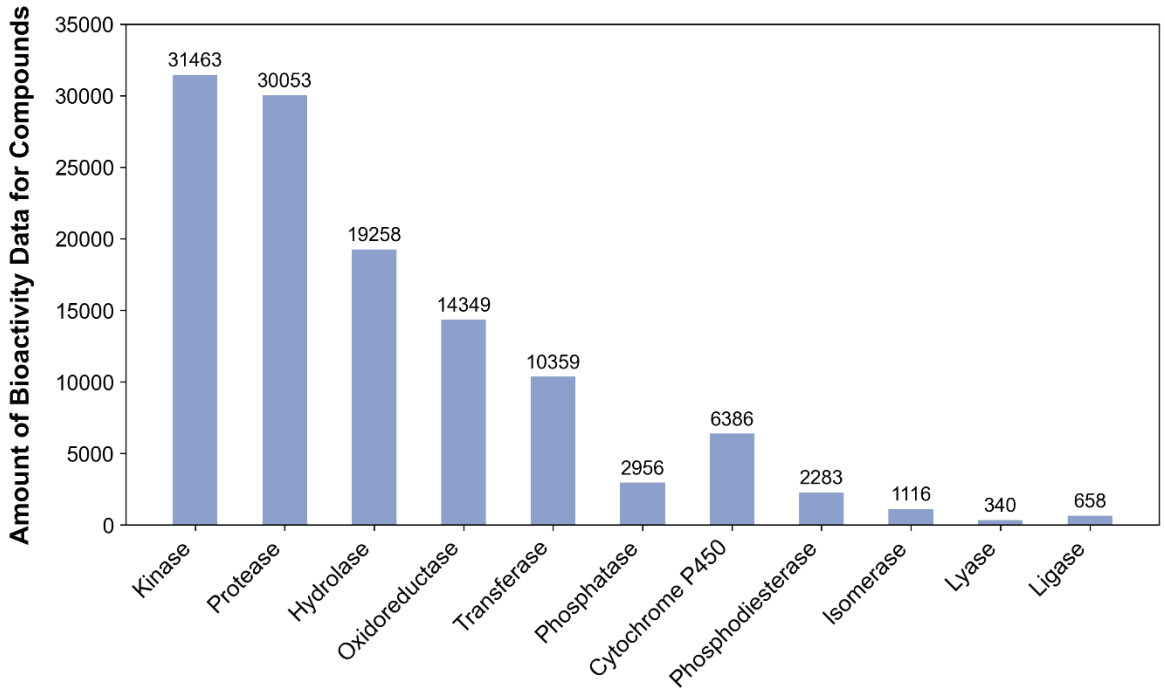


Figure 3.5. Bioactivity data distribution per protein group within the enzyme family after data preprocessing.

3.2. Bioactivity Predictive Performance Comparison of STL and MTL across Protein Groups in the Enzyme Family

The MTL studies considered three approaches: FBMTL, IBMTL, and BMTL. FBMTL and IBMTL were used to avoid missing values in the data frame, making it possible to apply a wide range of conventional machine learning algorithms rather than being restricted to deep learning algorithms. Deep learning models generally need large datasets to capture meaningful patterns effectively. Under data-scarce conditions, they generally exhibit poor generalization, while conventional machine learning models are often more effective on small datasets, leading to better predictive reliability (Dou et al., 2023). BMTL was adopted for comparison as a widely used MTL approach that employs a conventional MTL data frame and applies a deep learning algorithm, which in this study is a graph convolutional network.

Based on a comparative evaluation of three regression algorithms: ANN, SVM, and RF, across kinase group proteins, RF showed the most favorable performance. Combined with the Avalon fingerprint, previously identified as the most effective molecular representation, RF was therefore selected for further MTL studies (excluding BMTL) on all protein groups, given that it consistently produced the lowest average RMSE across the FBMTL and both IBMTL (AA-GSS and AA-LSS) models within the kinase group, as illustrated in Figure 3.6.

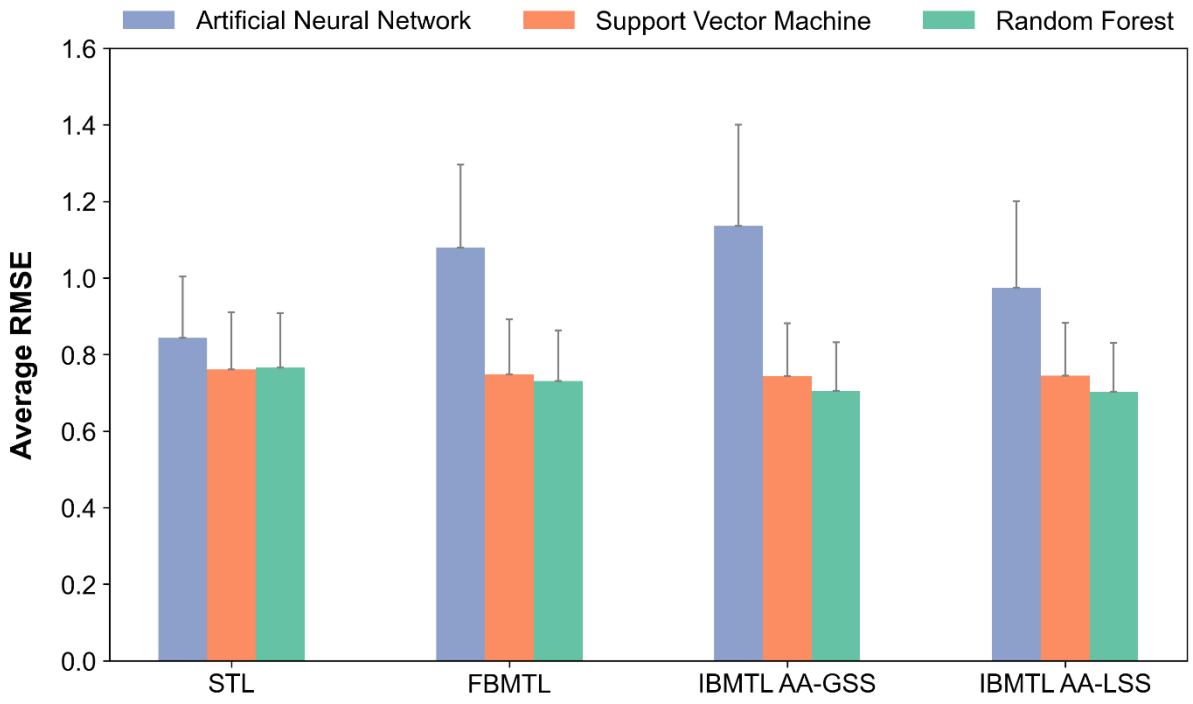


Figure 3.6. Evaluation of three regression learning algorithms (ANN, SVM, and RF) under STL and MTL settings for proteins in the kinase group.

The predictive performance comparison of the five approaches in RMSE across different protein groups is shown in Figure 3.7. While FBMTL outperforms STL in some groups, this improvement reaches statistical significance only for the kinase group ($p < 0.001$). In contrast, in other groups, including phosphatase, lyase, and ligase, the observed improvements are not statistically significant. Despite these results, FBMTL consistently underperforms IBMTL

across all protein groups, indicating that integrating protein similarity information, in this study, amino acid global and local sequence similarities, improves model performance.

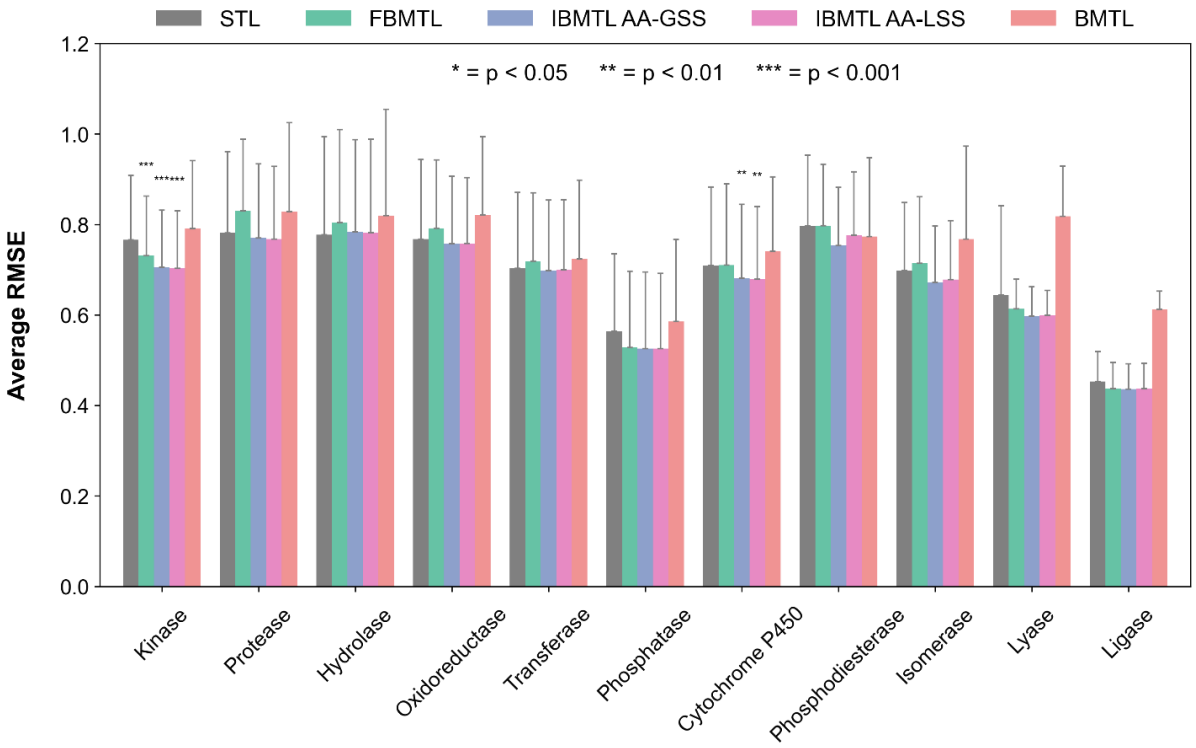


Figure 3.7. Evaluation of STL, FBMTL, IBMTL AA-GSS, IBMTL AA-LSS, and BMTL predictive performance across all protein groups under the enzyme family.

Compared with STL, IBMTL AA-GSS and IBMTL AA-LSS achieve lower average RMSE in almost all protein groups, except in the hydrolase group. Both IBMTL methods show statistically significant improvements over STL only in the kinase ($p < 0.001$) and cytochrome P450 ($p < 0.01$) groups. These two groups comprise proteins classified at higher-numbered, more specific levels of ChEMBL's six-level hierarchical protein classification, as shown in Tables S1 and S7 in the Supplementary Materials.

By contrast, both IBMTL methods exhibit poorer performance in the hydrolase, oxidoreductase, and transferase groups, where proteins are classified to lower-numbered, less

specific levels (level 2), as shown in Tables S3-S5. As illustrated in Figure 2.3, ChEMBL's six-level hierarchical protein classification assigns proteins with greater similarity to higher-numbered levels, while lower-numbered levels typically represent broader and more diverse proteins. These findings demonstrate that protein evolutionary relatedness is strongly associated with the IBMTL performance.

However, this trend is not observed in the protease group. Although most protease proteins are assigned to level 5 (Table S2) and the dataset size is similar to the kinase group (Figure 3.5), the improvement of IBMTL over STL does not reach statistical significance. This unexpected result is further examined in the section discussing MTL performance across the three hierarchical protein group levels.

BMTL performs inferior to the other models across most protein groups, with satisfactory results observed only in the phosphodiesterase group. This performance disparity appears to be more pronounced in the isomerase, lyase, and ligase groups, potentially due to their relatively small dataset sizes (Figure 3.5). The limited dataset poses challenges for the graph convolutional network, a deep learning approach, thereby limiting its ability to generalize effectively. The results demonstrate that the proposed models consistently perform better than BMTL, a commonly used MTL approach with a more complex deep learning framework, particularly for protein groups with limited data.

3.3. Effect of Three Hierarchical Protein Group Levels on MTL Performance in Kinase and Protease Groups

Given that IBMTL showed a significant performance improvement over STL in the kinase group but only a marginal improvement in the protease group, the effect of three hierarchical protein grouping on MTL performance was further investigated in both groups. As described in the Methods Section, three levels of protein hierarchical grouping were

considered: superclass, target parent, and protein class. Each Protein Class ID was required to include at least two proteins to ensure fair comparison across levels. Consequently, the analysis was limited to 85 target proteins in the kinase group and 90 in the protease group, compared with 123 and 105 targets, respectively, in the previous section.

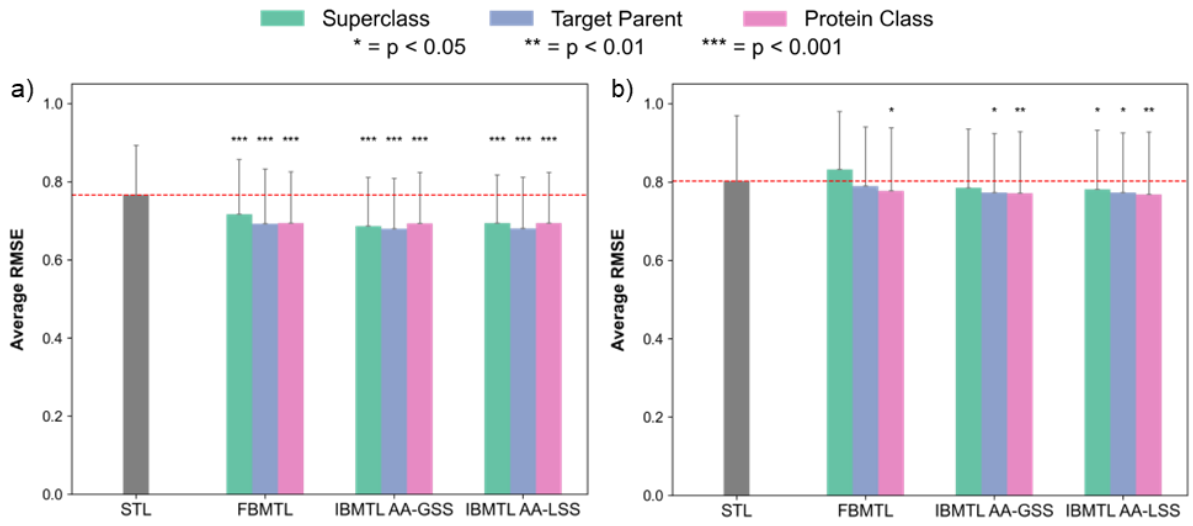


Figure 3.8. Evaluation of STL, FBMTL, IBMTL AA-GSS, and IBMTL AA-LSS predictive performance across superclass, target parent, and protein class levels in the kinase group (a) and protease group (b).

As shown in Figure 3.8.a, the average RMSE values of the four approaches are compared across three hierarchical protein group levels within the kinase group. FBMTL shows statistically significant improvements over STL at each level ($p < 0.001$). The best performance occurred at the target parent level, with only a marginal difference compared with the protein class level.

Across all three hierarchical levels, IBMTL models consistently achieved significantly better performance than STL ($p < 0.001$) and also outperformed FBMTL at every level. Among the levels examined, the target parent level yielded the best performance for IBMTL models, compared with the protein superclass and protein class levels. This trend indicates that, within

the kinase group, the optimal performance of IBMTL arises from a balance between the number of proteins included in each model, along with the corresponding bioactivity data of compounds, and the average similarity among those proteins. At the superclass level, all proteins with available bioactivity data for the predicted natural products were combined into a single group, resulting in a lower average protein similarity. Conversely, grouping proteins at the protein class level resulted in their distribution across multiple groups, reducing the number of proteins per model while increasing average protein similarity.

Figure 3.8.b presents a comparative analysis of the average RMSE across the four approaches at the three hierarchical protein group levels in the protease group. All MTL approaches show progressive improvement in performance from the superclass level to the target parent level, and then to the protein class level. In contrast to kinases, which possess a highly conserved catalytic domain within their family, proteases exhibit diverse catalytic domains, and certain members are not evolutionarily related to other proteases (Hanks et al., 1988; López-Otín & Bond, 2008). This heterogeneity contributes to the weaker performance of MTL models at the superclass level, where proteins are more diverse. As proteins are categorized into more specific groups at the protein class level, MTL model performance improves.

Across all hierarchical levels in the protease group, IBMTL AA-GSS and IBMTL AA-LSS outperform both STL and FBMTL. Peak performance for these IBMTL models occurs at the protein class level, where the improvement over STL is statistically significant ($p < 0.01$). The same pattern is observed in the kinase group and is consistent with the report by Moon and Kim, suggesting that applying MTL to more diverse targets may reduce predictive performance (Moon & Kim, 2022).

3.4. Effects of Protein Similarity, Protein Count, and Bioactivity Data Availability on MTL Performance at the Target Parent Level in the Kinase Group

In order to identify factors affecting MTL performance, the effects of average protein sequence similarity, protein count, and the availability of compound bioactivity data were analyzed at the target parent level in the kinase group, in which FBMTL and IBMTL achieved the lowest average RMSE. Because the two protein evolutionary relatedness metrics used in IBMTL produced comparable average RMSE results, only one metric, AA-LSS, was considered in this analysis.

Figure 3.9.a indicates that when the average local sequence similarity among proteins exceeds 30%, both FBMTL and IBMTL AA-LSS outperform STL. In contrast, STL achieves better performance than both MTL approaches only in cases of very low similarity, less than 30%. While IBMTL demonstrates a noticeable improvement over FBMTL at low similarity, this performance gap narrows as similarity increases, resulting in more comparable average RMSE. This trend suggests that when average local sequence similarity is high, incorporating evolutionary relatedness information provides limited additional benefit for IBMTL compared with FBMTL.

In addition, MTL model performance is influenced by the number of proteins belonging to the same Target Parent ID. As shown in Figure 3.9.b, both FBMTL and IBMTL AA-LSS consistently exhibit lower average RMSE than STL across all protein counts. As the number of proteins in the same Target Parent ID increases, IBMTL AA-LSS outperforms FBMTL, with the performance difference becoming more pronounced. On the other hand, a decrease in the number of proteins leads to a smaller performance difference. This trend suggests that for groups with few proteins, the protein sequence similarity attributes incorporated in IBMTL AA-LSS do not meaningfully enhance performance relative to FBMTL, and *vice versa*.

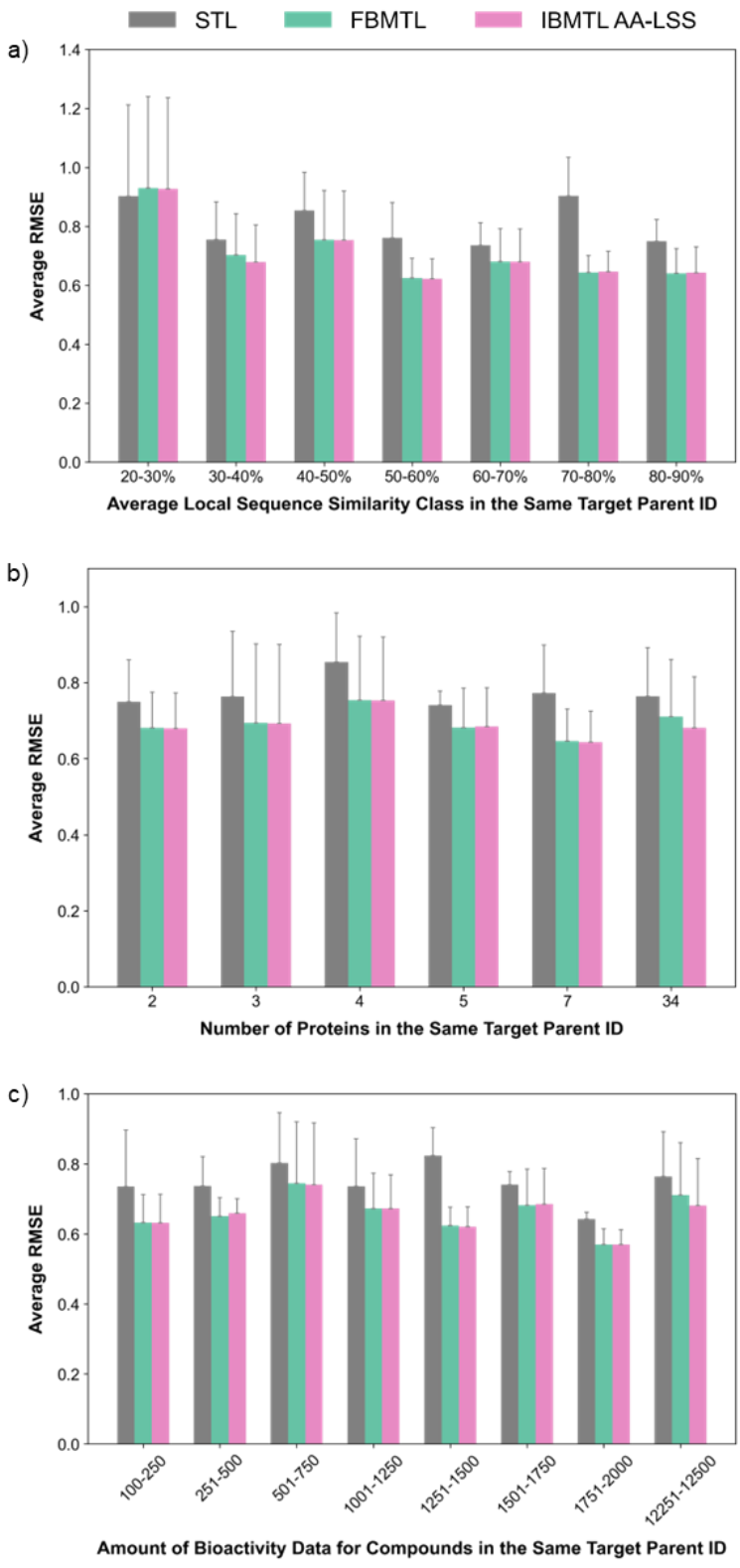


Figure 3.9. Evaluation of STL, FBMTL, and IBMTL AA-LSS predictive performance at the target parent level in the kinase group, evaluated by (a) average protein local sequence similarity, (b) protein count, and (c) compound bioactivity data availability.

The availability of compound bioactivity data targeting proteins in the same Target Parent ID also influences model performance. As shown in Figure 3.9.c, both FBMTL and IBMTL AA-LSS demonstrate better performance than STL across all classes of compound bioactivity data availability. As the amount of compound bioactivity data in the same Target Parent ID increases, IBMTL AA-LSS demonstrates superior performance over FBMTL, and the gap between the two methods widens. As the amount of compound bioactivity data declines, the performance gap narrows. However, this trend is partially driven by the number of proteins in the same Target Parent ID, as the amount of compound bioactivity data is associated with protein count.

3.5. Virtual Screening of Natural Products Against AKT2 Using the Best-Performing Predictive Model and Docking-Based Comparative Analysis

After comparing the average RMSE values of the four models: STL, FBMTL, IBMTL AA-GSS, and IBMTL AA-LSS, across three hierarchical protein group levels, their individual performance for AKT2 was further evaluated, as summarized in Table 3.1. Performance at the protein class level was not evaluated because AKT2 is associated with a Target Parent ID containing only a single Protein Class ID. As a result, identical model performance is observed at the target parent and protein class levels.

Among the evaluated models, IBMTL AA-LSS performed best at the target parent level, as indicated by the lowest average RMSE. Consequently, this model was employed for virtual screening to predict the bioactivity of compounds compiled from various natural product databases.

The final dataset for virtual screening consisted of 242118 unique natural product structures after structural deduplication. Among these, the ten compounds exhibiting the lowest predicted IC_{50} values, reflecting strong potential activity against AKT2, are listed in Table 3.2.

In addition, reference compounds are provided, including representative predicted natural products from the ChEMBL 35 database with actual IC_{50} values against AKT2 across a wide range of activities, together with an established synthetic AKT2 inhibitor with its actual IC_{50} value.

Table 3.1. Evaluation of STL, FBMTL, IBMTL AA-GSS, and IBMTL AA-LSS predictive performance for AKT2 across two hierarchical protein group levels

Model	Protein Group Level	Average RMSE
STL	–	0.640 ± 0.084
FBMTL	Superclass	0.618 ± 0.066
IBMTL AA-GSS	Superclass	0.582 ± 0.104
IBMTL AA-LSS	Superclass	0.593 ± 0.110
FBMTL	Target Parent	0.561 ± 0.066
IBMTL AA-GSS	Target Parent	0.561 ± 0.071
IBMTL AA-LSS	Target Parent	0.559 ± 0.072

As a comparative analysis, protein-ligand docking was performed using the crystal structure of human AKT2 complexed with GSK690693 (PDB ID: 3D0E) to further evaluate the virtual screening results. Docking validation was carried out by redocking the co-crystallized ligand (GSK690693) into human AKT2, yielding the lowest root mean square deviation (RMSD) values of 0.22 Å for AutoDock Vina and 0.978 Å for MOE, thereby confirming the reliability of both docking protocols. Docking performance is generally considered acceptable when the RMSD between the crystal and redocked ligand poses is below 2 Å (Tallei et al., 2020). An overlay of the crystal structure and the corresponding redocked poses of the co-crystallized ligand is shown in Figure 3.10.

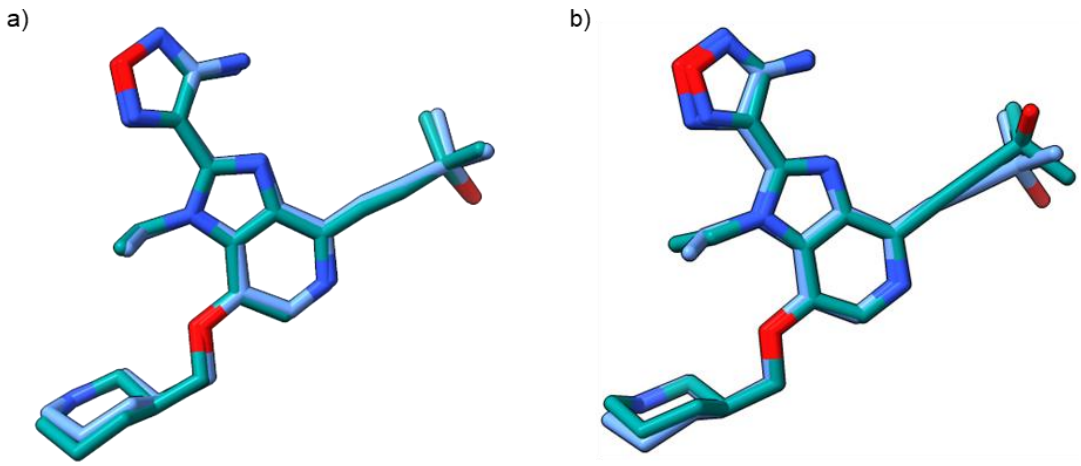


Figure 3.10. Overlay of the crystallographic pose (blue) and the best-ranked redocked pose (green) of the co-crystallized ligand (GSK690693). Redocking using (a) AutoDock Vina resulted in an RMSD of 0.22 Å, while (b) MOE produced an RMSD of 0.978 Å.

The binding affinity evaluation of the ten hit candidates was performed, and the results are presented in Table 3.2. Among these candidates, two compounds demonstrated favorable binding affinity scores relative to the reference compounds. STOCK1N-98616 and CMNPD23804 achieved scores of -9.9 and -9.38 kcal/mol using AutoDock Vina, and -8.4 and -9.40 kcal/mol using MOE, respectively.

Representative predicted natural products targeting AKT2 were selected from ChEMBL 35 to assess the relationship between bioactivity and binding affinity over a broad range of activity levels. Overall, the actual IC_{50} values of these compounds showed a consistent trend with their docking scores, with lower IC_{50} values generally associated with more favorable (more negative) binding affinity scores. For comparison, CHEMBL2177367, the representative predicted natural product with the lowest actual IC_{50}, showed binding affinity scores of -9.2 and -8.85 kcal/mol, while GSK690693 scored -10.3 and -9.57 kcal/mol. These results suggest that IBMTL AA-LSS can generate predictions that, for certain compounds, align

with docking-based evaluations. However, this observation is currently limited to AKT2, and further wet-laboratory validation is required to establish the model's reliability.

Table 3.2. Top ten natural products with the lowest predicted IC_{50} values targeting AKT2 screened using the best-performing model, together with reference compounds and their protein-ligand docking binding affinities

No.	Compound ID	Database Source	Compound Name	Molecular Formula	IC_{50} (nM)		Binding Affinity	
					Predicted	Actual	AutoDock Vina	MOE
Top Ten Natural Products with the Lowest Predicted IC_{50} Values Against AKT-2 Screened Using the Best-Performing Model								
1	STOCK1N-98616	InterBioScreen	–	$C_{37}H_{35}ClN_6O_8S$	4.92	–	-9.9	-9.38
2	CMNPD3014	CMNPD	Methoxydechlorochartelline A	$C_{21}H_{16}Br_4N_4O_2$	5.62	–	-8.4	-7.69
3	CMNPD2423	CMNPD	Chartelline A †	$C_{20}H_{13}Br_4ClN_4O$	5.64	–	-8.0	-6.94
4	CMNPD24363	CMNPD	Premarineosin A	$C_{25}H_{33}N_3O_2$	6.00	–	-8.1	-5.98
5	CMNPD24364	CMNPD	16-ketopremarineosin A	$C_{25}H_{31}N_3O_3$	6.30	–	-8.0	-6.41
6	CMNPD3015	CMNPD	Chartelline A	$C_{20}H_{13}Br_4ClN_4O$	6.32	–	-8.1	-5.99
7	CMNPD3013	CMNPD	Chartelline C	$C_{20}H_{15}Br_2ClN_4O$	6.60	–	-7.9	-6.22
8	STOCK1N-95086	InterBioScreen	–	$C_{23}H_{19}BrClNO_6S$	6.64	–	-8.2	-7.56
9	CMNPD3012	CMNPD	Chartelline B	$C_{20}H_{14}Br_3ClN_4O$	6.64	–	-8.4	-5.80
10	CMNPD23804	CMNPD	Atkamine A	$C_{40}H_{52}BrN_3O_3S$	6.69	–	-8.4	-9.40
Representative Predicted Natural Products in ChEMBL 35 Covering Low to High AKT2 Actual Activity Values								
11	CHEMBL2177367	ChEMBL	–	$C_{28}H_{38}ClN_5O_3$	–	10	-9.2	-8.85
12	CHEMBL2325740	ChEMBL	–	$C_{22}H_{27}ClN_6O_2$	–	130	-8.9	-8.58
13	CHEMBL227605	ChEMBL	–	$C_{19}H_{17}N_5$	–	430	-8.7	-7.19
14	CHEMBL227727	ChEMBL	–	$C_{13}H_{13}N_5$	–	1,600	-7.2	-6.50
15	CHEMBL264666	ChEMBL	–	$C_{10}H_{13}N_5$	–	59,000	-6.7	-6.06
Established Synthetic AKT2 Inhibitor								
16	GSK690693	PDB	–	$C_{21}H_{27}N_7O_3$	–	13	-10.3	-9.57

† An identifier to differentiate the compound from another with the same name but a distinct chemical structure.

3.6. References

Dou, B., Zhu, Z., Merkurjev, E., Ke, L., Chen, L., Jiang, J., Zhu, Y., Liu, J., Zhang, B., & Wei, G.-W. (2023). Machine Learning Methods for Small Data Challenges in Molecular Science. *Chemical Reviews*, *123*(13), 8736–8780. https://doi.org/10.1021/acs.chemrev.3c00189

Hanks, S. K., Quinn, A. M., & Hunter, T. (1988). The Protein Kinase Family: Conserved Features and Deduced Phylogeny of the Catalytic Domains. *Science*, *241*(4861), 42–52. https://doi.org/10.1126/science.3291115

López-Otín, C., & Bond, J. S. (2008). Proteases: Multifunctional Enzymes in Life and Disease*. *Journal of Biological Chemistry*, *283*(45), 30433–30437. https://doi.org/10.1074/jbc.R800035200

Moon, C., & Kim, D. (2022). Prediction of drug–target interactions through multi-task learning. *Scientific Reports*, *12*(1), 18323. https://doi.org/10.1038/s41598-022-23203-y

Simões, R. S., Maltarollo, V. G., Oliveira, P. R., & Honorio, K. M. (2018). Transfer and Multi-task Learning in QSAR Modeling: Advances and Challenges. *Frontiers in Pharmacology*, *9*. https://doi.org/10.3389/fphar.2018.00074

Tallei, T. E., Tumilaar, S. G., Niode, N. J., Kepel, B. J., Idroes, R., Effendi, Y., Sakib, S. A., & Emran, T. Bin. (2020). Potential of plant bioactive compounds as SARS-CoV-2 main protease (Mpro) and spike (S) glycoprotein inhibitors: a molecular docking study. *Scientifica*, *2020*. https://doi.org/10.1155/2020/6307457

Chapter 4

Conclusion

Protein evolutionary relatedness metrics, AA-GSS and AA-LSS, were incorporated as features into the IBMTL data framework. The predictive performance of IBMTL was evaluated against FMBTL, which lacks these features, and a baseline STL model. Model training and testing were performed using a limited dataset of predicted natural products with associated bioactivity data against target proteins, which are categorized under protein groups (level 2) within the enzyme family (level 1) of ChEMBL's six-level hierarchical protein classification.

Across most protein groups, IBMTL AA-GSS and IBMTL AA-LSS consistently surpassed FBMTL and STL, indicating that incorporating protein evolutionary relatedness features improves predictive performance. Statistically significant gains were primarily observed in protein groups with shared ancestry. Conversely, both models showed reduced performance on protein groups composed of heterogeneous, evolutionarily unrelated proteins. These results suggest that protein evolutionary relatedness plays a key role in determining the performance of IBMTL models.

Additional analyses focusing on the kinase group across three hierarchical protein group levels demonstrated that both FBMTL and IBMTL significantly outperformed STL. The highest predictive performance was obtained at the target parent level, indicating a trade-off between protein evolutionary similarity and the amount of available training data.

The effect of average protein similarity, protein count, and the availability of compound bioactivity data on model performance was further evaluated for the kinase group at the target parent level. Both FBMTL and IBMTL AA-LSS consistently exceeded STL when the average AA-LSS was above 30%. However, as protein similarity increased, the performance difference

between FBMTL and IBMTL AA-LSS tends to decrease. Comparable patterns were observed for both protein count and bioactivity data size: while FBMTL and IBMTL AA-LSS maintained superior performance over STL across all data regimes, the performance distinction between the two multi-task models became less pronounced under data-limited conditions.

Using the best-performing AKT2 model, bioactivity predictions were conducted for natural products collected from public databases, leading to the identification of ten candidates with the lowest predicted IC_{50} values. Notably, these candidates exhibited predicted IC_{50} values lower than the experimentally reported IC_{50} of GSK690693, a well-established synthetic AKT2 inhibitor. These ten candidates, together with reference compounds with known IC_{50} measurements, were further evaluated through protein-ligand docking simulations. For several compounds, the predicted activities were consistent with the docking outcomes, although this agreement was observed only for the AKT2 case study.

Limitations of the Study

The best-performing model, IBMTL AA-LSS at the target parent level, was able to identify natural products with potential inhibitory activity against AKT2 and showed better performance than the positive control GSK690693. Several natural products also exhibited predicted bioactivities that align with protein-ligand docking results. However, the robustness of the model still needs to be validated through wet-laboratory experiments as ground truth.

In addition, the best-performing model was applied only on AKT2 as a case study for predicting the bioactivity of natural product libraries. Consequently, the bioactivity prediction of natural products against other target proteins remains uncertain. Variations in protein families and differences in the amount of available protein and bioactivity data across protein groups may affect model performance.

Acknowledgements

This research was carried out at the Laboratory for Computational Biology, Institute for Protein Research, The University of Osaka, under the guidance and supervision of Professor Kenji Mizuguchi. I am deeply grateful for his continuous support, insightful advice, and encouragement throughout my research. I am likewise thankful to Dr. Reiko Watanabe, whose research inspired this work and whose mentorship and support were instrumental throughout this project. I also extend my appreciation to all current and former laboratory members, whom I cannot mention individually, for their support and assistance.

Furthermore, I would like to express my sincere gratitude to the Indonesia Endowment Fund for Education (LPDP), Republic of Indonesia, for supporting my doctoral scholarship. I would also like to thank my home institution, the National Research and Innovation Agency (BRIN), Republic of Indonesia, for providing the opportunity to pursue my doctoral degree.

Finally, this academic journey would not have been as meaningful as it is without the endless support of my family: my wife, Susi Latifah; my daughters, Rumaisha Aurum Nazihah and Nusaibah Rhenium Medina; and my parents, Asliani and Lina Haida. Words cannot adequately convey how deeply grateful I am for their unwavering love and patience.

List of Publications

*: Corresponding Author; †: Equal Contribution

Scientific Article (Main):

Ramadhan, D.[*], Watanabe, R., Mizuguchi, K.[*] (2025) Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. *ACS Omega*, 10 (37), 42776–42790. https://doi.org/10.1021/acsomega.5c05094.

Scientific Articles (Other):

Takadera, K.[†], **Ramadhan, D.**[*†], Watanabe, R.[*], Mizuguchi, K.* (2026) In Silico Regression Modelling and Improved Interpretability to Predict the Transport Inhibitory Activity of Breast Cancer Resistance Protein. (Manuscript Submitted for Publication).

Kustiawan, P. M.[†], Siregar, K. A. A. K.[†], Jauhar, M. M., **Ramadhan, D.**, Mardliyati, E., Syaifie, P. H.[*] (2024) Network Pharmacology and Bioinformatic Integrative Analysis Reveals Candidate Gene Targets and Potential Therapeutic of East Kalimantan Propolis against Hepatocellular Carcinoma. *Heliyon*, 10 (21). https://doi.org/10.1016/j.heliyon.2024.e39142.

Fatriani, R., Pratiwi, F. A. K., Annisa, A., Septaningsih, D. A., Aziz, S. A., Miladiyah, I., Kusumastuti, S. A., Nasution, M. A. F., **Ramadhan, D.**, Kusuma, W. A.[*] (2024) Unveiling the Anti-Obesity Potential of Kemuning (*Murraya paniculata*): A Network Pharmacology Approach. *PLoS One*, 19 (8), e0305544. https://doi.org/10.1371/journal.pone.0305544.

Syaifie, P. H.[*], Ibadillah, D., Jauhar, M. M., Reninta, R., Ningsih, S., **Ramadhan, D.**, Arda, A. G., Ningrum, D. W. C., Kaswati, N. M. N., Rochman, N. T., Mardliyati, E.[*] (2024) Phytochemical Profile, Antioxidant, Enzyme Inhibition, Acute Toxicity, In Silico Molecular Docking and Dynamic Analysis of *Apis mellifera* Propolis as Antidiabetic Supplement. *Chem Biodivers*, 21 (6), e202400433. https://doi.org/10.1002/cbdv.202400433.

Arda, A. G., Syaifie, P. H.[*], **Ramadhan, D.**, Jauhar, M. M., Nugroho, D. W., Kaswati, N. M. N., Noviyanto, A., Safihtri, M., Rochman, N. T., Andrianto, D., Mardliyati, E.[*] (2024) Activity of Propolis Compounds as Potential MMP1 and MMP2 Inhibitors by In Silico Studies in

Wound Healing Application. *J. Pharm. Pharmacogn. Res.*, 12 (2), 264-285. https://doi.org/10.56499/jppres23.1719_12.2.264.

Ningsih, S.*, Kusumastuti, S. A., Nuralih, N., Fajriawan, A. A., Permatasari, D., Yunianto, P.; **Ramadhan, D.**, Wulandari, M. T., Firdausi, N., Nurhadi, N., Giarni, R., Agustini, K., Wibowo, A. E., Rosidah, I., Rengganis, T. N., Ngatinem, N., Subiantoro, A. H., Supriyono, A. (2023) *Andrographis paniculata* (Burm. f.) Nees Extract Ameliorates Insulin Resistance in the Insulin-Resistant HepG2 Cells via GLUT2/IRS-1 Pathway. *Arch Physiol Biochem*, 130 (6), 779–789. https://doi.org/10.1080/13813455.2023.2273221.

Presentations:

Ramadhan, D., Watanabe, R., Mizuguchi, K. (2025) Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. Poster Presentation. *The 42nd Medicinal Chemistry Symposium*. Tokushima − Japan.

Ramadhan, D., Mizuguchi, K. (2024) Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. Oral Presentation. *Chem-Bio Informatics Society (CBI) Annual Meeting*. Tokyo − Japan.

Ramadhan, D., Mizuguchi, K. (2024) Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. Poster Presentation. *Life Intelligence Consortium (LINC) SHOWCASE*. Osaka − Japan.

Ramadhan, D., Mizuguchi, K. (2024) Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. Poster Presentation. *Institute for Protein Research (IPR) Retreat 2024*. Osaka − Japan.

Ramadhan, D., Mizuguchi, K. (2023) Optimizing Multi-Task Learning with Evolutionary Relatedness Metrics for Enhanced QSAR-Based Natural Product Activity Prediction. Poster Presentation. *Institute for Protein Research (IPR) Retreat 2023*. Osaka − Japan.

Supplementary Materials

Table S1. Number of bioactivity records per protein for predicted natural products in the kinase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	148	239	262	Glycogen synthase kinase-3 beta	Homo sapiens	5	539
2	148	239	2850	Glycogen synthase kinase-3 alpha	Homo sapiens	5	62
3	148	297	4224	Dual specificity protein kinase CLK1	Homo sapiens	5	94
4	148	297	4225	Dual specificity protein kinase CLK2	Homo sapiens	5	52
5	158	251	2185	Serine/threonine-protein kinase Aurora-B	Homo sapiens	5	224
6	158	251	4722	Serine/threonine-protein kinase Aurora-A	Homo sapiens	5	263
7	158	256	3835	Serine/threonine-protein kinase NEK2	Homo sapiens	5	55
8	158	256	5855	Serine/threonine-protein kinase Nek1	Homo sapiens	5	66
9	158	257	3024	Serine/threonine-protein kinase PLK1	Homo sapiens	5	346
10	158	257	3788	Serine/threonine-protein kinase PLK4	Homo sapiens	5	104
11	158	257	4897	Serine/threonine-protein kinase PLK3	Homo sapiens	5	70
12	158	259	5491	Serine/threonine-protein kinase WEE1	Homo sapiens	5	258
13	158	271	5443	Cell division cycle 7-related protein kinase	Homo sapiens	5	143
14	158	273	4896	PDZ-binding kinase	Homo sapiens	5	69
15	158	308	3983	Dual specificity protein kinase TTK	Homo sapiens	5	112
16	206	286	3582	Protein kinase C epsilon	Homo sapiens	6	153
17	206	286	3616	Protein kinase C eta	Homo sapiens	6	116
18	206	303	3438	Protein kinase C zeta	Homo sapiens	6	53
19	206	317	2938	Protein kinase C gamma	Homo sapiens	6	84
20	206	317	299	Protein kinase C alpha	Homo sapiens	6	239
21	206	317	3045	Protein kinase C beta	Homo sapiens	6	185
22	206	318	2996	Protein kinase C delta	Homo sapiens	6	309
23	206	318	3920	Protein kinase C theta	Homo sapiens	6	344
24	237	323	301	Cyclin-dependent kinase 2	Homo sapiens	6	451
25	237	323	308	Cyclin-dependent kinase 1	Homo sapiens	6	281

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
26	237	325	4036	Cyclin-dependent kinase 5	Homo sapiens	6	58
27	237	327	5719	Cell division protein kinase 8	Homo sapiens	6	66
28	237	328	3116	Cyclin-dependent kinase 9	Homo sapiens	6	566
29	249	329	2207	MAP kinase ERK2	Mus musculus	6	98
30	249	329	4040	MAP kinase ERK2	Homo sapiens	6	1031
31	249	330	2276	c-Jun N-terminal kinase 1	Homo sapiens	6	202
32	249	330	2637	c-Jun N-terminal kinase 3	Homo sapiens	6	105
33	249	330	4179	c-Jun N-terminal kinase 2	Homo sapiens	6	84
34	249	332	260	MAP kinase p38 alpha	Homo sapiens	6	347
35	257	397	5938	Serine/threonine-protein kinase PLK2	Homo sapiens	6	61
36	262	341	5749	Mitogen-activated protein kinase kinase kinase kinase 1	Homo sapiens	6	174
37	262	342	4527	TRAF2- and NCK-interacting kinase	Homo sapiens	6	66
38	262	401	4600	Serine/threonine-protein kinase PAK 1	Homo sapiens	6	54
39	262	402	4482	Serine/threonine-protein kinase PAK 4	Homo sapiens	6	66
40	305	450	2842	Serine/threonine-protein kinase mTOR	Homo sapiens	6	686
41	1100	148	331	Cyclin-dependent kinase 4	Homo sapiens	4	168
42	1100	158	1991	Inhibitor of nuclear factor kappa B kinase beta subunit	Homo sapiens	4	120
43	1100	158	3476	Inhibitor of nuclear factor kappa B kinase alpha subunit	Homo sapiens	4	61
44	1285	130	1862	Tyrosine-protein kinase ABL	Homo sapiens	5	88
45	1285	131	4599	Tyrosine kinase non-receptor protein 2	Homo sapiens	5	50
46	1285	133	4895	Tyrosine-protein kinase receptor UFO	Homo sapiens	5	149
47	1285	133	5314	Tyrosine-protein kinase receptor TYRO3	Homo sapiens	5	72
48	1285	133	5331	Proto-oncogene tyrosine-protein kinase MER	Homo sapiens	5	78
49	1285	135	2068	Ephrin type-A receptor 2	Homo sapiens	5	126
50	1285	135	5147	Ephrin type-B receptor 4	Homo sapiens	5	77
51	1285	136	2695	Focal adhesion kinase 1	Homo sapiens	5	288
52	1285	139	3717	Hepatocyte growth factor receptor	Homo sapiens	5	455
53	1285	140	2041	Tyrosine-protein kinase receptor RET	Homo sapiens	5	117

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
54	1285	141	1841	Tyrosine-protein kinase FYN	Homo sapiens	5	67
55	1285	141	258	Tyrosine-protein kinase LCK	Homo sapiens	5	311
56	1285	141	267	Tyrosine-protein kinase SRC	Homo sapiens	5	626
57	1285	141	3234	Tyrosine-protein kinase HCK	Homo sapiens	5	51
58	1285	142	2599	Tyrosine-protein kinase SYK	Homo sapiens	5	1101
59	1285	142	2803	Tyrosine-protein kinase ZAP-70	Homo sapiens	5	69
60	1285	143	2959	Tyrosine-protein kinase ITK/TSK	Homo sapiens	5	109
61	1285	143	3834	Tyrosine-protein kinase BMX	Homo sapiens	5	62
62	1285	143	5251	Tyrosine-protein kinase BTK	Homo sapiens	5	993
63	1285	144	4128	Tyrosine-protein kinase TIE-2	Homo sapiens	5	111
64	1285	146	2815	Nerve growth factor receptor Trk-A	Homo sapiens	5	295
65	1285	149	1824	Receptor protein-tyrosine kinase erbB-2	Homo sapiens	5	281
66	1285	149	203	Epidermal growth factor receptor erbB1	Homo sapiens	5	1046
67	1285	150	2742	Fibroblast growth factor receptor 3	Homo sapiens	5	68
68	1285	150	3650	Fibroblast growth factor receptor 1	Homo sapiens	5	334
69	1285	150	3973	Fibroblast growth factor receptor 4	Homo sapiens	5	99
70	1285	150	4142	Fibroblast growth factor receptor 2	Homo sapiens	5	83
71	1285	151	1957	Insulin-like growth factor I receptor	Homo sapiens	5	278
72	1285	151	1981	Insulin receptor	Homo sapiens	5	114
73	1285	152	2148	Tyrosine-protein kinase JAK3	Homo sapiens	5	1084
74	1285	152	2835	Tyrosine-protein kinase JAK1	Homo sapiens	5	918
75	1285	152	2971	Tyrosine-protein kinase JAK2	Homo sapiens	5	1216
76	1285	152	3553	Tyrosine-protein kinase TYK2	Homo sapiens	5	385
77	1285	159	1844	Macrophage colony stimulating factor receptor	Homo sapiens	5	153
78	1285	159	1913	Platelet-derived growth factor receptor beta	Homo sapiens	5	246
79	1285	159	1936	Stem cell growth factor receptor	Homo sapiens	5	236
80	1285	159	1974	Tyrosine-protein kinase receptor FLT3	Homo sapiens	5	439
81	1285	159	2007	Platelet-derived growth factor receptor alpha	Homo sapiens	5	135
82	1285	160	1868	Vascular endothelial growth factor receptor 1	Homo sapiens	5	142
83	1285	160	1955	Vascular endothelial growth factor receptor 3	Homo sapiens	5	105
84	1285	160	279	Vascular endothelial growth factor receptor 2	Homo sapiens	5	1034

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
85	1285	1614	5319	Epithelial discoidin domain-containing receptor 1	Homo sapiens	5	50
86	1289	203	2431	Serine/threonine-protein kinase AKT2	Homo sapiens	5	234
87	1289	203	4282	Serine/threonine-protein kinase AKT	Homo sapiens	5	674
88	1289	203	4816	Serine/threonine-protein kinase AKT3	Homo sapiens	5	122
89	1289	205	4101	cAMP-dependent protein kinase alpha-catalytic subunit	Homo sapiens	5	50
90	1289	232	2534	3-phosphoinositide dependent protein kinase-1	Homo sapiens	5	81
91	1301	210	4937	Casein kinase I epsilon	Homo sapiens	5	83
92	1331	298	3587	Dual specificity mitogen-activated protein kinase kinase 1	Homo sapiens	5	122
93	1331	1714	5888	Mitogen-activated protein kinase kinase kinase 14	Homo sapiens	5	53
94	1334	240	1075104	Leucine-rich repeat serine/threonine-protein kinase 2	Homo sapiens	5	210
95	1334	1638	1906	Serine/threonine-protein kinase RAF	Homo sapiens	5	251
96	1334	1638	5145	Serine/threonine-protein kinase B-raf	Homo sapiens	5	396
97	1334	1674	3778	Interleukin-1 receptor-associated kinase 4	Homo sapiens	5	383
98	1339	235	2147	Serine/threonine-protein kinase PIM1	Homo sapiens	5	888
99	1339	235	4523	Serine/threonine-protein kinase PIM2	Homo sapiens	5	714
100	1339	235	5407	Serine/threonine-protein kinase PIM3	Homo sapiens	5	292
101	1339	246	3062	Myosin light chain kinase, smooth muscle	Gallus gallus	5	82
102	1452	119	3861	Pyruvate dehydrogenase kinase isoform 2	Homo sapiens	5	428
103	1452	305	3142	DNA-dependent protein kinase	Homo sapiens	5	426
104	1452	305	3797	Serine-protein kinase ATM	Homo sapiens	5	64
105	1452	305	5024	Serine-protein kinase ATR	Homo sapiens	5	132
106	1618	302	3711550	Beta-adrenergic receptor kinase 1	Bos taurus	6	69
107	1623	307	5776	Mitogen-activated protein kinase kinase kinase 7	Homo sapiens	6	70
108	1648	288	4501	Ribosomal protein S6 kinase 1	Homo sapiens	6	60
109	1648	289	2345	Ribosomal protein S6 kinase alpha 3	Homo sapiens	6	128

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
110	1648	289	3906	Ribosomal protein S6 kinase alpha 2	Homo sapiens	6	202
111	1668	315	2973	Rho-associated protein kinase 2	Homo sapiens	6	472
112	1668	315	3231	Rho-associated protein kinase 1	Homo sapiens	6	170
113	1669	465	2292	Dual-specificity tyrosine-phosphorylation regulated kinase 1A	Homo sapiens	6	329
114	1669	465	5508	Dual specificity tyrosine-phosphorylation-regulated kinase 1A	Rattus norvegicus	6	80
115	1669	465	5543	Dual specificity tyrosine-phosphorylation-regulated kinase 1B	Homo sapiens	6	114
116	1678	348	3836	LIM domain kinase 1	Homo sapiens	6	76
117	1678	348	5932	LIM domain kinase 2	Homo sapiens	6	121
118	1693	404	4247	ALK tyrosine kinase receptor	Homo sapiens	6	622
119	1693	404	4439	TGF-beta receptor type I	Homo sapiens	6	51
120	1702	411	4630	Serine/threonine-protein kinase Chk1	Homo sapiens	6	403
121	1713	428	4204	MAP kinase signal-integrating kinase 2	Homo sapiens	6	57
122	1713	428	4718	MAP kinase-interacting serine/threonine-protein kinase MNK1	Homo sapiens	6	51
123	1713	472	2208	MAP kinase-activated protein kinase 2	Homo sapiens	6	279

Table S2. Number of bioactivity records per protein for predicted natural products in the protease group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	59	69	1892	Glutamate carboxypeptidase II	Homo sapiens	6	166
2	904	912	1932910	ATP-dependent Clp protease proteolytic subunit	Staphylococcus aureus (strain NCTC 8325)	5	60
3	1028	1079	2176771	Complement factor D	Homo sapiens	3	945
4	1028	1085	4296327	Protease	Human T-cell leukemia virus type I	3	69
5	1028	1087	5362	Sortase A	Staphylococcus aureus	3	72

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
6	1106	47	2046264	Renin	Macaca fascicularis	5	60
7	1106	47	2322	Renin	Rattus norvegicus	5	85
8	1106	47	2525	Beta secretase 2	Homo sapiens	5	189
9	1106	47	2581	Cathepsin D	Homo sapiens	5	806
10	1106	47	2714	Pepsin A	Sus scrofa	5	55
11	1106	47	286	Renin	Homo sapiens	5	2192
12	1106	47	4106	Cathepsin D	Bos taurus	5	138
13	1106	47	4414	Plasmepsin 2	Plasmodium falciparum	5	87
14	1106	47	4822	Beta-secretase 1	Homo sapiens	5	2906
15	1106	47	6056	Renin	Callithrix jacchus	5	78
16	1106	48	243	Human immunodeficiency virus type 1 protease	Human immunodeficiency virus 1	5	1743
17	1109	49	3988603	Genome polyprotein	Hepatitis C virus genotype 1b (isolate Japanese) (HCV)	5	313
18	1109	49	4620	Hepatitis C virus polyprotein	Hepatitis C virus genotype 1a (isolate 1) (HCV)	5	92
19	1110	28	3891	Calpain 1	Homo sapiens	5	209
20	1110	28	4143	Calpain 2	Sus scrofa	5	98
21	1110	50	2252	Dipeptidyl peptidase I	Homo sapiens	5	407
22	1110	50	268	Cathepsin K	Homo sapiens	5	334
23	1110	50	2954	Cathepsin S	Homo sapiens	5	279
24	1110	50	3563	Cruzipain	Trypanosoma cruzi	5	82
25	1110	50	3837	Cathepsin L	Homo sapiens	5	284
26	1110	50	4072	Cathepsin B	Homo sapiens	5	215
27	1110	50	4188	Rhodesain	Trypanosoma brucei rhodesiense	5	72
28	1110	50	5800	Falcipain 2	Plasmodium falciparum	5	97
29	1110	74	2157850	Ubiquitin carboxyl-terminal hydrolase 7	Homo sapiens	5	57
30	1111	52	4244	Legumain	Homo sapiens	5	78
31	1111	53	2334	Caspase-3	Homo sapiens	5	410
32	1111	53	3308	Caspase-6	Homo sapiens	5	70

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
33	1111	53	3468	Caspase-7	Homo sapiens	5	131
34	1111	53	3776	Caspase-8	Homo sapiens	5	202
35	1111	53	4801	Caspase-1	Homo sapiens	5	316
36	1114	62	3419	Carboxypeptidase B2 isoform A	Homo sapiens	5	163
37	1116	65	2474	Methionine aminopeptidase 1	Homo sapiens	5	60
38	1116	65	3922	Methionine aminopeptidase 2	Homo sapiens	5	168
39	1120	18	6115	Lysosomal protective protein	Homo sapiens	5	69
40	1120	19	2335	Lysosomal Pro-X carboxypeptidase	Homo sapiens	5	89
41	1120	19	3976	Dipeptidyl peptidase II	Homo sapiens	5	363
42	1120	20	2409	Epoxide hydratase	Homo sapiens	5	261
43	1120	20	4140	Epoxide hydratase	Mus musculus	5	67
44	1120	21	3202	Prolyl endopeptidase	Homo sapiens	5	169
45	1120	22	284	Dipeptidyl peptidase IV	Homo sapiens	5	1157
46	1120	22	4657	Dipeptidyl peptidase VIII	Homo sapiens	5	391
47	1120	22	4683	Fibroblast activation protein alpha	Homo sapiens	5	136
48	1120	22	4793	Dipeptidyl peptidase IX	Homo sapiens	5	228
49	1123	27	3771	Human herpes virus 5 capsid protein P40	Human cytomegalovirus (strain AD169) (HHV-5) (Human herpesvirus 5)	5	84
50	1130	31	1907	Aminopeptidase N	Homo sapiens	5	196
51	1130	31	2590	Aminopeptidase N	Sus scrofa	5	399
52	1130	31	2693	Cystinyl aminopeptidase	Homo sapiens	5	71
53	1130	31	4117	Zinc aminopeptidase	Plasmodium falciparum FcB1/Columbia	5	100
54	1130	31	4618	Leukotriene A4 hydrolase	Homo sapiens	5	78
55	1130	31	5043	Endoplasmic reticulum aminopeptidase 2	Homo sapiens	5	111
56	1130	31	5939	Endoplasmic reticulum aminopeptidase 1	Homo sapiens	5	93
57	1130	33	1808	Angiotensin-converting enzyme	Homo sapiens	5	558
58	1130	33	2625	Angiotensin-converting enzyme	Rattus norvegicus	5	192
59	1130	33	3736	Angiotensin-converting enzyme 2	Homo sapiens	5	53
60	1130	33	4074	Angiotensin-converting enzyme	Oryctolagus cuniculus	5	430

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
61	1130	55	1944	Neprilysin	Homo sapiens	5	225
62	1130	55	2676	Endothelin-converting enzyme 1	Rattus norvegicus	5	70
63	1130	55	3369	Neprilysin	Rattus norvegicus	5	214
64	1130	55	3768	Neprilysin	Oryctolagus cuniculus	5	146
65	1130	55	4791	Endothelin-converting enzyme 1	Homo sapiens	5	189
66	1130	57	5192	Botulinum neurotoxin type A	Clostridium botulinum	5	64
67	1131	82	280	Matrix metalloproteinase 13	Homo sapiens	5	553
68	1131	82	283	Matrix metalloproteinase 3	Homo sapiens	5	572
69	1131	82	321	Matrix metalloproteinase 9	Homo sapiens	5	436
70	1131	82	332	Matrix metalloproteinase-1	Homo sapiens	5	630
71	1131	82	333	Matrix metalloproteinase-2	Homo sapiens	5	1002
72	1131	82	3869	Matrix metalloproteinase 14	Homo sapiens	5	180
73	1131	82	4073	Matrix metalloproteinase 7	Homo sapiens	5	151
74	1131	82	4393	Matrix metalloproteinase 12	Homo sapiens	5	174
75	1131	82	4588	Matrix metalloproteinase 8	Homo sapiens	5	234
76	1131	83	3898	Bone morphogenetic protein 1	Homo sapiens	5	97
77	1131	84	2285	ADAMTS5	Homo sapiens	5	63
78	1131	84	2318	ADAMTS4	Homo sapiens	5	89
79	1131	84	3332	ADAM17	Sus scrofa	5	168
80	1131	84	3706	ADAM17	Homo sapiens	5	457
81	1131	84	5028	ADAM10	Homo sapiens	5	94
82	1135	78	2366517	Protease	Human immunodeficiency virus 1	5	410
83	1135	78	2857	Human rhinovirus A protease	Human rhinovirus sp.	5	138
84	1136	38	1801	Plasminogen	Homo sapiens	5	225
85	1136	38	2000	Plasma kallikrein	Homo sapiens	5	220
86	1136	38	204	Thrombin	Homo sapiens	5	704
87	1136	38	209	Trypsin I	Homo sapiens	5	195
88	1136	38	244	Coagulation factor X	Homo sapiens	5	630
89	1136	38	2443	Kallikrein 7	Homo sapiens	5	50
90	1136	38	248	Leukocyte elastase	Homo sapiens	5	467

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
91	1136	38	2820	Coagulation factor XI	Homo sapiens	5	190
92	1136	38	2821	Coagulation factor XII	Homo sapiens	5	91
93	1136	38	3286	Urokinase-type plasminogen activator	Homo sapiens	5	55
94	1136	38	3769	Trypsin I	Bos taurus	5	82
95	1136	38	3991	Coagulation factor VII	Homo sapiens	5	65
96	1136	38	4068	Chymase	Homo sapiens	5	144
97	1136	38	4071	Cathepsin G	Homo sapiens	5	54
98	1136	38	4471	Thrombin	Bos taurus	5	108
99	1136	38	5731	Complement factor B	Homo sapiens	5	53
100	1137	89	3317334	Proteasome subunit beta type-10	Homo sapiens	5	59
101	1137	89	3492	Proteasome Macropain subunit	Homo sapiens	5	146
102	1137	89	4208	Proteasome component C5	Homo sapiens	5	123
103	1137	89	4662	Proteasome Macropain subunit MB1	Homo sapiens	5	612
104	1137	89	5620	Proteasome subunit beta type-8	Homo sapiens	5	225
105	1137	925	1944495	Proteasome subunit beta type-9	Homo sapiens	5	116

Table S3. Number of bioactivity records per protein for predicted natural products in the hydrolase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	646	1075097	Arginase-1	Homo sapiens	2	161
2	1	646	1169598	Polymerase acidic protein	Influenza A virus (A/Puerto Rico/8/1934(H1N1))	2	140
3	1	646	1287610	Neuraminidase	Influenza A virus (strain A/Wilson-Smith/1933 H1N1) (Influenza A virus(strain A/WS/1933 H1N1))	2	53
4	1	646	1293246	Beta Lactamase	Pseudomonas aeruginosa	2	62
5	1	646	1293313	M17 leucyl aminopeptidase	Plasmodium falciparum 3D7	2	86
6	1	646	1641350	Beta-lactamase	Pseudomonas aeruginosa	2	109

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
7	1	646	1641358	Beta-lactamase	Klebsiella pneumoniae	2	109
8	1	646	1667684	Neuraminidase	Influenza A virus	2	53
9	1	646	1687677	Pancreatic triacylglycerol lipase	Sus scrofa	2	76
10	1	646	1741179	Probable DNA dC→dU-editing enzyme APOBEC-3A	Homo sapiens	2	52
11	1	646	1741217	DNA dC→dU-editing enzyme APOBEC-3G	Homo sapiens	2	59
12	1	646	1914	Butyrylcholinesterase	Homo sapiens	2	1461
13	1	646	2010635	Peptide deformylase	Staphylococcus aureus	2	66
14	1	646	2026	Beta-lactamase AmpC	Escherichia coli K-12	2	103
15	1	646	2034805	N-acylethanolamine-hydrolyzing acid amidase	Rattus norvegicus	2	79
16	1	646	2051	Neuraminidase	Influenza A virus (A/Puerto Rico/8/1934(H1N1))	2	430
17	1	646	2065	Beta-lactamase TEM	Escherichia coli	2	95
18	1	646	2074	Maltase-glucoamylase	Homo sapiens	2	155
19	1	646	2189121	GTPase KRas	Homo sapiens	2	382
20	1	646	220	Acetylcholinesterase	Homo sapiens	2	2521
21	1	646	2243	Anandamide amidohydrolase	Homo sapiens	2	98
22	1	646	247	Human immunodeficiency virus type 1 reverse transcriptase	Human immunodeficiency virus 1	2	1080
23	1	646	2528	Butyrylcholinesterase	Mus musculus	2	116
24	1	646	2536	Phospholipase D1	Homo sapiens	2	69
25	1	646	2608	Lysosomal alpha-glucosidase	Homo sapiens	2	57
26	1	646	2664	Adenosylhomocysteinase	Homo sapiens	2	63
27	1	646	2725	Beta-lactamase	Enterobacter cloacae	2	77
28	1	646	2726	Sialidase 1	Homo sapiens	2	63
29	1	646	2728	Beta-glucuronidase	Homo sapiens	2	80
30	1	646	2748	Sucrase-isomaltase	Homo sapiens	2	59
31	1	646	3046	Sialidase 3	Homo sapiens	2	71
32	1	646	3114	Sucrase-isomaltase	Rattus norvegicus	2	221
33	1	646	3180	Carboxylesterase 2	Homo sapiens	2	65
34	1	646	3198	Acetylcholinesterase	Mus musculus	2	239

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
35	1	646	3199	Acetylcholinesterase	Rattus norvegicus	2	704
36	1	646	3200	Sialidase 2	Homo sapiens	2	81
37	1	646	3217380	Beta-glucuronidase	Escherichia coli (strain K12)	2	50
38	1	646	3229	Anandamide amidohydrolase	Rattus norvegicus	2	176
39	1	646	3377	Neuraminidase	Influenza B virus (strain B/Lee/1940)	2	161
40	1	646	3403	Butyrylcholinesterase	Rattus norvegicus	2	210
41	1	646	3471	Human immunodeficiency virus type 1 integrase	Human immunodeficiency virus 1	2	1271
42	1	646	3474	Phospholipase A2 group IIA	Homo sapiens	2	230
43	1	646	3482	Beta-galactosidase	Bos taurus	2	109
44	1	646	3513	Acidic alpha-glucosidase	Rattus norvegicus	2	162
45	1	646	3545	Alpha-L-fucosidase 1	Bos taurus	2	56
46	1	646	3559	Steryl-sulfatase	Homo sapiens	2	264
47	1	646	3559643	Neuraminidase	Influenza A virus (strain A/USSR/90/1977 H1N1)	2	52
48	1	646	3580505	Toxin B	Peptoclostridium difficile	2	70
49	1	646	3590	Hormone sensitive lipase	Homo sapiens	2	97
50	1	646	3638350	Collagenase 3	Mus musculus	2	57
51	1	646	3855	UDP-3-O-acyl-GlcNAc deacetylase	Pseudomonas aeruginosa (strain ATCC 15692 / PAO1 / 1C / PRS 101 / LMG12228)	2	73
52	1	646	3921	Heparanase	Homo sapiens	2	60
53	1	646	4078	Acetylcholinesterase	Electrophorus electricus	2	2575
54	1	646	4161	Urease	Canavalia ensiformis	2	54
55	1	646	4174	Sialidase 4	Homo sapiens	2	64
56	1	646	4176	Alpha-L-fucosidase I	Homo sapiens	2	58
57	1	646	4191	Monoglyceride lipase	Homo sapiens	2	82
58	1	646	4342	Group X secretory phospholipase A2	Homo sapiens	2	58
59	1	646	4349	N-acylsphingosine-amidohydrolase	Homo sapiens	2	105
60	1	646	4426	Phospholipase A2 group 1B	Homo sapiens	2	76

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
61	1	646	4523156	Ubiquitin carboxyl-terminal hydrolase 19	Homo sapiens	2	281
62	1	646	4523419	Serine protease HTRA1	Homo sapiens	2	161
63	1	646	4768	Acetylcholinesterase	Bos taurus	2	205
64	1	646	4780	Acetylcholinesterase	Torpedo californica	2	110
65	1	646	5031	Beta-lactamase	Pseudomonas aeruginosa (strain ATCC 15692 / PAO1 / 1C / PRS 101 / LMG12228)	2	51
66	1	646	5077	Butyrylcholinesterase	Equus caballus	2	186
67	1	646	5189	Sialidase	Clostridium perfringens	2	115
68	1	646	5244	UDP-3-O-acyl-GlcNAc deacetylase	Escherichia coli K-12	2	92
69	1	646	5463	Acid ceramidase	Homo sapiens	2	65
70	1	646	5763	Cholinesterase	Equus caballus	2	1857
71	1	646	5848	Oligo-1,6-glucosidase	Saccharomyces cerevisiae S288c	2	66
72	1	646	5956	Beta-lactamase SHV-1	Escherichia coli	2	55
73	1	646	6135	Neuraminidase	Influenza A virus	2	449

Table S4. Number of bioactivity records per protein for predicted natural products in the oxidoreductase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	10	1787	Steroid 5-alpha-reductase 1	Homo sapiens	2	206
2	1	10	1849	Enoyl-[acyl-carrier-protein] reductase	Mycobacterium tuberculosis	2	50
3	1	10	1856	Steroid 5-alpha-reductase 2	Homo sapiens	2	281
4	1	10	1900	Aldose reductase	Homo sapiens	2	334
5	1	10	1926	Dihydrofolate reductase	Pneumocystis carinii	2	88
6	1	10	1929	Xanthine dehydrogenase	Homo sapiens	2	141
7	1	10	1935	Aldehyde dehydrogenase	Homo sapiens	2	92
8	1	10	1951	Monoamine oxidase A	Homo sapiens	2	747
9	1	10	1966	Dihydroorotate dehydrogenase	Homo sapiens	2	78

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
10	1	10	1973	Tyrosinase	Homo sapiens	2	161
11	1	10	2002	Inosine-5'-monophosphate dehydrogenase 2	Homo sapiens	2	162
12	1	10	2007625	Isocitrate dehydrogenase [NADP] cytoplasmic	Homo sapiens	2	256
13	1	10	202	Dihydrofolate reductase	Homo sapiens	2	171
14	1	10	2039	Monoamine oxidase B	Homo sapiens	2	1385
15	1	10	2140	Tryptophan 2,3-dioxygenase	Homo sapiens	2	202
16	1	10	2145	Kynurenine 3-monooxygenase	Homo sapiens	2	52
17	1	10	215	Arachidonate 5-lipoxygenase	Homo sapiens	2	633
18	1	10	221	Cyclooxygenase-1	Homo sapiens	2	359
19	1	10	2284	Glyceraldehyde-3-phosphate dehydrogenase liver	Homo sapiens	2	102
20	1	10	230	Cyclooxygenase-2	Homo sapiens	2	581
21	1	10	2311243	D-3-phosphoglycerate dehydrogenase	Homo sapiens	2	162
22	1	10	2363	Dihydrofolate reductase	Rattus norvegicus	2	150
23	1	10	2425	Dihydrofolate reductase	Toxoplasma gondii	2	112
24	1	10	2439	Myeloperoxidase	Homo sapiens	2	73
25	1	10	2622	Aldose reductase	Rattus norvegicus	2	360
26	1	10	2789	Estradiol 17-beta-dehydrogenase 2	Homo sapiens	2	73
27	1	10	2902	Dihydrofolate reductase	Lactobacillus casei	2	69
28	1	10	2903	Arachidonate 15-lipoxygenase	Homo sapiens	2	74
29	1	10	2949	Cyclooxygenase-1	Ovis aries	2	235
30	1	10	2980	Arachidonate 5-lipoxygenase	Bos taurus	2	85
31	1	10	2993	Monoamine oxidase B	Rattus norvegicus	2	298
32	1	10	3028	Hypoxia-inducible factor prolyl hydroxylase 1	Homo sapiens	2	175
33	1	10	3081	Aldose reductase	Bos taurus	2	155
34	1	10	312	Arachidonate 5-lipoxygenase	Rattus norvegicus	2	400
35	1	10	3181	Estradiol 17-beta-dehydrogenase 1	Homo sapiens	2	125
36	1	10	3247	HMG-CoA reductase	Rattus norvegicus	2	209
37	1	10	3318	Tyrosinase	Agaricus bisporus	2	353
38	1	10	3358	Monoamine oxidase A	Rattus norvegicus	2	211
39	1	10	3490	11-beta-hydroxysteroid dehydrogenase 2	Mus musculus	2	51

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
40	1	10	3568	Nitric-oxide synthase, brain	Homo sapiens	2	105
41	1	10	3623	Quinone reductase 1	Homo sapiens	2	87
42	1	10	3649	Xanthine dehydrogenase	Bos taurus	2	245
43	1	10	3746	11-beta-hydroxysteroid dehydrogenase 2	Homo sapiens	2	120
44	1	10	3813586	Dehydrosqualene desaturase	Staphylococcus aureus (strain Newman)	2	71
45	1	10	3840	Ribonucleoside-diphosphate reductase large chain	Herpes simplex virus (type 1 / strain 17)	2	63
46	1	10	3910	11-beta-hydroxysteroid dehydrogenase 1	Mus musculus	2	380
47	1	10	3959	Quinone reductase 2	Homo sapiens	2	64
48	1	10	3988583	Sepiapterin reductase	Homo sapiens	2	111
49	1	10	3994	Enoyl-[acyl-carrier-protein] reductase (FabI)	Staphylococcus aureus	2	99
50	1	10	402	HMG-CoA reductase	Homo sapiens	2	113
51	1	10	4102	Cyclooxygenase-2	Ovis aries	2	116
52	1	10	4234	Estradiol 17-beta-dehydrogenase 3	Homo sapiens	2	87
53	1	10	4235	11-beta-hydroxysteroid dehydrogenase 1	Homo sapiens	2	1533
54	1	10	4295843	Dihydroorotate dehydrogenase (fumarate)	Leishmania major	2	53
55	1	10	4358	Arachidonate 15-lipoxygenase	Oryctolagus cuniculus	2	83
56	1	10	4559	Aldose reductase	Sus scrofa	2	51
57	1	10	4681	Aldo-keto-reductase family 1 member C3	Homo sapiens	2	173
58	1	10	4685	Indoleamine 2,3-dioxygenase	Homo sapiens	2	669
59	1	10	5347	Glucose-6-phosphate 1-dehydrogenase	Homo sapiens	2	58
60	1	10	5689	Tryptophan 5-hydroxylase 1	Homo sapiens	2	58
61	1	10	5697	Egl nine homolog 1	Homo sapiens	2	426
62	1	10	5847	Aldo-keto reductase family 1 member C2	Homo sapiens	2	68
63	1	10	5983	Aldo-keto reductase family 1 member B10	Homo sapiens	2	65

Table S5. Number of bioactivity records per protein for predicted natural products in the transferase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	643	1075284	Diacylglycerol O-acyltransferase 1	Mus musculus	2	80
2	1	643	1770035	Phospho-N-acetylmuramoyl-pentapeptide-transferase	Staphylococcus aureus	2	66
3	1	643	1782	Farnesyl diphosphate synthase	Homo sapiens	2	56
4	1	643	1820	Thymidine kinase	Human herpesvirus 1 (strain SC16) (HHV-1) (Human herpes simplex virus1)	2	65
5	1	643	1952	Thymidylate synthase	Homo sapiens	2	154
6	1	643	2063	Ceramide glucosyltransferase	Homo sapiens	2	91
7	1	643	2321630	Alpha-1,6-mannosyl-glycoprotein 2-beta-N-acetylglucosaminyltransferase	Homo sapiens	2	53
8	1	643	2346486	Nicotinamide N-methyltransferase	Homo sapiens	2	118
9	1	643	2568	Liver glycogen phosphorylase	Homo sapiens	2	276
10	1	643	2782	Acyl coenzyme A:cholesterol acyltransferase 1	Homo sapiens	2	173
11	1	643	285	Acyl coenzyme A:cholesterol acyltransferase 1	Rattus norvegicus	2	99
12	1	643	2916	Telomerase reverse transcriptase	Homo sapiens	2	163
13	1	643	3105	Poly [ADP-ribose] polymerase-1	Homo sapiens	2	1105
14	1	643	3130	PI3-kinase p110-delta subunit	Homo sapiens	2	480
15	1	643	3145	PI3-kinase p110-beta subunit	Homo sapiens	2	332
16	1	643	3160	Thymidylate synthase	Mus musculus	2	258
17	1	643	3267	PI3-kinase p110-gamma subunit	Homo sapiens	2	358
18	1	643	3286067	DNA-directed RNA polymerase I subunit RPA1	Homo sapiens	2	55
19	1	643	3338	Squalene synthetase	Homo sapiens	2	85
20	1	643	3548	Peptide N-myristoyltransferase 1	Candida albicans (strain SC5314 / ATCC MYA-2876) (Yeast)	2	53
21	1	643	3638344	Genome polyprotein	Hepatitis C virus genotype 1a (isolate H) (HCV)	2	556

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
22	1	643	3815	Squalene synthetase	Rattus norvegicus	2	325
23	1	643	3957	Phospho-N-acetylmuramoyl-pentapeptide-transferase	Escherichia coli K-12	2	112
24	1	643	4005	PI3-kinase p110-alpha subunit	Homo sapiens	2	709
25	1	643	4158	Fatty acid synthase	Homo sapiens	2	169
26	1	643	4295747	Serine hydroxymethyltransferase, mitochondrial	Homo sapiens	2	58
27	1	643	4296320	RNA-directed RNA polymerase	Hepacivirus hominis	2	334
28	1	643	4370	UDP-glucuronosyltransferase 2B7	Homo sapiens	2	60
29	1	643	4465	Acyl coenzyme A:cholesterol acyltransferase 2	Homo sapiens	2	167
30	1	643	4523418	Inositol hexakisphosphate kinase 1	Homo sapiens	2	56
31	1	643	4523442	N-terminal Xaa-Pro-Lys N-methyltransferase 1	Homo sapiens	2	66
32	1	643	4523582	Replicase polyprotein 1ab	Severe acute respiratory syndrome coronavirus 2	2	1123
33	1	643	4696	Glycogen phosphorylase, muscle form	Oryctolagus cuniculus	2	285
34	1	643	4699	Isoprenylcysteine carboxyl methyltransferase	Homo sapiens	2	169
35	1	643	4739695	N6-adenosine-methyltransferase catalytic subunit	Homo sapiens	2	258
36	1	643	4879459	E3 ubiquitin-protein ligase CBL-B	Homo sapiens	2	66
37	1	643	5366	Poly [ADP-ribose] polymerase 2	Homo sapiens	2	288
38	1	643	5704	Elongation of very long chain fatty acids protein 6	Homo sapiens	2	81
39	1	643	5853	Diacylglycerol O-acyltransferase 2	Homo sapiens	2	198
40	1	643	5879	Hematopoietic prostaglandin D synthase	Homo sapiens	2	84
41	1	643	6009	Diacylglycerol O-acyltransferase 1	Homo sapiens	2	220
42	1	643	6141	Acyl-CoA:cholesterol acyltransferase	Oryctolagus cuniculus	2	106
43	1	643	6154	Tankyrase-2	Homo sapiens	2	287
44	1	643	6164	Tankyrase-1	Homo sapiens	2	391
45	1	643	6187	MecA	Staphylococcus aureus	2	71

Table S6. Number of bioactivity records per protein for predicted natural products in the phosphatase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	1030	3975	Fructose-1,6-bisphosphatase	Homo sapiens	2	73
2	1	1030	5957	5'-nucleotidase	Homo sapiens	2	150
3	1	1030	5979	Alkaline phosphatase, tissue-nonspecific isozyme	Homo sapiens	2	62
4	1103	156	2219	Protein-tyrosine phosphatase LC-PTP	Homo sapiens	4	55
5	1103	156	2889	Hematopoietic cell protein-tyrosine phosphatase 70Z-PEP	Homo sapiens	4	91
6	1103	156	3166	Protein-tyrosine phosphatase 1C	Homo sapiens	4	84
7	1103	156	3243	Leukocyte common antigen	Homo sapiens	4	76
8	1103	156	335	Protein-tyrosine phosphatase 1B	Homo sapiens	4	1388
9	1103	156	3807	T-cell protein-tyrosine phosphatase	Homo sapiens	4	219
10	1103	156	3864	Protein-tyrosine phosphatase 2C	Homo sapiens	4	150
11	1103	222	1795098	Carboxy-terminal domain RNA polymerase II polypeptide A small phosphatase 1	Homo sapiens	4	118
12	1103	279	2378	Dual specificity phosphatase Cdc25C	Homo sapiens	4	52
13	1103	279	2635	Dual specificity protein phosphatase 3	Homo sapiens	4	82
14	1103	279	3775	Dual specificity phosphatase Cdc25A	Homo sapiens	4	145
15	1103	279	4804	Dual specificity phosphatase Cdc25B	Homo sapiens	4	211

Table S7. Number of bioactivity records per protein for predicted natural products in the cytochrome P450 group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1179	167	2231	Cytochrome P450 1A1	Homo sapiens	5	227
2	1179	168	3356	Cytochrome P450 1A2	Homo sapiens	5	311
3	1180	169	4878	Cytochrome P450 1B1	Homo sapiens	5	243
4	1183	175	3721	Cytochrome P450 2C8	Homo sapiens	5	107
5	1183	176	3397	Cytochrome P450 2C9	Homo sapiens	5	584

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
6	1183	197	3622	Cytochrome P450 2C19	Homo sapiens	5	338
7	1184	180	289	Cytochrome P450 2D6	Homo sapiens	5	753
8	1186	184	340	Cytochrome P450 3A4	Homo sapiens	5	1467
9	1189	190	1835	Thromboxane-A synthase	Homo sapiens	5	218
10	1219	224	1908	Cytochrome P450 11B1	Homo sapiens	5	212
11	1219	225	2722	Cytochrome P450 11B2	Homo sapiens	5	259
12	1220	226	3522	Cytochrome P450 17A1	Homo sapiens	5	246
13	1220	226	4430	Cytochrome P450 17A1	Rattus norvegicus	5	95
14	1221	227	1978	Cytochrome P450 19A1	Homo sapiens	5	1326

Table S8. Number of bioactivity records per protein for predicted natural products in the phosphodiesterase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	1034	1075138	Tyrosyl-DNA phosphodiesterase 1	Homo sapiens	2	243
2	1	1034	2169736	Tyrosyl-DNA phosphodiesterase 2	Homo sapiens	2	101
3	1062	97	2652	Phosphodiesterase 2A	Homo sapiens	4	78
4	1063	98	241	Phosphodiesterase 3A	Homo sapiens	4	50
5	1064	100	254	Phosphodiesterase 4A	Homo sapiens	4	94
6	1064	101	275	Phosphodiesterase 4B	Homo sapiens	4	193
7	1064	103	288	Phosphodiesterase 4D	Homo sapiens	4	304
8	1065	104	1827	Phosphodiesterase 5A	Homo sapiens	4	449
9	1065	104	2304402	Phosphodiesterase 5A	Canis lupus familiaris	4	56
10	1065	104	3478	Phosphodiesterase 5A	Bos taurus	4	93
11	1067	111	3012	Phosphodiesterase 7A	Homo sapiens	4	75
12	1069	115	3535	Phosphodiesterase 9A	Homo sapiens	4	91
13	1077	123	4409	Phosphodiesterase 10A	Homo sapiens	4	369
14	1077	123	6140	cAMP and cAMP-inhibited cGMP 3',5'-cyclic phosphodiesterase 10A	Rattus norvegicus	4	87

Table S9. Number of bioactivity records per protein for predicted natural products in the isomerase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	644	1781	DNA topoisomerase I	Homo sapiens	2	246
2	1	644	1806	DNA topoisomerase II alpha	Homo sapiens	2	131
3	1	644	1858	DNA gyrase subunit A	Homo sapiens	2	112
4	1	644	1902	FK506-binding protein 1A	Homo sapiens	2	118
5	1	644	1949	Cyclophilin A	Homo sapiens	2	107
6	1	644	2288	Peptidyl-prolyl cis-trans isomerase NIMA-interacting 1	Homo sapiens	2	88
7	1	644	5422	Protein disulfide-isomerase	Homo sapiens	2	57
8	1	644	5658	Prostaglandin E synthase	Homo sapiens	2	257

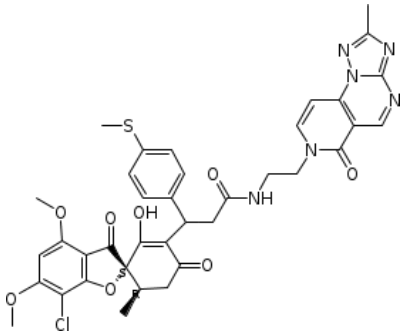
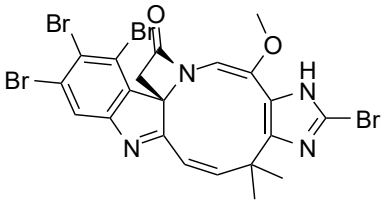
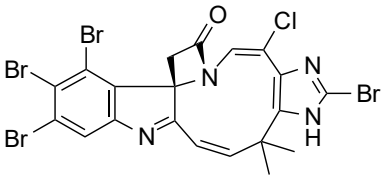
Table S10. Number of bioactivity records per protein for predicted natural products in the lyase group

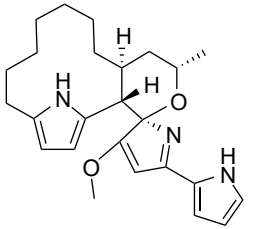
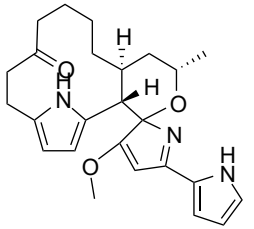
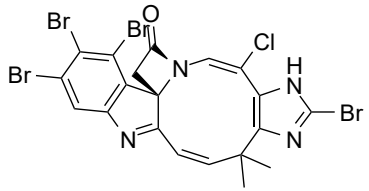
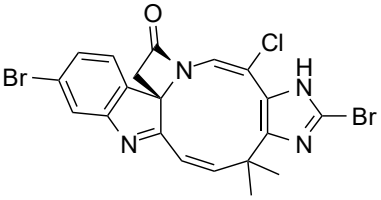
No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	648	1743183	Leukotriene C4 synthase	Homo sapiens	2	78
2	1	648	205	Carbonic anhydrase II	Homo sapiens	2	122
3	1	648	261	Carbonic anhydrase I	Homo sapiens	2	79
4	1	648	5359	Isocitrate lyase	Candida albicans	2	61

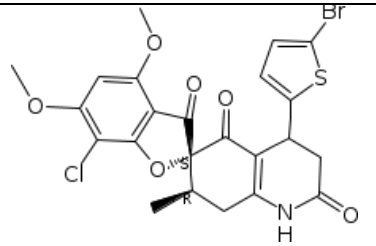
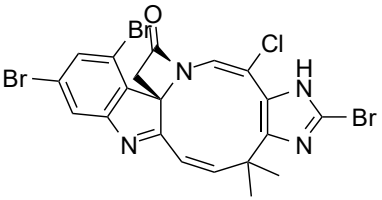
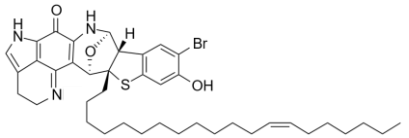
Table S11. Number of bioactivity records per protein for predicted natural products in the ligase group

No.	Target Parent ID	Protein Class ID	Target ChEMBL ID	Target Protein Name	Target Organism	Protein Class Level	Amount of Bioactivity Data
1	1	645	2397	Acetyl-CoA carboxylase 1	Rattus norvegicus	2	52
2	1	645	4662927	Histidine--tRNA ligase	Leishmania infantum	2	517
3	1	645	4829	Acetyl-CoA carboxylase 2	Homo sapiens	2	52

Table S12. Species origin and chemical structures of the ten most promising natural products with the lowest predicted IC_{50} values against AKT2 identified through virtual screening

No.	Compound ID	Database Source	Compound Name	Predicted IC_{50} (nM)	Species of Origin	Structure
1	STOCK1N-98616	InterBioScreen	–	4.92	–	
2	CMNPD3014	CMNPD	Methoxydechlorochartelline A	5.62	*Chartella papyracea*	
3	CMNPD2423	CMNPD	Chartelline A †	5.64	*Chartella papyracea*	

No.	Compound ID	Database Source	Compound Name	Predicted IC_{50} (nM)	Species of Origin	Structure
4	CMNPD24363	CMNPD	Premarineosin A	6.00	*Streptomyces venezuelae*	
5	CMNPD24364	CMNPD	16-ketopremarineosin A	6.30	*Streptomyces venezuelae*	
6	CMNPD3015	CMNPD	Chartelline A	6.32	*Chartella papyracea*	
7	CMNPD3013	CMNPD	Chartelline C	6.60	*Chartella papyracea*	

No.	Compound ID	Database Source	Compound Name	Predicted IC_{50} (nM)	Species of Origin	Structure
8	STOCK1N-95086	InterBioScreen	–	6.64	–	
9	CMNPD3012	CMNPD	Chartelline B	6.64	*Chartella papyracea*	
10	CMNPD23804	CMNPD	Atkamine A	6.69	*Latrunculia sp.*	

† An identifier to differentiate the compound from another with the same name but a distinct chemical structure.

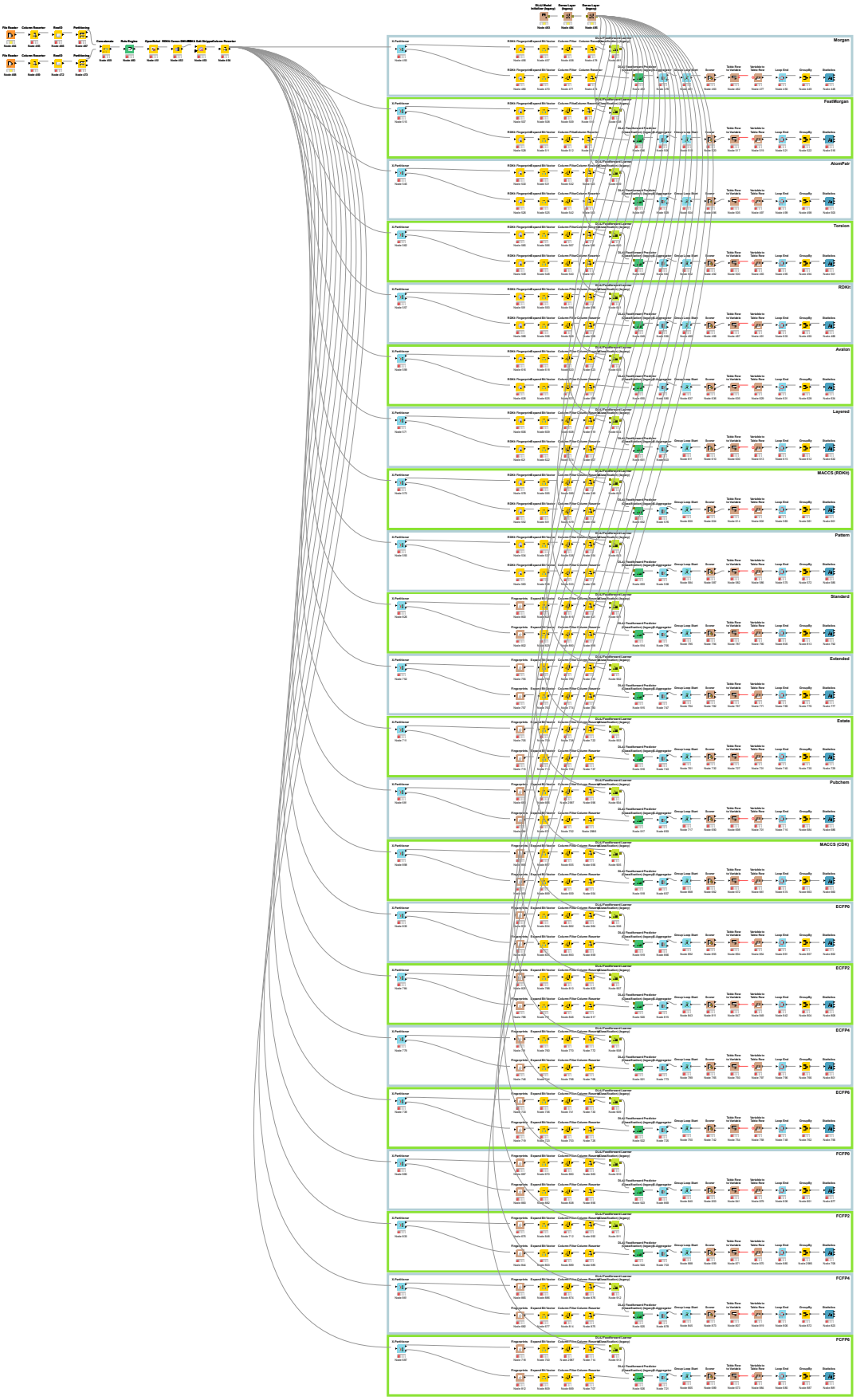


Figure S1. KNIME workflow for binary label classification studies using 21 molecular fingerprints and an artificial neural network algorithm.

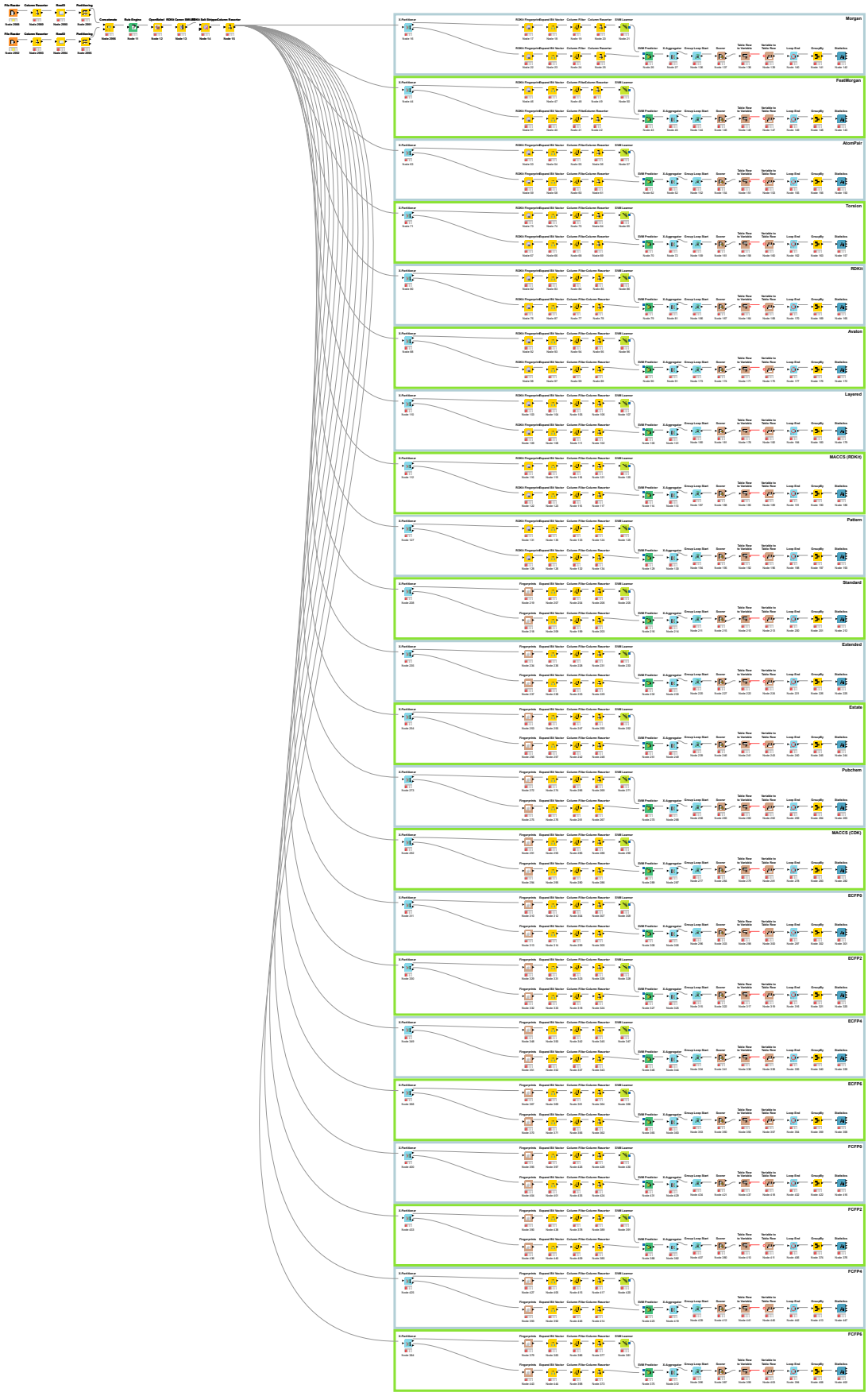


Figure S2. KNIME workflow for binary label classification studies using 21 molecular fingerprints and a support vector machine algorithm.

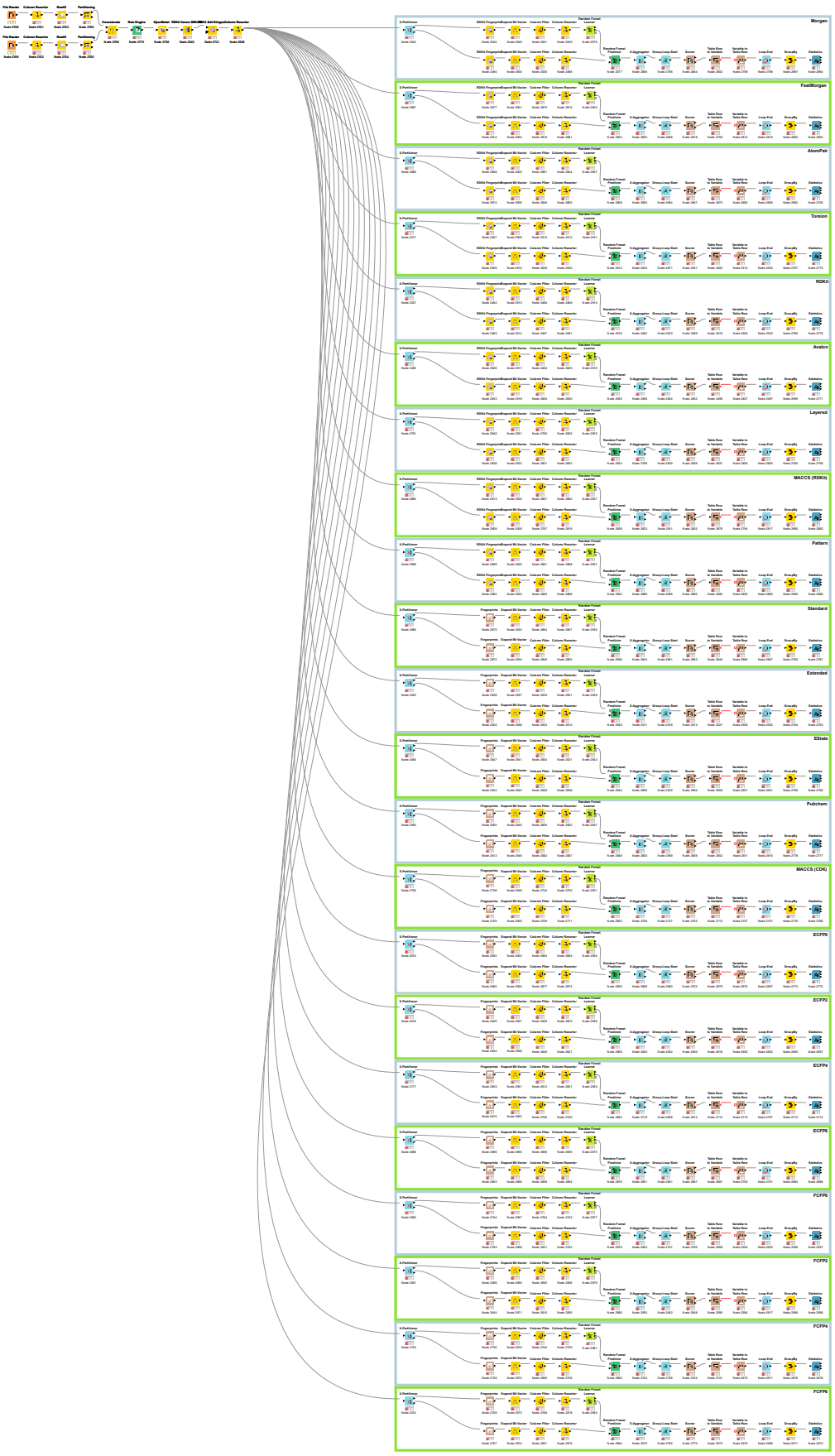


Figure S3. KNIME workflow for binary label classification studies using 21 molecular fingerprints and a random forest algorithm.

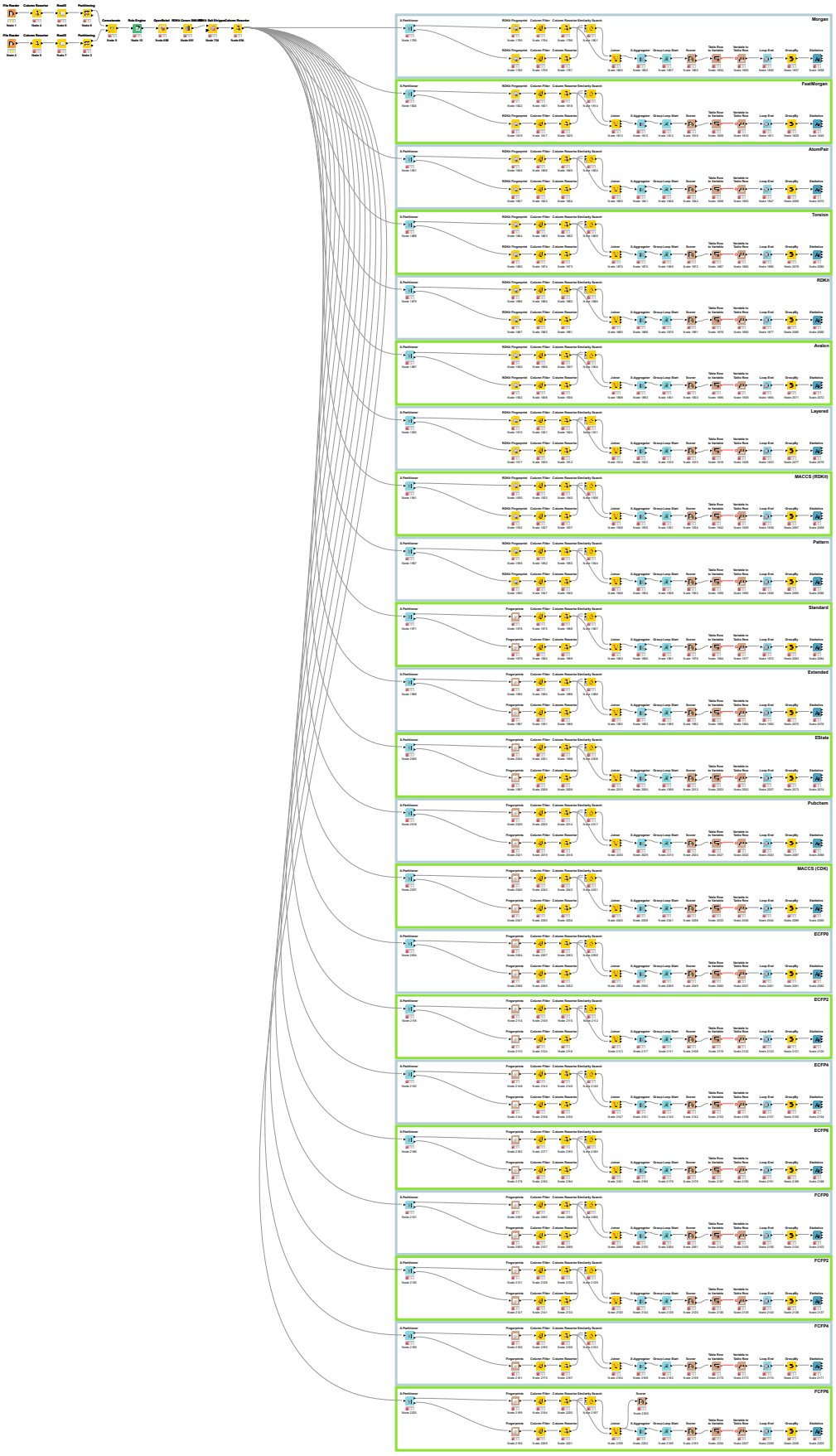


Figure S4. KNIME workflow for binary label classification studies using 21 molecular fingerprints and a 1-nearest neighbor algorithm.

Code S1. Construction of the best natural product filtering model using Avalon fingerprints and a random forest algorithm.

```
import pandas as pd
import re
from rdkit import Chem
from rdkit.Chem import SaltRemover
from rdkit.Avalon import pyAvalonTools
from sklearn.ensemble import RandomForestClassifier
import joblib

# Read the first file
file1_path = '/Users/Ramadhan/NP-
MTL/Binary_Classification/ibs2023mar_nc.smi'
file1_df = pd.read_csv(file1_path, header=None, delimiter='\t')

# Read the second file
file2_path = '/Users/Ramadhan/NP-
MTL/Binary_Classification/ibs2023mar_sc1.smi'
file2_df = pd.read_csv(file2_path, header=None, delimiter='\t')

# Randomly select 10000 rows from each file
random_state = 100
file1_sampled = file1_df.sample(n=10000, random_state=random_state)
file2_sampled = file2_df.sample(n=10000, random_state=random_state)

# Concatenate the sampled dataframes
concatenated_df = pd.concat([file1_sampled, file2_sampled])

# Function to convert SMILES to Canonical SMILES and remove salts, and
then convert to Avalon fingerprints
def convert_to_canonical_smiles_and_fingerprint(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        # Convert to canonical SMILES
        canonical_smiles = Chem.MolToSmiles(mol, isomericSmiles=False)
        # Remove salts
        remover = SaltRemover.SaltRemover()
        mol_without_salt = remover.StripMol(mol)
        if mol_without_salt is not None:
            canonical_smiles = Chem.MolToSmiles(mol_without_salt,
isomericSmiles=False)
            # Generate Avalon fingerprints with 1024 bits
            fingerprint = pyAvalonTools.GetAvalonFP(mol_without_salt,
nBits=1024)
            return canonical_smiles, fingerprint
        else:
            return canonical_smiles, None  # Return original SMILES and
None if salt stripping fails
    else:
        return None, None

# Apply the conversion function to the first column of the concatenated
dataframe
concatenated_df[[0, 2]] = concatenated_df[0].apply(lambda x:
pd.Series(convert_to_canonical_smiles_and_fingerprint(x)))
```

```
# Add third column based on conditions
pattern_nc = re.compile(r'STOCK.N.')
pattern_sc = re.compile(r'STOCK.S.')

concatenated_df[3] = concatenated_df[1].apply(lambda x: 'NC' if
pattern_nc.match(x) else 'SC')

# Initialize parameters for the Random Forest Classifier
n_estimators = 100

# Initialize a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=n_estimators,
random_state=random_state)

# Train the Random Forest Classifier on all data
X_train = list(concatenated_df[2])
y_train = concatenated_df[3]
rf_classifier.fit(X_train, y_train)

# Save the trained model
model_path = '/Users/Ramadhan/NP-
MTL/Binary_Classification/best_natural_product_filter.joblib'
joblib.dump(rf_classifier, model_path)
```

Code S2. Data preprocessing and natural product filtering for bioactivity records, demonstrated using the kinase group

```
import pandas as pd
import numpy as np
import joblib
from rdkit import Chem
from rdkit.Chem import SaltRemover, MolFromSmiles, MolToSmiles
from rdkit.Avalon import pyAvalonTools

# Load the data from the input file
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_datasets.tsv'
data = pd.read_csv(file_path, sep='\t', encoding='windows-1252',
low_memory=False)

# Split the rows based on "Standard Units"
nM_df = data[data['Standard Units'] == 'nM']
not_nM_df = data[data['Standard Units'] != 'nM']

# For the non-nM units, further split based on 'ug.mL-1'
ug_ml_df = not_nM_df[not_nM_df['Standard Units'] == 'ug.mL-1'].copy()
not_ug_ml_df = not_nM_df[not_nM_df['Standard Units'] != 'ug.mL-1'].copy()

# Convert 'Standard Value' for 'ug.mL-1' to 'nM'
ug_ml_df.loc[:, 'Standard Value'] = (ug_ml_df['Standard Value'] /
ug_ml_df['Molecular Weight']) * 1e6
ug_ml_df.loc[:, 'Standard Units'] = 'nM'

# Concatenate the modified DataFrames
```

```
data = pd.concat([nM_df, ug_ml_df])

# Remove rows with 'Standard Value' = 0
data = data[data['Standard Value'] != 0]

# Get unique combinations of the identifiers
unique_combinations = data[['Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID']].drop_duplicates()

# Initialize the salt remover
remover = SaltRemover.SaltRemover()

# Load the best natural product filter model
model_path = '/Users/Ramadhan/NP-
MTL/Binary_Classification/best_natural_product_filter.joblib'
rf_model = joblib.load(model_path)

# Function to process SMILES and calculate Avalon fingerprint
def process_smiles(smiles):
    try:
        mol = MolFromSmiles(smiles)
        if mol:
            mol = remover.StripMol(mol)
            cleaned_smiles = MolToSmiles(mol)
            fp = pyAvalonTools.GetAvalonFP(mol, nBits=1024)
            return [cleaned_smiles, list(fp)]
    except Exception as e:
        print(f"Error processing SMILES {smiles}: {e}")
    return [None, None]

# List to store each DataFrame
all_data_frames = []

# Iterate over each unique combination of IDs
for index, combo in unique_combinations.iterrows():
    filtered_data = data[
        (data['Target Parent ID'] == combo['Target Parent ID']) &
        (data['Protein Class ID'] == combo['Protein Class ID']) &
        (data['Target ChEMBL ID'] == combo['Target ChEMBL ID']) &
        (data['Standard Type'] == 'IC50')
    ]

    # Drop rows with missing values in 'SMILES', 'Standard Value', and
'Standard Relation'
    filtered_data = filtered_data.dropna(subset=['SMILES', 'Standard
Value', 'Standard Relation'])

    # Handle missing values in 'Document Year' by filling them with a
very low value (e.g., 0)
    filtered_data['Document Year'] = filtered_data['Document
Year'].fillna(0)

    # Sort the DataFrame first by 'Molecule ChEMBL ID', then by 'Document
Year' in descending order
    # and 'Standard Value' in ascending order to prepare for
deduplication
```

```
    filtered_data.sort_values(by=['Molecule ChEMBL ID', 'Document Year',
'Standard Value'], ascending=[True, False, True], inplace=True)

    # Remove duplicates: keeping the first entry for each 'Molecule
ChEMBL ID' which will be
    # the one with the maximum 'Document Year' and if tied, the minimum
'Standard Value'
    filtered_data = filtered_data.drop_duplicates(subset='Molecule ChEMBL
ID', keep='first')

    # Apply the process_smiles function and join results
    results = filtered_data['SMILES'].apply(lambda x: process_smiles(x))
    results_df = pd.DataFrame(results.tolist(), index=results.index,
columns=['Cleaned_SMILES', 'Avalon_FP'])
    filtered_data = filtered_data.join(results_df)

    # Prepare the Avalon fingerprints for prediction
    valid_fps = filtered_data['Avalon_FP'].dropna().tolist()
    valid_fps = np.array([np.array(fp) for fp in valid_fps if fp is not
None])

    # Predict using the best natural product filter model
    if len(valid_fps) > 0:
        predictions = rf_model.predict(valid_fps)
        filtered_data.loc[filtered_data['Avalon_FP'].notnull(),
'Predictions'] = predictions

    # Append to the list
    all_data_frames.append(filtered_data)

# Concatenate all DataFrames
final_df = pd.concat(all_data_frames)

# Filter for rows with prediction 'NC'
final_df = final_df[final_df['Predictions'] == 'NC']

# Filter for unique combinations with at least 50 entries
final_filtered_df = final_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).filter(lambda x: len(x) >= 50)

# Extract the numeric part of "Molecule ChEMBL ID" and convert it to an
integer for final sorting
final_filtered_df['Molecule ChEMBL ID Numeric'] =
final_filtered_df['Molecule ChEMBL ID'].str.extract('(\d+)').astype(int)

# Sort by 'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID', and
the numeric part of 'Molecule ChEMBL ID'
final_filtered_df.sort_values(by=['Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID', 'Molecule ChEMBL ID Numeric'], inplace=True)

# Drop the temporary numeric ID column
final_filtered_df.drop(columns=['Molecule ChEMBL ID Numeric'],
inplace=True)

# Filter rows with 'Standard Relation' of '='
input_table = final_filtered_df[final_filtered_df['Standard Relation'] ==
'=']
```

```
# Filter for unique combinations with at least 50 entries
filtered_table = input_table.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).filter(lambda x: len(x) >= 50)

# Convert 'Standard Value' to 'pIC50' and add it as a new column
filtered_table['pIC50'] = -np.log10(filtered_table['Standard Value'] *
10**-9)

# Remove the prefix 'CHEMBL' in 'Target ChEMBL ID'
filtered_table['Target ChEMBL ID'] = filtered_table['Target ChEMBL
ID'].str.replace('CHEMBL', '', regex=False)

# Save the final DataFrame to a new text file with tab separation
output_file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_for_stl_mtl.txt'
filtered_table.to_csv(output_file_path, index=False, sep='\t')
```

Code S3. Incorporation of protein amino acid global sequence similarity into the data frame, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0
```

```
# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'global'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_for_stl_mtl.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique UniProt Accession IDs
uniprot_ids = df['UniProt Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrix = {}
max_size = 0

# Process each group
valid_ids = [uid for uid in uniprot_ids if fetch_protein_sequence(uid) is
not None]
for id1 in valid_ids:
    seq1 = fetch_protein_sequence(id1)
    similarities = {}
    for id2 in valid_ids:
        seq2 = fetch_protein_sequence(id2)
        if seq1 and seq2:
            alignments = aligner.align(seq1, seq2)
            similarity = calculate_percentage_similarity(alignments[0],
seq1, seq2)
        else:
            similarity = None
        similarities[id2] = similarity
    similarity_matrix[id1] = similarities

# Convert similarity_matrix to DataFrame
df_sim = pd.DataFrame(similarity_matrix)
max_size = df_sim.shape[1]

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
```

```
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for idx, row in df.iterrows():
    accession = row['UniProt Accession']
    if accession in df_sim.columns:
        sim_values = df_sim.loc[accession].tolist()
        average_similarity =
calculate_average_similarity_above_diagonal(df_sim)
        df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
        df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_gss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S4. Incorporation of protein amino acid local sequence similarity into the data frame, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
```

```
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'local'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_for_stl_mtl.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique UniProt Accession IDs
uniprot_ids = df['UniProt Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrix = {}
max_size = 0

# Process each group
valid_ids = [uid for uid in uniprot_ids if fetch_protein_sequence(uid) is
not None]
for id1 in valid_ids:
    seq1 = fetch_protein_sequence(id1)
    similarities = {}
    for id2 in valid_ids:
        seq2 = fetch_protein_sequence(id2)
        if seq1 and seq2:
            alignments = aligner.align(seq1, seq2)
            similarity = calculate_percentage_similarity(alignments[0],
seq1, seq2)
        else:
            similarity = None
        similarities[id2] = similarity
    similarity_matrix[id1] = similarities

# Convert similarity_matrix to DataFrame
df_sim = pd.DataFrame(similarity_matrix)
```

```
max_size = df_sim.shape[1]

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for idx, row in df.iterrows():
    accession = row['UniProt Accession']
    if accession in df_sim.columns:
        sim_values = df_sim.loc[accession].tolist()
        average_similarity =
calculate_average_similarity_above_diagonal(df_sim)
        df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
        df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_lss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S5. Single-task learning, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan
```

```
# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID, Protein Class ID, and Target ChEMBL ID
groups = data.groupby(['Target Parent ID', 'Protein Class ID', 'Target
ChEMBL ID'])

# Process nested CV for each group separately using pre-assigned global
folds
for (target_parent_id, protein_class_id, target_chembl_id), group in
groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Avalon_FP'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
```

```
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, protein_class_id,
target_chembl_id, fold)] = best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
```

```
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))
    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the prediction data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_stl_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'STL Average RMSE', 'STL RMSE SD'
]

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_stl_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S6. Feature-based multi-task learning, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Process 'Avalon_FP' values
```

```
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before 'Avalon_FP'
data['Features'] = data.apply(lambda row: np.concatenate([[row['Target
ChEMBL ID']], row['Avalon_FP']]), axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Dictionary to store best hyperparameters per outer fold
best_params_dict = {}

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values

# Process nested CV
outer_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
fold_num = 0

for outer_train_index, outer_test_index in outer_skf.split(X,
data['Target ChEMBL ID']):
    X_outer_train, y_outer_train = X[outer_train_index],
y[outer_train_index]
    X_outer_test, y_outer_test = X[outer_test_index], y[outer_test_index]

    # Set up inner CV on the outer training set (using the same
stratification)
    inner_skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
    stratify_labels = data.loc[outer_train_index, 'Target ChEMBL ID']

    # Define the objective function for inner hyperparameter tuning
(using RMSE)
    def objective(trial):
        n_estimators = trial.suggest_int("n_estimators", 50, 200)
        max_depth = trial.suggest_int("max_depth", 3, 25)

        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=100
        )

        inner_rmse_scores = []
        for inner_train_index, inner_val_index in
inner_skf.split(X_outer_train, stratify_labels):
            X_inner_train = X_outer_train[inner_train_index]
            y_inner_train = y_outer_train[inner_train_index]
            X_inner_val = X_outer_train[inner_val_index]
            y_inner_val = y_outer_train[inner_val_index]

            model.fit(X_inner_train, y_inner_train)
            y_inner_pred = model.predict(X_inner_val)
```

```
            inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
            inner_rmse_scores.append(inner_rmse)

        return np.mean(inner_rmse_scores)

    # Optimize hyperparameters using Optuna on the inner CV
    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=20)
    best_params = study.best_params
    best_params_dict[f"fold_{fold_num}"] = best_params

    # Train the final model on the entire outer training set using the
best inner parameters
    final_model = RandomForestRegressor(
        n_estimators=best_params["n_estimators"],
        max_depth=best_params["max_depth"],
        random_state=100
    )
    final_model.fit(X_outer_train, y_outer_train)

    # Predict on the outer test set
    y_outer_pred = final_model.predict(X_outer_test)

    # Save predictions and fold number in the original DataFrame
    data.loc[outer_test_index, 'Prediction'] = y_outer_pred
    data.loc[outer_test_index, 'Fold'] = fold_num

    fold_num += 1

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
```

```
    })

# Sort the prediction data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_fbmtl_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'FBMTL Average RMSE', 'FBMTL RMSE SD'
]

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_fbmtl_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S7. Instance-based multi-task learning with protein amino acid global sequence similarity, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_gss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
```

```
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Dictionary to store best hyperparameters per outer fold
best_params_dict = {}

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values

# Process nested CV
outer_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
fold_num = 0

for outer_train_index, outer_test_index in outer_skf.split(X,
data['Target ChEMBL ID']):
    X_outer_train, X_outer_test = X[outer_train_index],
X[outer_test_index]
    y_outer_train, y_outer_test = y[outer_train_index],
y[outer_test_index]

    # Set up inner CV on the outer training set (using the same
stratification)
    inner_skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
    stratify_labels = data.loc[outer_train_index, 'Target ChEMBL ID']

    # Define the objective function for inner hyperparameter tuning
(using only RMSE)
    def objective(trial):
        n_estimators = trial.suggest_int("n_estimators", 50, 200)
        max_depth = trial.suggest_int("max_depth", 3, 25)

        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=100
        )

        inner_rmse_scores = []
        for inner_train_index, inner_val_index in
inner_skf.split(X_outer_train, stratify_labels):
            X_inner_train = X_outer_train[inner_train_index]
            y_inner_train = y_outer_train[inner_train_index]
            X_inner_val = X_outer_train[inner_val_index]
            y_inner_val = y_outer_train[inner_val_index]

            model.fit(X_inner_train, y_inner_train)
            y_inner_pred = model.predict(X_inner_val)
            inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
            inner_rmse_scores.append(inner_rmse)
```

```
        return np.mean(inner_rmse_scores)

    # Optimize hyperparameters using Optuna on the inner CV
    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=20)
    best_params = study.best_params
    best_params_dict[f"fold_{fold_num}"] = best_params

    # Train the final model on the entire outer training set with the
best inner parameters
    final_model = RandomForestRegressor(
        n_estimators=best_params["n_estimators"],
        max_depth=best_params["max_depth"],
        random_state=100
    )
    final_model.fit(X_outer_train, y_outer_train)

    # Predict on the outer test set
    y_outer_pred = final_model.predict(X_outer_test)

    # Save predictions and fold number in the original DataFrame
    data.loc[outer_test_index, 'Prediction'] = y_outer_pred
    data.loc[outer_test_index, 'Fold'] = fold_num

    fold_num += 1

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
```

```
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_ibmtl_gss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-GSS Average RMSE', 'IBMTL AA-GSS RMSE SD'
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_ibmtl_gss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S8. Instance-based multi-task learning with protein amino acid local sequence similarity, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_lss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
```

```
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Dictionary to store best hyperparameters per outer fold
best_params_dict = {}

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values

# Process nested CV
outer_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
fold_num = 0

for outer_train_index, outer_test_index in outer_skf.split(X,
data['Target ChEMBL ID']):
    X_outer_train, X_outer_test = X[outer_train_index],
X[outer_test_index]
    y_outer_train, y_outer_test = y[outer_train_index],
y[outer_test_index]

    # Set up inner CV on the outer training set (using the same
stratification)
    inner_skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
    stratify_labels = data.loc[outer_train_index, 'Target ChEMBL ID']

    # Define the objective function for inner hyperparameter tuning
(using only RMSE)
    def objective(trial):
        n_estimators = trial.suggest_int("n_estimators", 50, 200)
        max_depth = trial.suggest_int("max_depth", 3, 25)

        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=100
        )

        inner_rmse_scores = []
        for inner_train_index, inner_val_index in
inner_skf.split(X_outer_train, stratify_labels):
            X_inner_train = X_outer_train[inner_train_index]
            y_inner_train = y_outer_train[inner_train_index]
            X_inner_val = X_outer_train[inner_val_index]
            y_inner_val = y_outer_train[inner_val_index]

            model.fit(X_inner_train, y_inner_train)
            y_inner_pred = model.predict(X_inner_val)
            inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
```

```
            inner_rmse_scores.append(inner_rmse)

        return np.mean(inner_rmse_scores)

    # Optimize hyperparameters using Optuna on the inner CV
    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=20)
    best_params = study.best_params
    best_params_dict[f"fold_{fold_num}"] = best_params

    # Train the final model on the entire outer training set with the
best inner parameters
    final_model = RandomForestRegressor(
        n_estimators=best_params["n_estimators"],
        max_depth=best_params["max_depth"],
        random_state=100
    )
    final_model.fit(X_outer_train, y_outer_train)

    # Predict on the outer test set
    y_outer_pred = final_model.predict(X_outer_test)

    # Save predictions and fold number in the original DataFrame
    data.loc[outer_test_index, 'Prediction'] = y_outer_pred
    data.loc[outer_test_index, 'Fold'] = fold_num

    fold_num += 1

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })
```

```
# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_ibmtl_lss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-LSS Average RMSE', 'IBMTL AA-LSS RMSE SD'
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Studies/kinase_rf_ibmtl_lss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S9. Data filtering for three hierarchical protein group levels studies, demonstrated using the kinase group

```
import pandas as pd
import numpy as np

# Read the data from the file
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_final_dataset_for_stl_mtl.txt'
input_table = pd.read_csv(file_path, sep='\t')

# Ensure each Protein Class ID has at least 2 different Target ChEMBL IDs
filtered_table = input_table.groupby('Protein Class ID').filter(lambda x:
x['Target ChEMBL ID'].nunique() >= 2)

# Save the updated dataframe to a new file
output_file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
filtered_table.to_csv(output_file_path, sep='\t', index=False)
```

Code S10. Single-task learning for three hierarchical protein group levels studies, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID, Protein Class ID, and Target ChEMBL ID
groups = data.groupby(['Target Parent ID', 'Protein Class ID', 'Target
ChEMBL ID'])

# Process nested CV for each group separately using pre-assigned global
folds
for (target_parent_id, protein_class_id, target_chembl_id), group in
groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Avalon_FP'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]
```

```
        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, protein_class_id,
target_chembl_id, fold)] = best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
```

```
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))
    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the prediction data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
predictions_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_rf_stl_prediction.tx
t'
data.to_csv(predictions_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
```

```
    'STL Average RMSE', 'STL RMSE SD'
]

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_rf_stl_statistics.tx
t'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S11. Feature-based multi-task learning for the superclass level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before 'Avalon_FP'
data['Features'] = data.apply(lambda row: np.concatenate([[row['Target
ChEMBL ID']], row['Avalon_FP']]), axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Dictionary to store best hyperparameters per outer fold
best_params_dict = {}

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values

# Process nested CV
outer_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
fold_num = 0

for outer_train_index, outer_test_index in outer_skf.split(X,
data['Target ChEMBL ID']):
    X_outer_train, y_outer_train = X[outer_train_index],
y[outer_train_index]
    X_outer_test, y_outer_test = X[outer_test_index], y[outer_test_index]
```

```
    # Set up inner CV on the outer training set (using the same
stratification)
    inner_skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
    stratify_labels = data.loc[outer_train_index, 'Target ChEMBL ID']

    # Define the objective function for inner hyperparameter tuning
(using RMSE)
    def objective(trial):
        n_estimators = trial.suggest_int("n_estimators", 50, 200)
        max_depth = trial.suggest_int("max_depth", 3, 25)

        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=100
        )

        inner_rmse_scores = []
        for inner_train_index, inner_val_index in
inner_skf.split(X_outer_train, stratify_labels):
            X_inner_train = X_outer_train[inner_train_index]
            y_inner_train = y_outer_train[inner_train_index]
            X_inner_val = X_outer_train[inner_val_index]
            y_inner_val = y_outer_train[inner_val_index]

            model.fit(X_inner_train, y_inner_train)
            y_inner_pred = model.predict(X_inner_val)
            inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
            inner_rmse_scores.append(inner_rmse)

        return np.mean(inner_rmse_scores)

    # Optimize hyperparameters using Optuna on the inner CV
    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=20)
    best_params = study.best_params
    best_params_dict[f"fold_{fold_num}"] = best_params

    # Train the final model on the entire outer training set using the
best inner parameters
    final_model = RandomForestRegressor(
        n_estimators=best_params["n_estimators"],
        max_depth=best_params["max_depth"],
        random_state=100
    )
    final_model.fit(X_outer_train, y_outer_train)

    # Predict on the outer test set
    y_outer_pred = final_model.predict(X_outer_test)

    # Save predictions and fold number in the original DataFrame
    data.loc[outer_test_index, 'Prediction'] = y_outer_pred
    data.loc[outer_test_index, 'Fold'] = fold_num

    fold_num += 1
```

```
# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the prediction data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_superclass_rf_fbmtl_
prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'FBMTL Average RMSE', 'FBMTL RMSE SD'
]

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_superclass_rf_fbmtl_
statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S12. Incorporation of protein amino acid global sequence similarity into the data frame for the superclass level, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'global'

# Load data
```

```
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique UniProt Accession IDs
uniprot_ids = df['UniProt Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrix = {}
max_size = 0

# Process each group
valid_ids = [uid for uid in uniprot_ids if fetch_protein_sequence(uid) is
not None]
for id1 in valid_ids:
    seq1 = fetch_protein_sequence(id1)
    similarities = {}
    for id2 in valid_ids:
        seq2 = fetch_protein_sequence(id2)
        if seq1 and seq2:
            alignments = aligner.align(seq1, seq2)
            similarity = calculate_percentage_similarity(alignments[0],
seq1, seq2)
        else:
            similarity = None
        similarities[id2] = similarity
    similarity_matrix[id1] = similarities

# Convert similarity_matrix to DataFrame
df_sim = pd.DataFrame(similarity_matrix)
max_size = df_sim.shape[1]

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for idx, row in df.iterrows():
    accession = row['UniProt Accession']
    if accession in df_sim.columns:
        sim_values = df_sim.loc[accession].tolist()
        average_similarity =
calculate_average_similarity_above_diagonal(df_sim)
        df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
```

```
        df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_superclass_final_dataset_
gss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S13. Instance-based multi-task learning with protein amino acid global sequence similarity for the superclass level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_superclass_final_dataset_
gss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Dictionary to store best hyperparameters per outer fold
best_params_dict = {}

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values

# Process nested CV
outer_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
fold_num = 0
```

```
for outer_train_index, outer_test_index in outer_skf.split(X,
data['Target ChEMBL ID']):
    X_outer_train, X_outer_test = X[outer_train_index],
X[outer_test_index]
    y_outer_train, y_outer_test = y[outer_train_index],
y[outer_test_index]

    # Set up inner CV on the outer training set (using the same
stratification)
    inner_skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
    stratify_labels = data.loc[outer_train_index, 'Target ChEMBL ID']

    # Define the objective function for inner hyperparameter tuning
(using only RMSE)
    def objective(trial):
        n_estimators = trial.suggest_int("n_estimators", 50, 200)
        max_depth = trial.suggest_int("max_depth", 3, 25)

        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=100
        )

        inner_rmse_scores = []
        for inner_train_index, inner_val_index in
inner_skf.split(X_outer_train, stratify_labels):
            X_inner_train = X_outer_train[inner_train_index]
            y_inner_train = y_outer_train[inner_train_index]
            X_inner_val = X_outer_train[inner_val_index]
            y_inner_val = y_outer_train[inner_val_index]

            model.fit(X_inner_train, y_inner_train)
            y_inner_pred = model.predict(X_inner_val)
            inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
            inner_rmse_scores.append(inner_rmse)

        return np.mean(inner_rmse_scores)

    # Optimize hyperparameters using Optuna on the inner CV
    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=20)
    best_params = study.best_params
    best_params_dict[f"fold_{fold_num}"] = best_params

    # Train the final model on the entire outer training set using the
best inner parameters
    final_model = RandomForestRegressor(
        n_estimators=best_params["n_estimators"],
        max_depth=best_params["max_depth"],
        random_state=100
    )
    final_model.fit(X_outer_train, y_outer_train)

    # Predict on the outer test set
```

```
    y_outer_pred = final_model.predict(X_outer_test)

    # Save predictions and fold number in the original DataFrame
    data.loc[outer_test_index, 'Prediction'] = y_outer_pred
    data.loc[outer_test_index, 'Fold'] = fold_num

    fold_num += 1

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_superclass_rf_ibmtl_
gss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-GSS Average RMSE', 'IBMTL AA-GSS RMSE SD'
```

```
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_superclass_rf_ibmtl_
gss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S14. Incorporation of protein amino acid local sequence similarity into the data frame for the superclass level, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
```

```
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'local'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique UniProt Accession IDs
uniprot_ids = df['UniProt Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrix = {}
max_size = 0

# Process each group
valid_ids = [uid for uid in uniprot_ids if fetch_protein_sequence(uid) is
not None]
for id1 in valid_ids:
    seq1 = fetch_protein_sequence(id1)
    similarities = {}
    for id2 in valid_ids:
        seq2 = fetch_protein_sequence(id2)
        if seq1 and seq2:
            alignments = aligner.align(seq1, seq2)
            similarity = calculate_percentage_similarity(alignments[0],
seq1, seq2)
        else:
            similarity = None
        similarities[id2] = similarity
    similarity_matrix[id1] = similarities

# Convert similarity_matrix to DataFrame
df_sim = pd.DataFrame(similarity_matrix)
max_size = df_sim.shape[1]

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
```

```
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for idx, row in df.iterrows():
    accession = row['UniProt Accession']
    if accession in df_sim.columns:
        sim_values = df_sim.loc[accession].tolist()
        average_similarity =
calculate_average_similarity_above_diagonal(df_sim)
        df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
        df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_superclass_final_dataset_
lss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S15. Instance-based multi-task learning with protein amino acid local sequence similarity for the superclass level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_superclass_final_dataset_
lss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
```

```
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Dictionary to store best hyperparameters per outer fold
best_params_dict = {}

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values

# Process nested CV
outer_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
fold_num = 0

for outer_train_index, outer_test_index in outer_skf.split(X,
data['Target ChEMBL ID']):
    X_outer_train, X_outer_test = X[outer_train_index],
X[outer_test_index]
    y_outer_train, y_outer_test = y[outer_train_index],
y[outer_test_index]

    # Set up inner CV on the outer training set (using the same
stratification)
    inner_skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
    stratify_labels = data.loc[outer_train_index, 'Target ChEMBL ID']

    # Define the objective function for inner hyperparameter tuning
(using only RMSE)
    def objective(trial):
        n_estimators = trial.suggest_int("n_estimators", 50, 200)
        max_depth = trial.suggest_int("max_depth", 3, 25)

        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=100
        )

        inner_rmse_scores = []
        for inner_train_index, inner_val_index in
inner_skf.split(X_outer_train, stratify_labels):
            X_inner_train = X_outer_train[inner_train_index]
            y_inner_train = y_outer_train[inner_train_index]
            X_inner_val = X_outer_train[inner_val_index]
            y_inner_val = y_outer_train[inner_val_index]

            model.fit(X_inner_train, y_inner_train)
            y_inner_pred = model.predict(X_inner_val)
            inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
            inner_rmse_scores.append(inner_rmse)

        return np.mean(inner_rmse_scores)

    # Optimize hyperparameters using Optuna on the inner CV
    study = optuna.create_study(direction="minimize")
```

```
    study.optimize(objective, n_trials=20)
    best_params = study.best_params
    best_params_dict[f"fold_{fold_num}"] = best_params

    # Train the final model on the entire outer training set using the
best inner parameters
    final_model = RandomForestRegressor(
        n_estimators=best_params["n_estimators"],
        max_depth=best_params["max_depth"],
        random_state=100
    )
    final_model.fit(X_outer_train, y_outer_train)

    # Predict on the outer test set
    y_outer_pred = final_model.predict(X_outer_test)

    # Save predictions and fold number in the original DataFrame
    data.loc[outer_test_index, 'Prediction'] = y_outer_pred
    data.loc[outer_test_index, 'Fold'] = fold_num

    fold_num += 1

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
```

```
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_superclass_rf_ibmtl_
lss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-LSS Average RMSE', 'IBMTL AA-LSS RMSE SD'
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_superclass_rf_ibmtl_
lss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S16. Feature-based multi-task learning for the target parent level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before 'Avalon_FP'
data['Features'] = data.apply(lambda row: np.concatenate([[row['Target
ChEMBL ID']], row['Avalon_FP']]), axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
```

```
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID (excluding Protein Class ID)
groups = data.groupby(['Target Parent ID'])

# Process nested CV for each group separately using the pre-assigned
global folds
for target_parent_id, group in groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Features'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        # Use 'Target ChEMBL ID' as stratification labels from the outer
training set
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
```

```
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, fold)] = best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
```

```
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the prediction data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_target_parent_rf_fbm
tl_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'FBMTL Average RMSE', 'FBMTL RMSE SD'
]

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_target_parent_rf_fbm
tl_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S17. Incorporation of protein amino acid global sequence similarity into the data frame for the target parent level, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
```

```
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'global'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique combinations of 'Target Parent ID' and corresponding
'UniProt Accession'
grouped = df.groupby(['Target Parent ID'])['UniProt Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
```

```
similarity_matrices = {}
max_size = 0

# Process each group
for (target_parent_id), uniprot_ids in grouped.items():
    valid_ids = [uid for uid in uniprot_ids if
fetch_protein_sequence(uid) is not None]
    similarity_matrix = {}
    for id1 in valid_ids:
        seq1 = fetch_protein_sequence(id1)
        similarities = {}
        for id2 in valid_ids:
            seq2 = fetch_protein_sequence(id2)
            if seq1 and seq2:
                alignments = aligner.align(seq1, seq2)
                similarity =
calculate_percentage_similarity(alignments[0], seq1, seq2)
            else:
                similarity = None
            similarities[id2] = similarity
        similarity_matrix[id1] = similarities
    df_sim = pd.DataFrame(similarity_matrix)
    similarity_matrices[(target_parent_id)] = df_sim
    max_size = max(max_size, df_sim.shape[1])

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for (tpid), matrix in similarity_matrices.items():
    ids = matrix.columns
    for idx, row in df[(df['Target Parent ID'] == tpid)].iterrows():
        accession = row['UniProt Accession']
        if accession in ids:
            sim_values = matrix.loc[accession].tolist()
            average_similarity =
calculate_average_similarity_above_diagonal(matrix)
            df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
            df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_target_parent_final_datas
et_gss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S18. Instance-based multi-task learning with protein amino acid global sequence similarity for the target parent level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_target_parent_final_datas
et_gss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID only
groups = data.groupby(['Target Parent ID'])

# Process nested CV for each group separately using the pre-assigned
global folds
for target_parent_id, group in groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Features'].values)
    y_group = group['pIC50'].values
```

```
    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        # Use 'Target ChEMBL ID' as stratification labels from the outer
training set
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
```

```
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, fold)] = best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_target_parent_rf_ibm
tl_gss_prediction.txt'
```

```
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-GSS Average RMSE', 'IBMTL AA-GSS RMSE SD'
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_target_parent_rf_ibm
tl_gss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S19. Incorporation of protein amino acid local sequence similarity into the data frame for the target parent level, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
```

```
        for i, j in zip(range(startA, endA), range(startB, endB)):
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'local'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique combinations of 'Target Parent ID' and corresponding
'UniProt Accession'
grouped = df.groupby(['Target Parent ID'])['UniProt Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrices = {}
max_size = 0

# Process each group
for (target_parent_id), uniprot_ids in grouped.items():
    valid_ids = [uid for uid in uniprot_ids if
fetch_protein_sequence(uid) is not None]
    similarity_matrix = {}
    for id1 in valid_ids:
        seq1 = fetch_protein_sequence(id1)
        similarities = {}
        for id2 in valid_ids:
            seq2 = fetch_protein_sequence(id2)
            if seq1 and seq2:
                alignments = aligner.align(seq1, seq2)
                similarity =
calculate_percentage_similarity(alignments[0], seq1, seq2)
            else:
                similarity = None
```

```
            similarities[id2] = similarity
        similarity_matrix[id1] = similarities
    df_sim = pd.DataFrame(similarity_matrix)
    similarity_matrices[(target_parent_id)] = df_sim
    max_size = max(max_size, df_sim.shape[1])

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for (tpid), matrix in similarity_matrices.items():
    ids = matrix.columns
    for idx, row in df[(df['Target Parent ID'] == tpid)].iterrows():
        accession = row['UniProt Accession']
        if accession in ids:
            sim_values = matrix.loc[accession].tolist()
            average_similarity =
calculate_average_similarity_above_diagonal(matrix)
            df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
            df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_target_parent_final_datas
et_lss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S20. Instance-based multi-task learning with protein amino acid local sequence similarity for the target parent level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_target_parent_final_datas
et_lss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)
```

```
# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID only
groups = data.groupby(['Target Parent ID'])

# Process nested CV for each group separately using the pre-assigned
global folds
for target_parent_id, group in groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Features'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
```

```
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        # Use 'Target ChEMBL ID' as stratification labels from the outer
training set
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, fold)] = best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
```

```
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_target_parent_rf_ibm
tl_lss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-LSS Average RMSE', 'IBMTL AA-LSS RMSE SD'
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
```

```
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_target_parent_rf_ibm
tl_lss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S21. Feature-based multi-task learning for the protein class level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before 'Avalon_FP'
data['Features'] = data.apply(lambda row: np.concatenate([[row['Target
ChEMBL ID']], row['Avalon_FP']]), axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID and Protein Class ID (excluding Target ChEMBL
ID)
groups = data.groupby(['Target Parent ID', 'Protein Class ID'])

# Process nested CV for each group separately using the pre-assigned
global folds
for (target_parent_id, protein_class_id), group in groups:
```

```
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Features'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        # Use 'Target ChEMBL ID' as stratification labels from the outer
training set
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)
```

```
            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, protein_class_id, fold)] =
best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the prediction data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)
```

```
# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_protein_class_rf_fbm
tl_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'FBMTL Average RMSE', 'FBMTL RMSE SD'
]

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_protein_class_rf_fbm
tl_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S22. Incorporation of protein amino acid global sequence similarity into the data frame for the protein class level, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
```

```
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'global'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique combinations of 'Target Parent ID' and 'Protein Class
ID' and corresponding 'UniProt Accession'
grouped = df.groupby(['Target Parent ID', 'Protein Class ID'])['UniProt
Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrices = {}
max_size = 0

# Process each group
for (target_parent_id, protein_class_id), uniprot_ids in grouped.items():
    valid_ids = [uid for uid in uniprot_ids if
fetch_protein_sequence(uid) is not None]
    similarity_matrix = {}
    for id1 in valid_ids:
        seq1 = fetch_protein_sequence(id1)
        similarities = {}
        for id2 in valid_ids:
            seq2 = fetch_protein_sequence(id2)
            if seq1 and seq2:
                alignments = aligner.align(seq1, seq2)
                similarity =
calculate_percentage_similarity(alignments[0], seq1, seq2)
            else:
                similarity = None
```

```
            similarities[id2] = similarity
        similarity_matrix[id1] = similarities
    df_sim = pd.DataFrame(similarity_matrix)
    similarity_matrices[(target_parent_id, protein_class_id)] = df_sim
    max_size = max(max_size, df_sim.shape[1])

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)

# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for (tpid, pcid), matrix in similarity_matrices.items():
    ids = matrix.columns
    for idx, row in df[(df['Target Parent ID'] == tpid) & (df['Protein
Class ID'] == pcid)].iterrows():
        accession = row['UniProt Accession']
        if accession in ids:
            sim_values = matrix.loc[accession].tolist()
            average_similarity =
calculate_average_similarity_above_diagonal(matrix)
            df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
            df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_protein_class_final_datas
et_gss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S23. Instance-based multi-task learning with protein amino acid global sequence similarity for the protein class level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_protein_class_final_datas
et_gss_for_stl_mtl.txt'
```

```
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID and Protein Class ID (excluding Target ChEMBL
ID)
groups = data.groupby(['Target Parent ID', 'Protein Class ID'])

# Process nested CV for each group separately using the pre-assigned
global folds
for (target_parent_id, protein_class_id), group in groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Features'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
```

```
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        # Use 'Target ChEMBL ID' as stratification labels from the outer
training set
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']

        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, protein_class_id, fold)] =
best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
```

```
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_protein_class_rf_ibm
tl_gss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-GSS Average RMSE', 'IBMTL AA-GSS RMSE SD'
]
```

```
# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_protein_class_rf_ibm
tl_gss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S24. Incorporation of protein amino acid local sequence similarity into the data frame for the protein class level, demonstrated using the kinase group

```
import pandas as pd
from Bio import Align, SeqIO
from Bio.Align import substitution_matrices, PairwiseAligner
from io import StringIO
import requests

# Function to fetch protein sequence from UniProt
def fetch_protein_sequence(uniprot_id):
    url = f"https://www.uniprot.org/uniprot/{uniprot_id}.fasta"
    response = requests.get(url)
    if response.status_code == 200:
        fasta_data = response.text
        record = SeqIO.read(StringIO(fasta_data), "fasta")
        return str(record.seq)
    else:
        print(f"Failed to fetch sequence for {uniprot_id}")
        return None

# Function to calculate similarity between two sequences using
PairwiseAligner
def calculate_percentage_similarity(alignment, seq1, seq2):
    seqA, seqB = alignment.aligned
    alignment_length = 0
    similarity_count = 0

    for (startA, endA), (startB, endB) in zip(seqA, seqB):
        for i, j in zip(range(startA, endA), range(startB, endB)):
            if seq1[i] == seq2[j]:
                similarity_count += 1
            alignment_length += 1

    return (similarity_count / alignment_length) if alignment_length > 0
else 0

# Function to calculate average similarity above the diagonal
def calculate_average_similarity_above_diagonal(matrix):
    values = []
    for i in range(matrix.shape[0]):
        for j in range(i+1, matrix.shape[1]):
            values.append(matrix.iloc[i, j])
```

```
    return sum(values) / len(values) if values else None

# Load the BLOSUM62 matrix
matrix = substitution_matrices.load("BLOSUM62")

# Define the aligner and set gap penalties
aligner = PairwiseAligner()
aligner.substitution_matrix = matrix
aligner.open_gap_score = -10
aligner.extend_gap_score = -0.5
aligner.mode = 'local'

# Load data
data_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_final_dataset_for_stl_mtl
.txt'
df = pd.read_csv(data_path, sep='\t', low_memory=False)

# Extract unique combinations of 'Target Parent ID' and 'Protein Class
ID' and corresponding 'UniProt Accession'
grouped = df.groupby(['Target Parent ID', 'Protein Class ID'])['UniProt
Accession'].unique()

# Initialize empty dictionary to store similarity matrices and maximum
matrix size
similarity_matrices = {}
max_size = 0

# Process each group
for (target_parent_id, protein_class_id), uniprot_ids in grouped.items():
    valid_ids = [uid for uid in uniprot_ids if
fetch_protein_sequence(uid) is not None]
    similarity_matrix = {}
    for id1 in valid_ids:
        seq1 = fetch_protein_sequence(id1)
        similarities = {}
        for id2 in valid_ids:
            seq2 = fetch_protein_sequence(id2)
            if seq1 and seq2:
                alignments = aligner.align(seq1, seq2)
                similarity =
calculate_percentage_similarity(alignments[0], seq1, seq2)
            else:
                similarity = None
            similarities[id2] = similarity
        similarity_matrix[id1] = similarities
    df_sim = pd.DataFrame(similarity_matrix)
    similarity_matrices[(target_parent_id, protein_class_id)] = df_sim
    max_size = max(max_size, df_sim.shape[1])

# Find position to insert new columns before 'Avalon_FP'
avalon_fp_index = df.columns.get_loc('Avalon_FP')

# Insert new columns before 'Avalon_FP'
for i in range(max_size):
    df.insert(avalon_fp_index + i, f"Sim_{i+1}", None)
```

```
# Find position to insert 'Average Similarity' column before 'pIC50'
class_index = df.columns.get_loc('pIC50')

# Insert the 'Average Similarity' column before 'pIC50'
df.insert(class_index, 'Average Similarity', None)

# Populate the new columns with similarity values and calculate the
average similarity
for (tpid, pcid), matrix in similarity_matrices.items():
    ids = matrix.columns
    for idx, row in df[(df['Target Parent ID'] == tpid) & (df['Protein
Class ID'] == pcid)].iterrows():
        accession = row['UniProt Accession']
        if accession in ids:
            sim_values = matrix.loc[accession].tolist()
            average_similarity =
calculate_average_similarity_above_diagonal(matrix)
            df.loc[idx, [f"Sim_{i+1}" for i in range(len(sim_values))]] =
sim_values
            df.loc[idx, 'Average Similarity'] = average_similarity

# Save the updated DataFrame to the specified path
output_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_protein_class_final_datas
et_lss_for_stl_mtl.txt'
df.to_csv(output_path, sep='\t', index=False)
```

Code S25. Instance-based multi-task learning with protein amino acid local sequence similarity for the protein class level, demonstrated using the kinase group and a random forest algorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import optuna

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_protein_class_final_datas
et_lss_for_stl_mtl.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
```

```
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare columns for predictions and fold information
data['Prediction'] = np.nan
data['Fold'] = np.nan

# Assign global fold numbers using StratifiedKFold on the entire dataset
(stratified by 'Target ChEMBL ID')
global_skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=100)
data['GlobalFold'] = np.nan
for fold, (train_index, test_index) in enumerate(global_skf.split(data,
data['Target ChEMBL ID'])):
    data.loc[test_index, 'GlobalFold'] = fold

# Dictionary to store best hyperparameters per group and fold
best_params_dict = {}

# Group by Target Parent ID and Protein Class ID (excluding Target ChEMBL
ID)
groups = data.groupby(['Target Parent ID', 'Protein Class ID'])

# Process nested CV for each group separately using the pre-assigned
global folds
for (target_parent_id, protein_class_id), group in groups:
    # Prepare feature matrix X and target vector y for the group
    X_group = np.stack(group['Features'].values)
    y_group = group['pIC50'].values

    # Get the unique global fold values present in the group
    group_global_folds = group['GlobalFold'].unique()

    for fold in group_global_folds:
        # Identify outer test and train indices for the group based on
the pre-assigned GlobalFold
        outer_test_index = group.index[group['GlobalFold'] == fold]
        outer_train_index = group.index[group['GlobalFold'] != fold]

        # Get positions within the group (since X_group is a numpy array
following the group order)
        outer_test_positions = np.where(group['GlobalFold'] == fold)[0]
        outer_train_positions = np.where(group['GlobalFold'] != fold)[0]

        X_outer_train = X_group[outer_train_positions]
        y_outer_train = y_group[outer_train_positions]
        X_outer_test = X_group[outer_test_positions]
        y_outer_test = y_group[outer_test_positions]

        # For inner CV, use StratifiedKFold on the outer training set
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=100)
        # Use 'Target ChEMBL ID' as stratification labels from the outer
training set
        stratify_labels_inner = group.iloc[outer_train_positions]['Target
ChEMBL ID']
```

```
        # Define the objective function for inner hyperparameter tuning
(using RMSE)
        def objective(trial):
            n_estimators = trial.suggest_int("n_estimators", 50, 200)
            max_depth = trial.suggest_int("max_depth", 3, 25)

            model = RandomForestRegressor(
                n_estimators=n_estimators,
                max_depth=max_depth,
                random_state=100
            )

            inner_rmse_scores = []
            for inner_train_index, inner_val_index in
inner_cv.split(X_outer_train, stratify_labels_inner):
                X_inner_train = X_outer_train[inner_train_index]
                y_inner_train = y_outer_train[inner_train_index]
                X_inner_val = X_outer_train[inner_val_index]
                y_inner_val = y_outer_train[inner_val_index]

                model.fit(X_inner_train, y_inner_train)
                y_inner_pred = model.predict(X_inner_val)
                inner_rmse = np.sqrt(mean_squared_error(y_inner_val,
y_inner_pred))
                inner_rmse_scores.append(inner_rmse)

            return np.mean(inner_rmse_scores)

        # Optimize hyperparameters using Optuna on the inner CV
        study = optuna.create_study(direction="minimize")
        study.optimize(objective, n_trials=20)
        best_params = study.best_params
        best_params_dict[(target_parent_id, protein_class_id, fold)] =
best_params

        # Train the final model on the entire outer training set using
the best inner parameters
        final_model = RandomForestRegressor(
            n_estimators=best_params["n_estimators"],
            max_depth=best_params["max_depth"],
            random_state=100
        )
        final_model.fit(X_outer_train, y_outer_train)

        # Predict on the outer test set
        y_outer_pred = final_model.predict(X_outer_test)

        # Save predictions and fold number in the original DataFrame
        data.loc[outer_test_index, 'Prediction'] = y_outer_pred
        data.loc[outer_test_index, 'Fold'] = fold

# Calculate outer test performance (RMSE) for each unique group
results = []
group_columns = ['Target Parent ID', 'Protein Class ID', 'Target ChEMBL
ID', 'Fold']
for id_group, group_data in data.groupby(group_columns):
    true_values = group_data['pIC50']
```

```
    predicted_values = group_data['Prediction']

    # Skip groups with NaN or infinite values
    if np.any(np.isnan(true_values)) or
np.any(np.isnan(predicted_values)):
        print(f"NaN values found in {id_group}, skipping this group.")
        continue
    if np.any(np.isinf(true_values)) or
np.any(np.isinf(predicted_values)):
        print(f"Infinite values found in {id_group}, skipping this
group.")
        continue

    rmse = np.sqrt(mean_squared_error(true_values, predicted_values))

    results.append({
        'Target Parent ID': id_group[0],
        'Protein Class ID': id_group[1],
        'Target ChEMBL ID': id_group[2],
        'Fold': id_group[3],
        'RMSE': rmse
    })

# Sort the predictions data as specified
data.sort_values(by=['Fold', 'Target Parent ID', 'Protein Class ID',
'Target ChEMBL ID'], inplace=True)

# Save the predictions file
data_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_protein_class_rf_ibm
tl_lss_prediction.txt'
data.to_csv(data_save_path, sep='\t', index=False)

# Convert the results list to a DataFrame and compute grouped statistics
(outer performance) per target
results_df = pd.DataFrame(results)
final_results = results_df.groupby(['Target Parent ID', 'Protein Class
ID', 'Target ChEMBL ID']).agg({
    'RMSE': ['mean', 'std']
}).reset_index()
final_results.columns = [
    'Target Parent ID', 'Protein Class ID', 'Target ChEMBL ID',
    'IBMTL AA-LSS Average RMSE', 'IBMTL AA-LSS RMSE SD'
]

# Merge with 'Average Similarity' from the input file
average_similarity = data[['Target Parent ID', 'Protein Class ID',
'Average Similarity']].drop_duplicates()
final_results = final_results.merge(average_similarity, on=['Target
Parent ID', 'Protein Class ID'], how='left')

# Save the statistic file
final_results_save_path = '/Users/Ramadhan/NP-
MTL/STL_MTL_Three_Groups_Studies/kinase_three_groups_protein_class_rf_ibm
tl_lss_statistics.txt'
final_results.to_csv(final_results_save_path, sep='\t', index=False)
```

Code S26. Data preprocessing for the natural product virtual screening model construction

```
# Read the tab-delimited file into a DataFrame
df = pd.read_csv("/Users/Ramadhan/NP-
MTL/Filtering_Preprocessing/kinase_three_groups_target_parent_final_datas
et_gss_for_stl_mtl", sep="\t", low_memory=False)

# Filter the DataFrame for rows where 'Target Parent ID' equals 1289.
# Adjust the type (int or string) as needed; here we assume it's numeric.
filtered_df = df[df["Target Parent ID"] == 1289]

# Define the list of columns to remove: "Sim_4" to "Sim_34"
cols_to_remove = [f"Sim_{i}" for i in range(4, 35)]

# Drop the specified columns. Using errors='ignore' in case some columns
are missing.
filtered_df = filtered_df.drop(columns=cols_to_remove, errors='ignore')

# Write the filtered DataFrame to a new tab-delimited file
filtered_df.to_csv("/Users/Ramadhan/NP-
MTL/Virtual_Screening/kinase_three_groups_target_parent_final_dataset_gss
_for_virtual_screening.txt", sep="\t", index=False)
```

Code S27. Natural product virtual screening model construction

```
# -*- coding: windows-1252 -*-

import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
import joblib

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Virtual_Screening/kinase_three_groups_target_parent_final_dataset_gss
_for_virtual_screening.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare feature matrix X and target vector y
X = np.stack(data['Features'].values)
y = data['pIC50'].values
```

```
# Define fixed hyperparameters
n_estimators = 199
max_depth = 17

# Train the Random Forest Regressor with fixed hyperparameters
model = RandomForestRegressor(n_estimators=n_estimators,
max_depth=max_depth, random_state=100)
model.fit(X, y)

# Save the trained model to a joblib file
model_save_path = '/Users/Ramadhan/NP-
MTL/Virtual_Screening/virtual_screening_akt2_model.joblib'
joblib.dump(model, model_save_path)
```

Code S28. Data preprocessing for the natural product virtual screening

```
import pandas as pd
from rdkit import Chem
from rdkit.Chem import SaltRemover, MolFromSmiles, MolToSmiles
from rdkit.Avalon import pyAvalonTools

# Initialize the salt remover
remover = SaltRemover.SaltRemover()

# Function to process SMILES and calculate Avalon fingerprint
def process_smiles(smiles):
    try:
        mol = MolFromSmiles(smiles)
        if mol:
            mol = remover.StripMol(mol)  # Remove salts and clean the
molecule
            cleaned_smiles = MolToSmiles(mol)
            fp = pyAvalonTools.GetAvalonFP(mol, nBits=1024)  # Generate
Avalon fingerprint
            return [cleaned_smiles, list(fp)]
    except Exception as e:
        print(f"Error processing SMILES {smiles}: {e}")
    return [None, None]

# Read the tab-separated text file
file_path = '/Users/Ramadhan/NP-
MTL/Virtual_Screeing/natural_product_structures.txt'
data = pd.read_csv(file_path, sep='\t', low_memory=False)

# Apply the Avalon fingerprint conversion to the "smiles" column
fingerprint_data = data['smiles'].apply(process_smiles)

# Split the result into two new columns: 'Cleaned_SMILES' and 'Avalon_FP'
data[['Cleaned_SMILES', 'Avalon_FP']] =
pd.DataFrame(fingerprint_data.tolist(), index=data.index)

# Add the new "Target ChEMBL ID" column with a fixed value of "2431"
data.insert(data.columns.get_loc('Cleaned_SMILES') + 1, 'Target ChEMBL
ID', '2431')

# List of updated values for the Sim columns
```

```
sim_values = [
    1, 0.8162839248434238, 0.7857142857142857
]

# Add the new "Sim_" columns with their respective values
for i, value in enumerate(sim_values, start=1):
    column_name = f'Sim_{i}'
    data[column_name] = value

# Move the 'Avalon_FP' column to the end
columns = [col for col in data.columns if col != 'Avalon_FP'] +
['Avalon_FP']
data = data[columns]

# Save the updated file
output_path = '/Users/Ramadhan/NP-
MTL/Virtual_Screeing/natural_product_structures_final_data.txt'
data.to_csv(output_path, sep='\t', index=False)
```

Code S29. Natural product virtual screening

```
import pandas as pd
import numpy as np
import joblib

# Load dataset
file_path = '/Users/Ramadhan/NP-
MTL/Virtual_Screeing/natural_product_structures_final_data.txt'
data = pd.read_csv(file_path, sep="\t", low_memory=False)

# Identify 'Sim_*' columns dynamically
sim_columns = [col for col in data.columns if col.startswith('Sim_')]

# Process 'Avalon_FP' values
data['Avalon_FP'] = data['Avalon_FP'].apply(lambda x:
np.fromstring(x.strip('[]'), sep=','))

# Include 'Target ChEMBL ID' as a feature, placing it before
'sim_columns'
data['Features'] = data.apply(lambda row: np.concatenate(
    [[row['Target ChEMBL ID']],
row[sim_columns].dropna().values.astype(float), row['Avalon_FP']]),
axis=1)

# Prepare feature matrix X
X = np.stack(data['Features'].values)

# Load the trained Random Forest Regressor model
model_path = '/Users/Ramadhan/NP-
MTL/Virtual_Screening/virtual_screening_akt2_model.joblib'
model = joblib.load(model_path)

# Predict pIC50 values
Prediction = model.predict(X)

# Add predictions to the DataFrame
```

```
data['Prediction'] = Prediction

# Save the predictions to a new file
output_file = '/Users/Ramadhan/NP-
MTL/Virtual_Screeing/natural_product_activity_akt2_prediction.txt'
data.to_csv(output_file, sep='\t', index=False)
```