ENHANCING MULTIMODAL DRUG-TARGET INTERACTION PREDICTION WITH DOMAIN ADAPTATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ARDAN YILMAZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2025

Approval of the thesis:

ENHANCING MULTIMODAL DRUG-TARGET INTERACTION PREDICTION WITH DOMAIN ADAPTATION

submitted by **ARDAN YILMAZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Naci Emre Altun
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Halit Oğuztüzün
Supervisor, **Computer Engineering, METU** _____

Prof. Dr. Mehmet Volkan Atalay
Co-supervisor, **Quinlan School of Business, LUC** _____

Examining Committee Members:

Prof. Dr. Tunca Doğan
Computer Engineering, Hacettepe University _____

Prof. Dr. Halit Oğuztüzün
Computer Engineering, METU _____

Assist. Prof. Dr. Aybar Can Acar
Graduate School of Informatics, Medical Informatics, METU _____

Date:31.01.2025

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: ARDAN YILMAZ

Signature :

ABSTRACT

ENHANCING MULTIMODAL DRUG-TARGET INTERACTION PREDICTION WITH DOMAIN ADAPTATION

YILMAZ, ARDAN
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Halit Oğuztüzün
Co-Supervisor: Prof. Dr. Mehmet Volkan Atalay

February 2025, 96 pages

Drug-target interaction (DTI) prediction remains challenging due to the scarcity of annotated data in a vast input space. We treat DTI prediction as a binary classification problem with two inputs (drugs and proteins) and employ a state-of-the-art cross-attention mechanism to fuse these modalities. However, the data distribution typically differs between training and inference settings, making standard random splits overly optimistic. To simulate realistic conditions, we use dissimilar drug-protein pairs in training and test sets, introducing a domain shift. We then apply domain adaptation to learn a domain-invariant feature extractor, aligning source and target distributions alongside a primary classifier. In particular, we leverage advanced methods, including Maximum Mean Discrepancy (MMD) Loss—which, to the best of our knowledge, has not been used previously for DTI prediction—and adversarial training for robust feature extraction. Our multimodal learning with domain adaptation achieves performance on par with the state of the art on the widely used BindingDB dataset, demonstrating the effectiveness of our approach even under domain shifts.

Keywords: Drug-Target Interaction Prediction, Multimodal Learning, Domain Adaptation, Transfer Learning

ÖZ

ALAN ADAPTASYONU İLE ÇOK KİPLİ İLAÇ-HEDEF ETKİLEŞİM TAHMİNİNİN İYİLEŞTİRİLMESİ

YILMAZ, ARDAN
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Halit Oğuztüzün
Ortak Tez Yöneticisi: Prof. Dr. Mehmet Volkan Atalay

Şubat 2025 , 96 sayfa

İlaç–hedef etkileşimi tahmini, geniş genomik ve moleküler veri uzayında etiket eksikliği nedeniyle biyoenformatik ve kimyasal-enformatik alanlarında zorlu bir problem olarak karşımıza çıkar. Bu problem, ilaç ve protein olmak üzere iki girdiyi kullanarak ikili sınıflandırma biçiminde modellenebilir ve çok kipli öğrenme yaklaşımıyla ele alınabilir. Farklı kiplerden gelen özelliklerin etkileşimini doğrudan incelemek için en yeni algoritmalardan olan çapraz dikkat (cross-attention) mekanizmasını kullanıyoruz. Ancak eğitim ve test aşamalarında verinin dağılımı genellikle farklılık gösterdiğinden, rastgele eğitim/test bölünmeleri gerçekçi olmayan ve iyimser sonuçlara yol açmaktadır. Daha gerçekçi bir senaryo için, eğitim ve test kümelerinde istatistiksel olarak farklı ilaç–protein çiftleri kullanarak bir alan kayması (domain shift) oluşturuyoruz. Bu kaymayı gidermek amacıyla, asıl sınıflayıcının yanı sıra alan uyarlama yöntemlerini kullanarak, kaynak ve hedef dağılımları aynı alana hizalayabilen alan-bağımsız bir nitelik çıkarıcı eğitiyoruz. Bunun için, çekişmeli (adversarial) eğitim ve (bildiğimiz kadarıyla DTI tahmini bağlamında daha önce kullanılmamış olan) Maksi-

mum Ortalama Farklılık (MMD) Kayıp Fonksiyonu gibi ileri düzey yöntemlerden yararlanıyoruz. Çok kipli öğrenme ve alan uyarlamasını bir araya getiren yaklaşımımız, yaygın olarak kullanılan BindingDB veri kümesinde literatürdeki en iyi yöntemlerle karşılaştırıldığında çeşitli metriklerde üstün veya benzer performans sergilemekte ve alan kayması durumunda dahi etkinliğini kanıtlamaktadır.

Anahtar Kelimeler: İlaç-Hedef Etkileşimi Tahmini, Çok Kipli Öğrenme, Alan Uyarlama, Öğrenme Aktarımı

To those who've acted in ways that turn life into a wonderful adventure—the blessings in my life, with all my heart.

ACKNOWLEDGMENTS

Throughout the course of this research, I have been exceptionally fortunate to receive support from my family, friends, and colleagues, some of whom deserve special mention. My mother, Sabiha Doğan, has shown exceptional and selfless support, for which I am profoundly grateful. I also owe immense gratitude to Deniz Germen, whose intellectual contributions and companionship have been of utmost importance. Additionally, I am deeply thankful for the guidance from my supervisor, Volkan Hoca, whose insights and assistance have been invaluable. Last but not least, I extend my heartfelt thanks to Tunca Hoca who, although not my official supervisor, has played a very influential role in shaping my academic journey from the outset.

My years at ODTU have greatly enriched my life, thanks to the incredible people I've met along the way. In particular, my dear friend and housemate, Elifnaz Eker, with her spark, has been phenomenal in my life. Şeyma Ertürk, my former housemate, has contributed in countless ways. Gizem Yıldırım, one of my closest friends, has been pivotal in my personal development. I deeply appreciate Musa Batır for his genuine presence and company. I am grateful to Yağmur Yüksel for offering a unique perspective that has always been enlightening. Despite meeting Ilgaz and Ekin later in my ODTU years, they have significantly impacted my outlook on life. Though our friendship is still relatively new, Ahmet Doğan's company has deepened my appreciation for the meaning of love and friendship. I am indebted to many more extraordinary individuals, including but not limited to Naz, Çağrı, Melis, Yağız, Şamil, Gözde, and others who have made this journey wonderful.

All these remarkable people, with their unique perspectives, have contributed significantly to my life. I hold their contributions in the highest regard and am deeply thankful for the insight they have shared.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AUROC	Area Under the Receiver Operating Characteristic
BCE	Binary Cross Entropy
BLAST	Basic Local Alignment Search Tool
BERT	Bidirectional Encoder Representations from Transformers
CDAN	Conditional Adversarial Domain Adaptation
DA	Domain Adaptation
DDAN	Domain-Adversarial Neural Network
DTI	Drug-Target Interaction
ECFP4	Extended-Connectivity Fingerprints
ESM	Evolutionary Scale Modeling
fpr	False Positive Rate
GAN	Generative Adversarial Network
GCN	Graph Convolutional Network
GRL	Gradient Reversal Layer
HAC	Hierarchical Agglomerative Clustering
ITL	Inductive Transfer Learning
MCC	Matthews Correlation Coefficient
MLP	Multi-layer Perceptron
MMD	Maximum Mean Discrepancy
NLP	Natural Language Processing
PLM	Protein Language Modeling
PSC	Position-Specific Counts
ROC	Receiver Operating Characteristic
RKHS	Reproducing Kernel Hilbert Space

SMILES	Simplified Molecular Input Line Entry System
t-SNE	t-distributed Stochastic Neighbor Embedding
TL	Transfer Learning
tpr	True Positive Rate
TTL	Transductive Transfer Learning

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

1.1.1 Context and Motivation

Drug-Target Interaction (DTI) prediction is a critical task in drug discovery and personalized medicine. This remains a crucial challenge in bioinformatics and cheminformatics due to the vastness of genomic and molecular input space with limited annotated data [2]. Traditional wet lab experiments are highly resource-intensive, costly, time-consuming, and laborious. Accurately identifying which small molecules will bind to which macromolecular structures (proteins) can significantly streamline and reduce the cost of early-stage drug development. This is where computational methods come in handy for narrowing down the set of candidate compounds, aiming at a target protein, before experimental validation.

1.1.2 Challenges Addressed in This Study

Drug-Target Interaction prediction can be formulated as a binary classification task: given a drug-protein pair, the objective is to predict whether they will interact. This setting is suitable for multimodal learning, where the protein and drug are input into the system, and their feature vectors need fusion before being fed to the interaction classifier.

Furthermore, the limited availability of annotated data in the huge input space leads to distributional shifts between training and inference data. As a consequence, models

that perform well during training may fail to generalize to novel instances at inference time. This problem can be regarded as a domain shift. To address this challenge, we incorporate domain adaptation techniques. This approach aims to learn a domain-invariant feature extractor and an interaction classifier with robustness across different domains, thereby improving its ability to generalize to unseen data.

1.1.3 Limitations of Conventional Approaches and the Need for Domain Adaptation

Despite promising results reported in DTI prediction literature, employing standard random train-test splits has been shown to lead to over-optimistic evaluations [2]. Such data splits assume that the training and evaluation distributions are similar, an assumption that does not hold in real-world DTI settings. The training data covers only a small portion of the entire chemical and genomic input space, due to the vastness of the input space and the lack of annotations. At inference time, novel compounds or protein instances exhibit different distributional properties than those seen during training. Thus, the learned models cannot generalize well to the unseen data at inference time.

1.2 Approaches to Feature Extraction

Computational models require conversion of raw input data (e.g., protein sequences and molecular structures) into numerical representations suitable to feed machine learning algorithms.

The protein representation is typically handled as a Natural Language Processing (NLP) problem over the language defined by the amino acids. Commonly found amino acids in nature compose a 20-letter alphabet that defines a language for proteins. There are multiple methods to convert these sequences into real-valued feature vectors that use both statistical methods and advanced deep learning approaches.

For small-molecule drugs, on the other hand, the SMILES notation is typically used to represent their chemical structures textually. Similar to protein modeling, drug

molecules can be featurized using traditional methods and sophisticated transformer-based models that learn detailed chemical properties from these notations.

1.3 Domain-Invariant Representation Learning for DTI:

To address similar domain shift problems outlined in Section 1.1.3, particularly within the field of computer vision, researchers have devised strategies for domain adaptation aimed at domain-invariant feature learning. An illustrative example is the use of training datasets for autonomous vehicles. Consider a system trained exclusively on images of pedestrians during sunny conditions. The system must recognize pedestrians across all weather conditions, with a significant domain shift from the training to the application environment. This exemplifies a domain shift where the same task is undertaken across similar, yet distinctively different datasets. To bridge this gap between the training and inference datasets, domain adaptation techniques have been explored. These methods focus on developing features that remain robust across domains.

To simulate the domain shifts between training and evaluation data in DTI, researchers have proposed creating source and target domains by clustering the data [2]. In particular, Hierarchical Agglomerative Clustering (HAC) is proposed to construct source and target datasets composed of dissimilar subsets of proteins and drugs. By the principle of separation of clusters, this method results in target and source datasets that pose different statistical distributions, introducing a domain shift. This process, overall, establishes a setting suitable for domain adaptation. Training models on source data and subsequently evaluating them on target data reflects a more realistic scenario for DTI prediction, accounting for the distributional shift likely to be encountered in the inference data.

Although domain adaptation techniques have been applied in the DTI literature, there is still a substantial need for further research. To this end, we explore two widely adopted strategies in the literature: the Maximum Mean Discrepancy (MMD) loss [3] and adversarial training approaches, such as Domain-Adversarial Neural Network (DDAN) [4] and Conditional Adversarial Domain Adaptation (CDAN) [5]. Using

these domain adaptation methods, we aim at learning a domain-invariant feature extractor and an interaction classifier that, although trained exclusively on source data, can effectively generalize to the target data in a zero-shot setting.

1.4 Contributions and Novelties

Our primary contributions are:

- Thorough analysis of the effect of fusion strategies in multi-modal learning frameworks for DTI prediction.
- The first-time use of Maximum Mean Discrepancy (MMD) loss for cross-domain alignment in DTI research, exploring its potential benefits.
- A novel strategy to schedule the contribution of domain adaptation throughout training, rather than treating it as a fixed hyperparameter.

1.5 The Outline of the Thesis

Chapter 2 offers background information and a review of existing literature on protein and drug modeling, strategies for multi-modal input fusion and learning, transfer learning, domain adaptation. Building on this foundation, Chapter 3 describes the research methodology, detailing the data sources, feature extraction processes, and input fusion strategies employed in the experiments. It introduces the training and evaluation pipelines for in-domain and cross-domain tasks, with separate consideration for MMD- and adversarial-based adaptation techniques. The initial findings, including the performance change and feature alignment on the validation set during training are presented in this section. The chapter further explains the hyperparameter tuning procedures and data loading mechanisms. Finally, Chapter 4 provides a comprehensive account of the results from these experiments, benchmarks them against state-of-the-art methods, and covers both in-domain and cross-domain settings. In addition, it presents our reflections and insights regarding our experimental results.

CHAPTER 2

BACKGROUND INFORMATION & RELATED WORK

Drug-target interaction prediction problem, being an interdisciplinary subject, requires a deep understanding of a wide array of topics, including molecular biology, bioinformatics, cheminformatics, and advanced computational techniques. In this chapter, we survey comprehensive relevant background information and prior work for our study.

For this, we begin by presenting approaches to protein language modeling, which embeds the amino acid sequences into learned feature representations. Next, we explore various methods for learning drug (small-molecule) representations. We then discuss multi-modal fusion strategies to combine protein and drug embeddings for predictive tasks. This is followed with a review of the key concepts in transfer learning and domain adaptation. Then, we conclude with an overview of DTI prediction.

2.1 Protein Language Modeling (PLM)

Amino acids that are commonly found in nature compose a 20-letter alphabet over which proteins are formed. This makes an intuitive setup analogous to natural language processing (NLP) tasks. Being one of the most popular topics, NLP techniques are extensively studied and offer a very promising set of tools for representation learning.

Early studies employ statistical methods based on comparing new sequences to the known ones, employing alignment-based algorithms. For instance, BLAST [6] searches against a protein database to identify regions of similarity, while HMMER [7] uses a

probabilistic profile hidden Markov model to capture conserved motifs. PFAM [8], on the other hand, further extends these ideas by offering a comprehensive database of protein families, each represented by multiple sequence alignments.

Subsequent advances introduced machine/deep learning-based methods, by learning embeddings as continuous vector representations of protein sequences. ProtVec [9] was among the first to adopt word2vec-inspired embeddings for biological sequences, using k-mer statistics to preserve local contexts.

More recently, with the advancements in NLP and transformer-based architectures, methods with higher capacity in learning representations have been proposed. For example, the MSA-Transformer [10] processes multiple sequence alignments to infer deep evolutionary relationships. ProtBERT and ProtT5-XL [11, 12] leverage self-supervised objectives (inspired by BERT [13] and T5 [14]) to learn rich bidirectional or encoder–decoder representations, capturing both local residue context and long-range dependencies. Similarly, ESM-2 [15] refines large-scale protein language modeling for harder tasks like structure prediction, while ESM-3 [16] extends these by employing evolutionary information at large sequence scales.

Beyond these transformer models, new approaches continue to explore multi-modal or structure-informed representations. ProstT5 [17] proposes a "bilingual" architecture that jointly models sequence and structural information. In a similar attempt to explore multimodal approaches, SaProt [18] integrates sequence data with structure-aware vocabulary tokens, thereby improving contextual embeddings for proteins with limited labeled data.

Altogether, these modern PLM methods provide robust representations that are commonly used for a wide array of protein-related predictive tasks, including function, structure, and drug–protein interaction prediction.

2.2 Drug Representation Learning

A typical approach to represent drugs is through the Simplified Molecular Input Line Entry System (SMILES) [19]. This linear string notation effectively encodes the

molecular graph structure. Early methods to featurize these string representations of drugs often transform them into binary vectors, where each bit represents the presence of specific chemical substructures within defined neighborhoods. A notable example is the Extended-Connectivity Fingerprints (ECFP4) [20], which determines the presence of substructures through iterative expansion of atom neighborhoods up to four bonds away.

The advent of deep learning methods has enabled richer embeddings of drug molecules, employing transformer and BERT-based architectures. Compelling examples include ChemBERTa [21] and SELFormer [22] that pretrain networks for drug representation learning.

Moreover, an intuitive way to represent molecules is the use of graphs. Accordingly, attempts at capturing deeper relations regarding molecules employ graph and hyper-graph-based learning methods. Wang et al.[23] provides a review on the use of graph neural networks for molecular representation. Also, hyper-graph-based approaches, such as HyperGCN [24] effectively capture the complex connectivity and interactions within molecules, potentially leading to improved predictive power for molecular properties.

2.3 Multi-modal Input Fusion

Our approach takes both protein and drug representations as inputs, which must be fused before classification to predict their potential interaction. In general, multimodal fusion refers to the process of integrating data from multiple modalities to leverage complementary information performance [25]. A crucial design decision lies in determining how and when to fuse these modalities within a learning architecture.

2.3.1 Early vs. Late Fusion

Fusion strategies are commonly classified as early or late. Early fusion (also referred to as "feature-level fusion") combines raw embedding vectors near the input layer [25]. Simple concatenation of the feature vectors is a compelling example of

this.

Late fusion (or "decision-level fusion") processes each modality independently, often with separate neural networks, and then merges the resulting predictions at the final layer [26]. While this preserves modality-specific pipelines, it can overlook nuanced relationships in between modalities (e.g., a particular binding site in the protein aligning with a functional substructure of the drug).

2.3.2 Concatenation-based Fusion

One straightforward method for this is simple concatenation. Though does not provide an explicit mechanism to capture intricate cross-modal interactions, this appears to be a simple yet effective strategy in the literature [25]. Suppose $\mathbf{p} \in \mathbb{R}^{d_p}$ denotes a learned protein embedding and $\mathbf{d} \in \mathbb{R}^{d_d}$ a drug embedding. Then, the fused representation is constructed as

$$\mathbf{z} = [\mathbf{p}; \mathbf{d}] \in \mathbb{R}^{d_p + d_d}.$$

This vector $\mathbf{z}$ is fed to the main classifier network to predict interaction scores and labels.

To increase modeling capacity, self-attention [27] can be applied immediately after concatenation. In this setup, the concatenated embeddings are passed into self-attention layers, enabling the network to compute the attention scores for both intra-modal (protein–protein) and inter-modal (drug–protein) relations.

2.3.3 Cross-Attention for Fusion

While self-attention jointly processes a single combined sequence, cross-attention explicitly learns to attend from one modality to another [27, 28]. Like standard attention, cross-attention relies on the concepts of Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$) matrices, which are typically derived via learned linear projections of the input embeddings. Formally, the attention mechanism computes

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\Big)\,\mathbf{V},$$

where d_k is a scaling factor (the dimension of $\mathbf{K}$).

In standard self-attention, all Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$) matrices are derived from the same modality, where the attention formula above allows the network to capture the relations within that modality. To accommodate for the multi-modal setting (i.e., to compute the attention scores across different modalities), the $\mathbf{K}$ and $\mathbf{V}$ matrices for one modality are calculated using the other modality, while using the same modality to calculate the $\mathbf{Q}$ matrix [29].

This setup focuses the attention mechanism on inter-modal relationships: each protein "queries" the drug embeddings to find relevant substructures, and vice versa. This query is basically measured as a dot product between items from each matrix, referred to as attention. As a result, cross-attention can learn explicit interactions between specific amino acid regions in the protein and groups in the drug.

2.4 Transfer Learning and Domain Adaptation

Transfer learning (TL) addresses the challenge of leveraging knowledge extracted from one task (the source) to improve learning or performance in another, potentially different but related, setting (the target) [30, 31]. It is particularly useful in scenarios where sufficient labeled data exist for one task, but there is limited labeled data for a related task. Rather than training a new model from scratch for the domain with limited annotated data, TL allows us pre-train a model on a sufficiently large dataset (of related task) and then adapt or reuse the learned features for the target task.

Several strategies for TL are widely employed, depending on the similarity of the source and target tasks as well as the amount of available target data [32, 33]:

- **Full Fine-Tuning:** A model is pre-trained on the source task and then all its (learnable) parameters are updated (fine-tuned) using the limited labeled data of the target task [30].
- **Gradual (Layer-wise) Fine-Tuning:** The model's layers are incrementally unfrozen. The standard approach is to first tune the layers close to the output layer, while the others remain frozen. Subsequently, earlier layers are unfrozen and

tuned in a stage-by-stage manner [34].

- **Fine-Tuning Final Layers Only:** The lower layers, generally attributed to the task of feature extraction, are frozen, and only the task-specific final layers (e.g., classification layers) are fine-tuned [32].

- **Feature Extraction + Shallow Classifier:** The pre-trained network is treated purely as a feature extractor to yield expressive embeddings from the input data. For the output layer, a shallow machine learning model (e.g., logistic regression, SVM, or random forest) is attached to be trained on the target data with limited data. Thus, we learn a feature extactor on large datasets and further specify a shallow model for the target-specific task. [31].

Beyond these general adaptation approaches, TL also encompasses scenarios in which distributions differ significantly across domains ($D_s \neq D_t$) or the predictive tasks differ ($T_s \neq T_t$). When the source and target tasks remain the same but the underlying data distributions differ, the problem is commonly referred to as domain adaptation (DA) [30]. Domain adaptation can therefore be viewed as a sub-field of TL, focusing on the challenge of aligning or bridging the distributional shift between source and target domains.

The remainder of this section focuses on domain adaptation methods. In particular, we will focus on zero-shot adaptation scenario, assuming zero supervision from the target domain during training.

2.4.1 Transfer Learning Settings

Inductive Transfer Learning (ITL). In inductive TL, both the source and target domains have labeled data, possibly with different quantities of samples, for slightly different but related tasks [30]. The main objective is to leverage knowledge gained from the labeled source domain to improve performance on the labeled target task. An example is fine-tuning a large pre-trained model (trained on a labeled source dataset) to perform a different but related labeled task with limited training data in the target domain.

Transductive Transfer Learning (TTL). In transductive TL, the source domain has labeled data, but the target domain may have very limited labeled data or, in many cases, no labeled data at all [30]. Here, the label space is typically assumed to be identical (or heavily overlapping) between source and target, but the data distributions differ. Many domain adaptation (DA) methods fall under transductive TL, especially unsupervised domain adaptation, where no target labels are used at all. In this scenario, the algorithm exploits unlabeled target data to align source and target distributions, ensuring that the model trained on the labeled source can generalize effectively to the target domain.

2.4.2 Domain Adaptation

Domain adaptation is often regarded as a sub-problem within transductive TL, where the main difference between source and target lies in the data distributions rather than the tasks. Under the classical DA assumption, the label (or task) space remains identical ($T_s = T_t$) or nearly the same between source and target [30]. For instance, we might have the same classification task (e.g., object recognition) in both source and target, but the images or features in the target domain follow a distinct distribution.

Figure 2.1: An example of domain shift in the computer vision domain. This figure is adapted from [1].

Figure 2.1, adapted from [1], illustrates a compelling example of domain shift in the vision domain. Imagine an object detection task where the same scene is captured under different lighting conditions. Although the objects present in both images are identical, the differences in luminance and color distributions introduce a domain shift. If the training set predominantly contains images similar to the left-hand side

of the figure, a robust model is required to generalize effectively to images like those on the right-hand side that were not seen during training.

2.4.2.1 Domain Adaptation for DTI

A similar concept applies to Drug-Target Interaction (DTI) prediction. In the context of DTI, while the task remains the same—determining whether a drug and a protein interact—the structural properties of the molecules can vary significantly. For example, two molecules with distinct structural features can be said to exhibit the same functional properties when interacting with a specific protein. This is analogous to the computer vision scenario where different lighting conditions do not change the identity of the objects. In DTI, even if the structural representations of drugs vary (similar to different lighting conditions in images), a robust model should still recognize that their functional interactions with the protein are equivalent. Thus, by employing domain adaptation techniques, we aim to learn a domain-invariant feature extractor that ensures robust interaction prediction regardless of the variability in molecular representations.

2.4.2.2 Supervised, Semi-Supervised, and Unsupervised DA

Domain adaptation methods also differ in how they use labeled data in the target domain:

Supervised DA. In this case, a small amount of labeled target data is available. Models typically leverage this limited labeled set alongside the more abundant labeled source data to calibrate or fine-tune predictions in the target domain.

Semi-Supervised DA. Here, a small set of labeled target samples and a larger set of unlabeled target samples are available. Methods combine the ideas of semi-supervised learning and domain adaptation, guiding the model using both labeled and unlabeled target data.

Unsupervised DA. No labeled data exist in the target domain; the algorithm relies solely on unlabeled target samples. This is the most challenging and widely studied

setting [35, 36, 5], pushing the model to learn domain-invariant features without any label guidance from the target.

Zero-Shot DA. In some scenarios, although the target labels are available, they are not used for supervision to simulate a zero-shot evaluation in the target domain. Our work adopts this perspective, ensuring that no target supervision is used for the classification.

2.4.3 Distribution Alignment Approaches

A foundational component of domain adaptation is to align the feature distributions across source and target data. By enforcing distribution similarity, a model learns representations that are domain-invariant. Two prominent strategies are:

Marginal Distribution Alignment seeks to match global (marginal) distributions without explicit consideration of class or label information. A common technique for this is the Maximum Mean Discrepancy (MMD), which compares the mean embeddings of distributions in a reproducing kernel Hilbert space [3].

Class-Conditional Distribution Alignment leverages label or pseudo-label information to align distributions *within* each class. This approach typically produces stronger results when reliable class-conditional structure can be identified. Conditional Adversarial Domain Adaptation (CDAN) [5] exemplifies this idea by conditioning a domain discriminator on both feature representations and predicted labels, promoting alignment at a finer granularity.

2.4.4 Adversarial Domain Adaptation

Adversarial learning techniques formulate domain adaptation as a minimax game between a feature extractor and a domain discriminator:

Domain-Adversarial Neural Network (DANN) [36] introduces a gradient reversal layer to encourage the feature extractor to produce representations that fool a domain discriminator. By optimizing this objective alongside a task-specific loss (e.g.,

classification or regression in the source domain), the model learns domain-invariant features.

Conditional Adversarial Domain Adaptation (CDAN) [5] builds on DANN by incorporating label information (or predicted labels) into the adversarial objective. This addition performs class-conditional alignment, improving adaptation performance especially when different classes exhibit markedly different distributions across domains.

These adversarial methods are particularly relevant in an unsupervised setting, where the model relies on distributional cues and source supervision to bridge the source-target gap. They are widely applied in computer vision and natural language processing, and have recently shown promise in biomedical tasks, such as drug-target interaction prediction, by enabling robust cross-dataset or cross-species generalization.

2.5 Drug–Target Interaction (DTI) Prediction

Drug–Target Interaction (DTI) prediction is a key step in computer-aided drug discovery, aiming to identify potential relationships between chemical compounds (drugs) and biological targets (often proteins). As summarized by Chen et al. [37], early DTI studies frequently employed classical machine learning methods alongside engineered molecular descriptors, but these approaches often struggled with capturing the intricate structural and functional diversity of molecules and proteins. More recently, deep learning architectures have emerged as powerful tools for end-to-end representation learning, achieving impressive results in many prediction tasks.

However, the huge molecular and genomic space, coupled with the scarcity of annotated data, remains a significant challenge for DTI prediction. One major issue is that limited annotated data in specific domains can make deep model training exceptionally difficult. Furthermore, many real-world DTI scenarios involve distribution shifts between training and inference data, complicating model deployment.

In an effort to address the challenge of limited annotated data, transfer learning has

emerged as a promising approach. Cai et al. [38] provide a comprehensive survey of transfer learning techniques applied to DTI prediction. One compelling example is the work by Dalkıran et al. [39], which illustrates how pre-trained models on large-scale drug or protein datasets can be repurposed or fine-tuned for new tasks with minimal additional supervision. This style of deep transfer learning holds particular promise when labeled data for specific targets or compound classes are limited.

The issue of distributional shift between training and inference data has been noted by Bai et al. [2]. They show that standard random train–test splits (assuming similar distributions in training and inference data) yields overoptimistic performance estimates, as per the distributional shift posed by novel input at inference time. To address these discrepancies, researchers have turned to domain adaptation, a specialized subfield of transfer learning that focuses on aligning source and target data distributions.

One popular domain adaptation framework is the conditional domain adversarial network (CDAN) [5], which aligns feature representations with an adversarial objective. **DrugBAN** [40] and **CAT-DTI** [41] both employ the adversarial objective for domain adaptation.

- **DrugBAN** employs a bilinear attention mechanism to model local drug–protein interactions and integrates adversarial alignment to mitigate distribution shifts across diverse datasets.
- **CAT-DTI** combines convolutional neural networks and Transformer-based encoders with a cross-attention module, capturing intricate drug–target relationships. A conditional adversarial component further refines alignment, enhancing predictive performance on out-of-distribution data.

CHAPTER 3

METHODS

This section details the methods we employ to address the DTI prediction problem and how we tackle the challenges of multimodal input fusion and domain adaptation.

At its core, our research handles a binary classification problem where we determine whether a given drug-protein pair interacts. This involves a multimodal approach, requiring a fusion of protein and drug features before being fed to the interaction classifier. The overall framework for this is depicted in Figure 3.1.

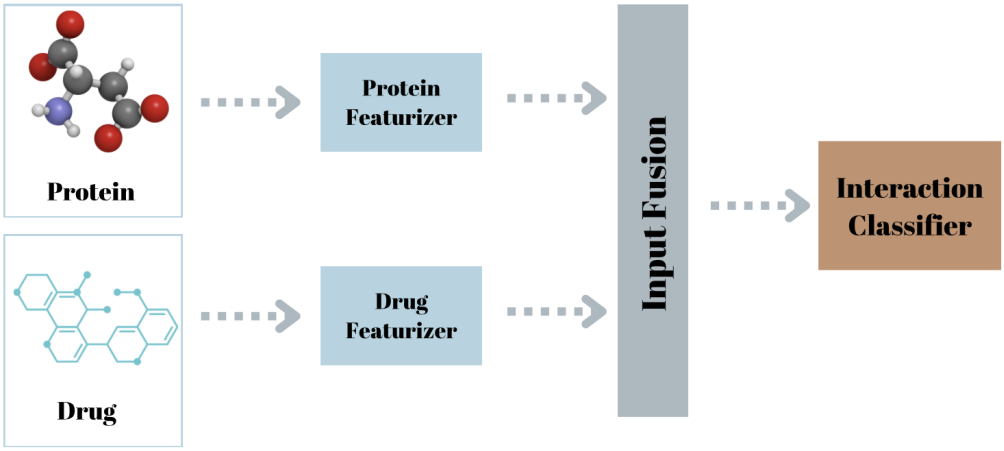


Figure 3.1: In-domain interaction classification setup, illustrating the fusion of protein and drug modalities.

Though existing methods report high predictive performance on test sets crafted via random train-test splits, these results are often not transferable to data at inference

time. In particular, a substantial performance drop is frequently observed when the trained classifier is applied to data from a different distribution ('target domain' compared to the training data, or 'source domain'). We verify this phenomenon in our experiments (See Section 3.4.2), confirming the domain shift problem noted by previous researchers [2]. To mitigate this issue, we incorporate domain adaptation techniques that encourage the learned feature representations to be domain-invariant, allowing the classifier to perform effectively across both domains.

Figure 3.2 conceptually illustrates how alignment methods match the feature distributions from different domains. Specifically, the left panel illustrates marginal alignment, which matches feature distributions regardless of class labels, whereas the right panel demonstrates conditional alignment, taking class labels into account and aligning distributions with respect to their class labels.

In an attempt to mitigate the issue of domain shift, we aim to train a robust feature extractor that generalizes well to unseen target data. For this, we train a domain-invariant feature extractor to align source and target features in a common latent space, similar to the process in Figure 3.2. To this end, we employ various strategies for domain adaptation, detailed in Section 3.5.2 and Section 3.5.1.

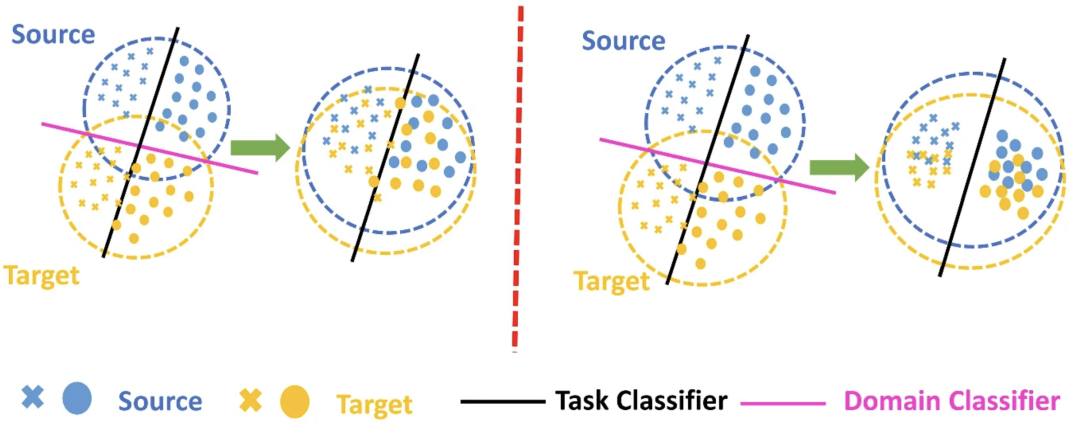


Figure 3.2: In-domain interaction classification setup, illustrating the fusion of protein and drug modalities. Adapted from [42].

3.1 Data

We utilize a low-bias version of the binary BindingDB dataset [**?**] for model training and evaluation. This dataset contains chemical compounds (represented as SMILES strings), proteins (represented by their amino acid sequences), and binary interaction labels indicating whether each drug–protein pair interacts. Following the IC50 threshold protocol described by Gao et al. [43], a drug–target pair is labeled as positive if its IC50 is less than 100 nM and negative if its IC50 is greater than 10,000 nM, ensuring a 100-fold difference between the two classes.

3.1.1 Dataset for In-Domain Setting

For the in-domain experiments, the dataset is randomly split into training, validation, and test sets. The key characteristics are as follows:

- **Training Set:**
  - **Size:** 34,439 instances
  - **Columns:** SMILES, Protein, and binary label Y
  - **Unique Entries:** 13,887 unique SMILES (drugs) and 2,347 unique proteins
  - **Duplicates:** 20,552 duplicate SMILES entries and 32,092 duplicate protein entries (no duplicate (SMILES, Protein) pairs)
  - **Label Distribution:** 19,856 negatives and 14,583 positives
- **Validation Set:**
  - **Size:** 4,920 instances
  - **Unique Entries:** 4,050 unique SMILES and 1,083 unique proteins
  - **Label Distribution:** 2,903 negatives and 2,017 positives
- **Test Set:**
  - **Size:** 9,840 instances

- **Unique Entries:** 6,951 unique SMILES and 1,441 unique proteins
- **Label Distribution:** 5,766 negatives and 4,074 positives

Combined, the full dataset (training + validation + test) contains 49,199 instances, with 14,643 unique SMILES strings and 2,623 unique proteins. The overall label distribution is 28,525 negatives and 20,674 positives.

3.1.2 Data Representation

We need to represent drugs and proteins in real-valued vector formats for processing. For protein representation, we employ NLP-based methods detailed in Section 2.1. Specifically, we use k-mer ($k = 2$) counting and ProtT5-XL [12]. For drug representation, on the other hand, we employ ECFP4 fingerprints [20] and an advanced transformer-based model ChemBERTa [21] (detailed in Section 2.2).

For the in-domain tasks, where we tried to identify the optimal input fusion strategy, we only utilized the simpler representation methods: k-mer ($k = 2$) for proteins and ECFP4 for drugs. However, they did not provide promising results during our preliminary analysis of the cross-domain tasks. Consequently, for all subsequent cross-domain experiments, we opted for the more advanced representation methods (ProtT5-XL for proteins and ChemBERTa for drugs).

Different levels of representation for input significantly affect the overall learning process, impacting both predictive and computational performance. The complexity level is crucial for the remainder of the deep models in the pipeline. That is, the methods later employed for domain alignment need to be on par in terms of their statistical representation power. A similar phenomenon is observed in generative adversarial networks (GANs), where the discriminator's complexity must be comparable to that of the feature extractor to prevent issues such as mode collapse [44]. Our approach, employing a very similar mechanism, is also susceptible to these problems. Therefore, we require that the remainder of the networks in our pipeline possess sufficient statistical complexity to capture the intricacies identified by the featurization method opted for.

3.2 Input Fusion Strategies

The feature representations of proteins and drugs need to be fused to form a joint representation for the interaction classification task, as illustrated in Figure 3.1. For this, we explore several fusion strategies, ranging from simple concatenation to more complex attention-based mechanisms.

3.2.1 Concatenation

The simplest method is direct concatenation of the protein and drug feature vectors ($\mathbf{F}_{protein}$ and $\mathbf{F}_{drug}$):

$$\mathbf{F} = [\mathbf{F}_{protein}; \mathbf{F}_{drug}]$$

This fused vector $\mathbf{F}$ is then passed to the binary interaction classifier, as illustrated in Figure 3.3. Although this approach does not explicitly model direct interactions between the modalities, it serves as a simple yet effective baseline.

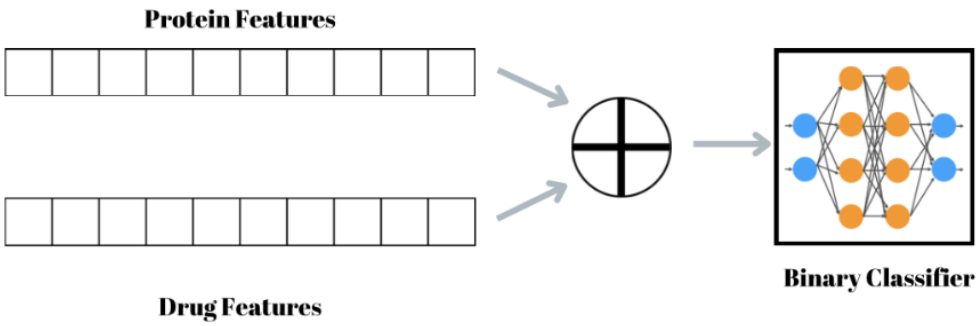


Figure 3.3: Using direct concatenation for input fusion.

3.2.2 Concatenation + Self-Attention

To better capture interactions between the protein and drug feature sets, we incorporate a more sophisticated approach, self-attention [27], after concatenation, as illustrated in Figure 3.4.

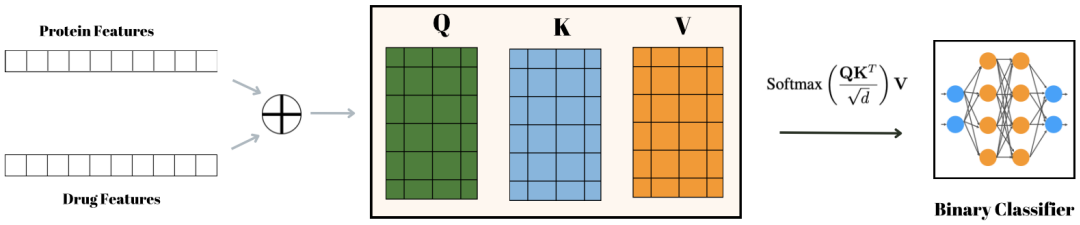


Figure 3.4: Using concatenation and self attention for input fusion.

Let $\mathbf{F} \in \mathbb{R}^{L \times d}$ denote the concatenated features (where L is the combined sequence length and d is the embedding dimension). As in a standard attention setting, we require query ($\mathbf{Q}$), key ($\mathbf{K}$), and value ($\mathbf{V}$) matrices. These are calculated as linear transformations applied on top of the feature vectors of drugs and proteins, computed as follows:

$$\mathbf{Q} = \mathbf{F}W_Q, \quad \mathbf{K} = \mathbf{F}W_K, \quad \mathbf{V} = \mathbf{F}W_V$$

The self-attention mechanism computes attention weights as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}$$

Here, $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are learnable projection matrices. The dot product $\mathbf{Q}\mathbf{K}^T$ measures the similarity of each feature vector with every other feature vector. This approach allows the model to calculate how much each feature we use to represent a modality interacts with all the features from both modalities.

3.2.3 Cross-Attention

In an attempt to capture the relations between modalities more explicitly, we employ a cross-attention mechanism. The goal is to model how features from one modality can directly attend to features from the other.

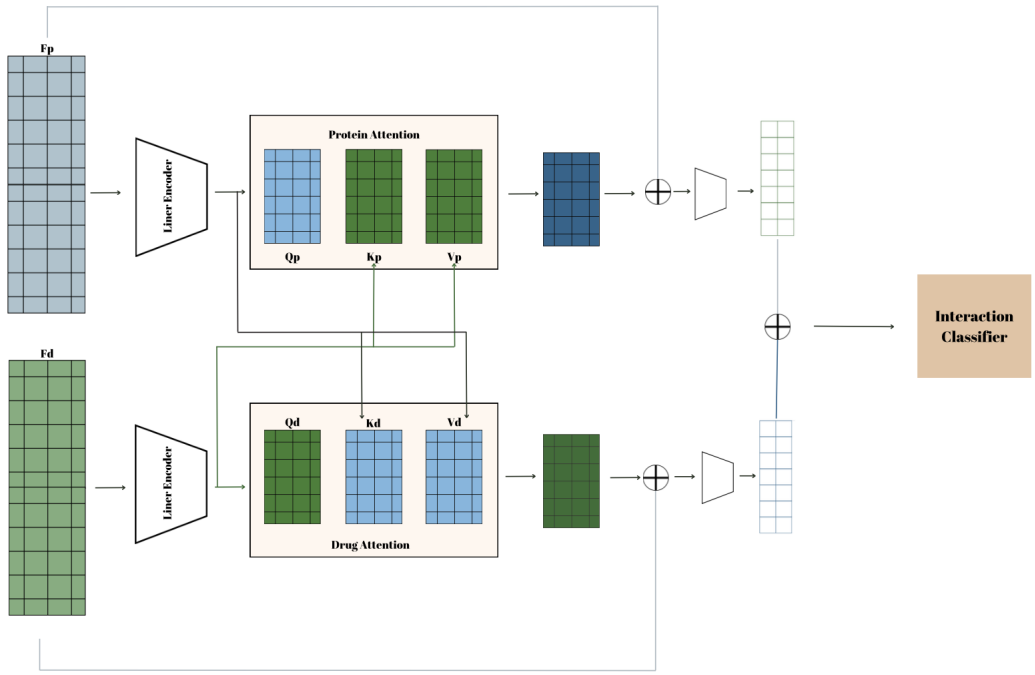


Figure 3.5: Using cross attention for input fusion

This process, demonstrated in Figure 3.5, involves the following steps:

1. **Separate Feature Representations:** We start with separate feature representations for proteins and drugs:

$$\mathbf{F}_p \in \mathbb{R}^{L_p \times d}, \quad \mathbf{F}_d \in \mathbb{R}^{L_d \times d}.$$

   Here, L_p and L_d denote the sequence lengths for proteins and drugs respectively, and d is the embedding dimension.

2. **Cross-Attention Setup:** Unlike standard self-attention where queries, keys, and values for modality are derived from the embedding of the same modality, cross-attention computes queries from one modality and keys/values from the other. Specifically:

   - **Drug-to-Protein Attention**: The drug features are used to compute the query matrix, whereas linear transformations to protein features provide the keys and value matrices for Drug-to-Protein Attention.

- **Protein-to-Drug Attention**: Similar approach to computing the Drug-to-Protein Attention. Specifically, the protein features are used to compute the query matrix, whereas linear transformations to drug features provide the keys and value matrices for Protein-to-Drug Attention.

This setup ensures computing how much each feature from one modality attends to those in the other, accounting for the cross-attention setting.

Formally, for drug-to-protein direction:

$$\mathbf{Q}_d = \mathbf{F}_d W_Q^d, \quad \mathbf{K}_p = \mathbf{F}_p W_K^p, \quad \mathbf{V}_p = \mathbf{F}_p W_V^p,$$

and for protein-to-drug direction:

$$\mathbf{Q}_p = \mathbf{F}_p W_Q^p, \quad \mathbf{K}_d = \mathbf{F}_d W_K^d, \quad \mathbf{V}_d = \mathbf{F}_d W_V^d.$$

Here, W_Q^d, W_K^p, W_V^p, W_Q^p, W_K^d, and W_V^d are learnable projection matrices.

3. **Computing Cross-Attention Outputs:** The cross-attention operation for the drug-to-protein direction is computed as:

$$\mathbf{Z}_d = \text{Softmax}\left(\frac{\mathbf{Q}_d \mathbf{K}_p^T}{\sqrt{d}}\right)\mathbf{V}_p.$$

Similarly, for the protein-to-drug direction:

$$\mathbf{Z}_p = \text{Softmax}\left(\frac{\mathbf{Q}_p \mathbf{K}_d^T}{\sqrt{d}}\right)\mathbf{V}_d.$$

The dot product between $\mathbf{Q}$ and $\mathbf{K}^T$ measures similarity between feature vectors of one modality to those of the other. This setting accounts for a cross-modal attention calculation.

4. **Combining Cross-Attended Features:** After obtaining the cross-attention representations $\mathbf{Z}_d$ and $\mathbf{Z}_p$, we combine them with the original embeddings, via residual layers. Then, the final joint representation is concatenated.

$$\mathbf{Z} = [\mathbf{Z}_p; \mathbf{Z}_d].$$

5. **Classification:** The combined representation $\mathbf{Z}$ is then passed through a classification layer to predict the interaction label.

Being able to explicitly model how one modality attends to the other cross-attention stands out as a more expressive approach.

3.3 Overall Framework for In-Domain Tasks

For in-domain tasks, the model is trained and evaluated using data that share similar distributional characteristics. This can be achieved by employing standard random traintest splits. Our overall approach for in-domain experiments can be seen in Figure 3.1.

In this part of our work, we primarily focus on evaluating fusion strategies, rather than tuning models on an extensive hyperparameter set, exploring different featurization strategies, or employing advanced representation learning techniques to boost predictive performance. That's why we have opted for using straightforward and common feature representations. For proteins, we employ K-mer counting with $k = 2$, resulting in a frequency vector of length 20^2, where each entry indicates the existence of a possible amino acid pair. For drugs, we use ECFP4 fingerprints, which produce a 1024-dimensional binary vector, whose each bit indicates the presence of a particular chemical substructure.

3.3.1 Training for In-Domain Setting

We train the interaction classifier for each of the fusion methods detailed in earlier sections:

1. Concatenation (Section 3.2.1)
2. Concatenation followed with Self-attention (Section 3.2.2)
3. Cross-Attention (Section 3.2.3)

For each method, the classifier is trained using a set of standard procedures for binary classification tasks.

As in a typical binary classification setting, we employ `Sigmoid` activation function at the output layer, producing probabilistic distribution across classes. Instead of setting the threshold to a score of > 0.5 for binarization of the predictions, we opt for a strategy to find the threshold to optimize the trade-off between True Positive

Rate (tpr) and the False Positive Rate (fpr). Specifically, we set the threshold to $\max(\text{tpr} - \text{fpr})$ on the ROC curve based on the validation set.

To assess model performance, we consider the standard array of evaluation metrics for classification, including accuracy, precision, recall, F1-score, and the area under the ROC curve (AUROC).

3.4 Switching from In-Domain to Cross-Domain Setting

As previously noted, the standard random train-test splits assume distributional similarity between training and inference data, leading to over-optimistic evaluations. To verify the performance drop attributed to domain shifts, as discussed by previous researchers [2], we conducted experiments detailed in Section 3.4.2. This section details the methodology employed for introducing distributional shifts between domains and the drop in the model's performance on the target domain—data it has never encountered before.

3.4.1 Dataset for Cross-Domain Setting

To better reflect real-world scenarios where test data distribution deviates from that of training data, previous studies have employed hierarchical clustering-based methods to create distinct source and target domains [2].

Below steps outline the process of crafting these datasets:

1. **Feature Extraction for Clustering:** For molecules, ECFP4 fingerprints [20] are computed, and Jaccard distance is used to measure their similarity based on proportion of shared chemical substructures between pairs of molecules. For proteins, Position-Specific Counts (PSC) are utilized along with cosine similarity-based distances.

2. **Hierarchical Clustering:** Standard hierarchical agglomerative clustering have been applied separately for proteins and drugs, where the most similar cluster pairs (measured via single-linkage method, setting the most similar data points

between clusters as the inter-cluster distance) are merged gradually. This process inherently results in an increase in the inter-cluster distance at each step of merging. The merging is pruned when the inter-cluster distance reaches a pre-defined threshold, marking the point where clusters are separate enough to craft source and target datasets.

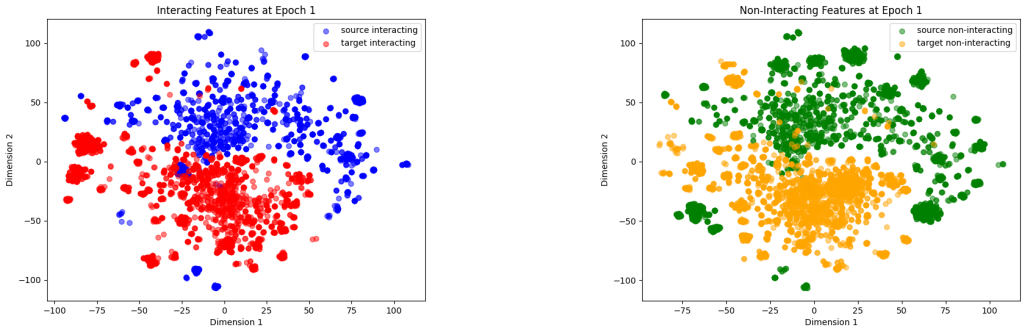
3. **Source-Target Split:** After clustering, 60% of the protein and drug clusters are selected as the source domain. While the pairs from these source clusters constitute the source dataset, the remaining clusters form the target. The source dataset is used for training, while the target dataset is split so that 80% can be used as additional training data for the feature extraction, and 20% serve as a held-out test set.

This clustering-based approach yields approximately 2,800 drug clusters and 1,700 protein clusters, with dissimilar instances by the principle of separation. This constitutes a suitable setting for a domain adaptation scenario, with source and target datasets.

To further investigate the data distribution across domains, we performed a preliminary analysis. Key findings include:

- **Distinct Domains:** There are no common proteins, drugs, or (SMILES, Protein) pairs between the source and target datasets.
- **Within-Domain Overlap:** While overlaps are observed between training and validation splits within each domain (as expected in a random split), no such overlap exists across domains.

To visualize the discrepancy between domains, we employed t-SNE on the fused features of drugs and proteins. For input featurization, we utilized k-mer counts with $k = 2$ for proteins and ECFP4 fingerprints for drugs. And, for input fusion, we used cross attention. The visualizations were generated using embeddings from the first epoch, before any domain alignment is introduced. We chose to display the domains on a per-class basis to enhance interpretability. Specifically, Figure 3.6a illustrates the interacting pairs, with blue points representing source data and red points representing



(a) Interacting

(b) Non-interacting

Figure 3.6: Comparison of interacting and non-interacting classes over epochs.

target data. Figure 3.6b depicts the non-interacting pairs, where green points denote source data and yellow points correspond to target data. These figures effectively highlight the distributional discrepancies between domains for both classes. To be able to monitor the alignment process better, we have visualized the extracted features at certain epochs during training too (Presented in Section 3.5.11;).

3.4.2 Cross-Domain Performance without Domain Adaptation

To evaluate the baseline performance in a cross-domain scenario without any domain adaptation techniques, we train models exclusively on the source domain data and then test them on the target domain. This allows us to quantify the degree of performance degradation when the model encounters data from a distinctly different distribution than that of the training set.

In this setup, the dataset configuration is as follows:

- **Training & Validation Data:** Source domain is randomly split to training/validation (90/10 %). This results in 13440 and 1504 data instances for training and validation, respectively.
- **Test Data:** Target domain data, held out and not used during training, contains 8896 data points.

This does not yield an optimal validation strategy, as our validation data does not exhibit similar statistical characteristics to those of the test data. In effect, it enables us to highlight the discrepancy between validation performance and the actual performance on the target test set.

We employ the same hyperparameter configuration that produced the best results in the in-domain setting for training the models. Figure 3.7 illustrates the training and validation curves on the source domain. The left panel shows the loss values over epochs, while the right panel depicts various metrics (e.g., accuracy, precision, recall, F1-score, AUROC) on the validation set. These curves suggest stable training and decent validation performance with the source data.

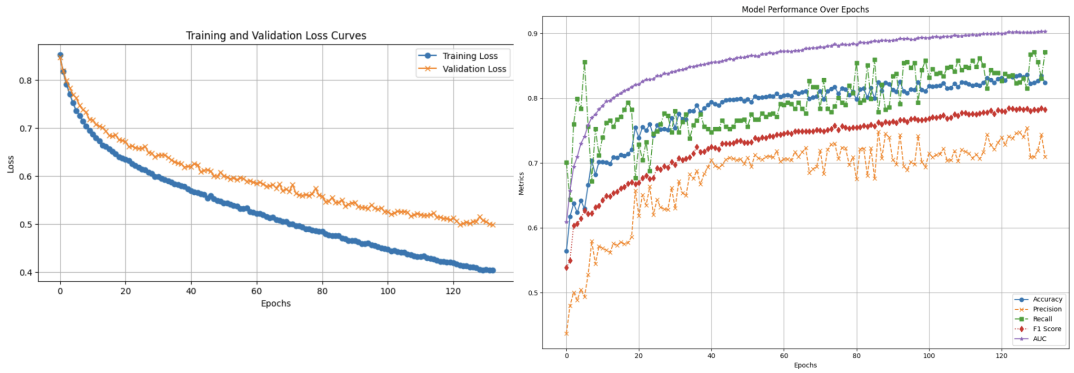


Figure 3.7: Training and validation performance on the source domain (no domain adaptation).

Though Figure 3.7 exhibits very promising learning curves, the trained model's performance on the test set is far from ideal. In fact, its performance is at random, as illustrated in Table 3.1. Here, the performance of the same model in the in-domain setting and the cross-domain setting is compared:

Table 3.1: Performance Metrics Across Different Settings

Setting	**Accuracy**	**Precision**	**Recall**	**F1**	**AUROC**
In-Domain	0.86	0.83	0.83	0.84	0.94
Cross-Domain	0.54	0.51	0.68	0.58	0.58

There is a dramatic drop in performance when switching from in-domain to cross-domain tasks. While in-domain performance appears impressive, cross-domain performance is no better than random. This underscores the impact of domain discrepancies on model performance and the need for robust domain adaptation.

Notably, the drop in precision is more pronounced than in recall, indicating a tendency for models to classify instances as positive, despite there being no class imbalance between positive and negative samples.

3.5 Framework for Cross-Domain Setting

This section outlines the domain adaptation approaches employed. Initially, we discuss the distributions targeted for alignment, followed by a high-level overview of the methods used for domain alignment, specifically MMD Loss and Adversarial Training. Subsequently, we describe the data loading pipeline. These are followed with the detailed explanations of our approaches. Then, we conclude with hyperparameter tuning, presenting the intermediate performance results, including loss curves, metrics on the validation set, and distributional alignment.

3.5.1 Aligning Marginal vs. Conditional Distributions

A key design choice in domain adaptation is whether to align marginal distributions or conditional distributions. Marginal alignment attempts to match the overall feature distributions $P(X_{source})$ and $P(X_{target})$ without considering the labels. In contrast, conditional alignment attempts to align distributions conditioned on the interaction label, i.e., $P(X|Y = 0)$ and $P(X|Y = 1)$ across domains.

In this work, we implement both strategies to compare their effectiveness.

For marginal alignment, we attempted to minimize the distributional discrepancy between the source and target data regardless of their class label.

For conditional alignment, we use class labels to align data from different domains, focusing separately on interacting ($Y = 1$) and non-interacting ($Y = 0$) pairs.

Specifically, for non-interacting pairs, we aim to minimize the divergence between $P(X_{\text{source}}|Y=0)$ and $P(X_{\text{target}}|Y=0)$. Similarly, for interacting pairs, our methods target minimizing the divergence between $P(X_{\text{source}}|Y=1)$ and $P(X_{\text{target}}|Y=1)$.

3.5.2 Maximum Mean Discrepancy (MMD Loss)

One method employed in the literature for feature alignment is using an auxiliary loss function to minimize the distributional discrepancy between domains. For this, Maximum Mean Discrepancy (MMD) Loss is commonly used, providing a statistical measure of the discrepancy between source and target distributions $P(X_{\text{source}})$ and $P(X_{\text{target}})$ by leveraging embeddings in a Reproducing Kernel Hilbert Space (RKHS).[1] By minimizing MMD, we aim to reduce the domain shift and learn more domain-invariant representations.

The core concept of MMD is to embed source and target samples into a Reproducing Kernel Hilbert Space (RKHS) via a kernel function, such as the Gaussian or polynomial kernel, and then compare the mean embeddings of these distributions. By minimizing the distance between these mean embeddings, MMD effectively drives the model to learn domain-invariant features. In contrast to many other approaches, MMD does not assume a specific parametric form of the distributions (such as when using the KL divergence) and the kernel trick allows it to capture complex, high-dimensional relationships without explicitly computing the mapping into a large feature space.

An RKHS is a type of function space that simplifies data point comparisons using a kernel function $k(\cdot,\cdot)$. Instead of dealing with a large or infinite-dimensional feature space, we compute pairwise similarities $k(x,x')$ between points in the original input space. This indirect mapping offers a practical method to compare and align probability distributions. In MMD, these kernel-based similarities help measure the closeness of source and target distributions without directly managing high-dimensional feature embeddings.

[1] RKHS is primarily favored because it allows for a flexible, kernel-based measurement of distributional differences without the need to assume any specific parametric form.

3.5.2.1 MMD Loss Function for Marginal Alignment

Let $\{x_s^i\}_{i=1}^{n_s}$ be samples from the source domain and $\{x_t^j\}_{j=1}^{n_t}$ be samples from the target domain. We consider a kernel function $k(\cdot,\cdot)$ defined on the input feature space. The MMD between $P(X_{\text{source}})$ and $P(X_{\text{target}})$ is computed as

$$\text{MMD}^2(P,Q) = \frac{1}{n_s^2}\sum_{i=1}^{n_s}\sum_{i'=1}^{n_s} k(x_s^i, x_s^{i'}) + \frac{1}{n_t^2}\sum_{j=1}^{n_t}\sum_{j'=1}^{n_t} k(x_t^j, x_t^{j'}) - \frac{2}{n_s n_t}\sum_{i=1}^{n_s}\sum_{j=1}^{n_t} k(x_s^i, x_t^j).$$

The first two terms in the equation calculate the average intra-domain similarity, while the last one calculates that for inter-domain. The overall loss calculates the difference between intra- and inter-domain distances. Minimizing this quantity encourages the distributions of different domains to become similar.

3.5.2.2 MMD Loss Function for Conditional Alignment

To incorporate label information, MMD loss can be computed separately for each class and then averaged over all classes. This method is formalized as follows:

$$\text{MMD}_{\text{cond}} = \frac{1}{|C|}\sum_{c\in C} \text{MMD}^2\big(P(X_s^c), Q(X_t^c)\big).$$

where $P(X_s^c)$ and $Q(X_t^c)$ represent the probability distributions of the source and target domain data for class c, respectively.

3.5.2.3 Using MMD as an Auxiliary Regularizer Loss

The MMD loss is added to the classification loss with a weight given by a hyperparameter λ for training. This way, MMD Loss acts as a regularizer term in the overall loss function, which is given by:

$$\mathcal{L} = \mathcal{L}_{\text{classification}} + \lambda \cdot \mathcal{L}_{\text{MMD}},$$

where:

- $\mathcal{L}$: Total loss.
- $\mathcal{L}_{\text{classification}}$: the classification loss, measured based on the classification performance of the model.
- $\mathcal{L}_{\text{MMD}}$: the Maximum Mean Discrepancy loss, measured by the discrepancy between feature distributions of domains.

By tuning λ and the kernel parameters, we can control both the strength and nature of the domain alignment.

3.5.3 Adversarial Domain Adaptation

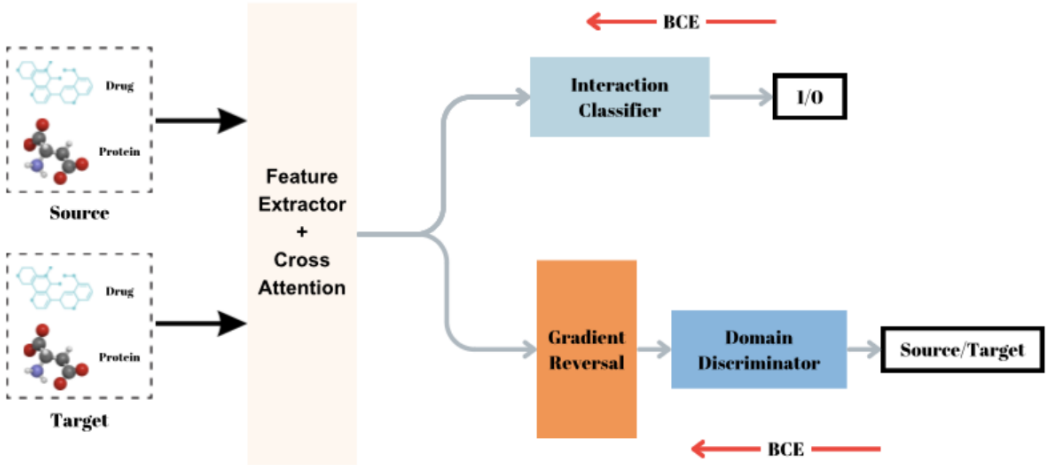


Figure 3.8: Adversarial Model Architecture for Cross-Domain Settings. The feature extractor receives input from both domains and passes fused representations to both the interaction classifier and the domain discriminator. The gradient reversal layer inverts gradients from the domain discriminator, enforcing domain-invariant feature learning.

In addition to MMD-based alignment, a common approach is to utilize an adversarial objective, adopting an auxiliary discriminator network to determine whether the extracted features originate from the source or target domain. The adversarial objec-

tive is characterized by a min-max game, where the domain discriminator attempts to classify the domain of the features, while the feature extractor is trained to produce features that are indistinguishable for the discriminator, as depicted in Figure 3.8. The objective here is very similar to Generative Adversarial Networks (GANs).

The aim is to learn a domain-invariant feature extractor using an adversarial setup. Concurrently, we train the primary interaction classifier exclusively on source data, which will later be evaluated on the target dataset to assess zero-shot performance in an unsupervised domain adaptation context.

Although this setup is similar to GANs, instead of using a typical GAN loss, we employ a Gradient Reversal Layer (GRL). In this setup, the extracted features are forwarded to both the interaction classifier and the domain discriminator. Here, the Gradient Reversal Layer acts as an identity function during the forward pass. In the backward pass, however, it flips the calculated gradients in the opposite direction, setting an environment for adversarial training. That is, during backpropagation, the reversed gradients will cause updates to the feature extractor in a manner that makes its output harder to distinguish for the domain discriminator.

This way we learn a domain-invariant feature extractor that is still useful for the interaction classifier.

3.5.3.1 Marginal Alignment using a Domain Discriminator

A typical approach, inspired by Domain-Adversarial Neural Networks (DANN) [36], focuses on marginal alignment by directly aligning $P(X_{\text{source}})$ and $P(X_{\text{target}})$. The domain discriminator receives raw features and tries to distinguish source from target, while the reversed gradients encourage the feature extractor to produce domain-invariant representations.

3.5.3.2 Conditional Alignment using CDAN

Conditional Domain Adversarial Networks (CDAN) [5] enhance adversarial alignment by leveraging label information. Instead of passing raw features directly to the

domain discriminator, CDAN combines features with class predictions to form a joint representation. The joint representation, which captures both feature information and label structure, is then passed through the Gradient Reversal Layer before reaching the domain discriminator. This process enforces alignment of conditional distributions $P(X_{\text{source}} \mid Y = c)$ and $P(X_{\text{target}} \mid Y = c)$ for each class c, rather than merely matching marginal distributions.

3.5.4 Data Loading Pipeline with Multiple Domains

3.5.4.1 Multi-Domain Data Pipeline and Loading Modes

We randomly partition 20% of both the source and target training datasets to serve as validation sets, allowing us to monitor the model's performance in each domain during training. This approach establishes training and validation data loaders for each domain, in addition to a test data loader for the target domain. To facilitate cross-domain training, we employ a custom `MultiDataLoader` class capable of iterating over batches from both domains concurrently within an epoch.

Using a multi-dataloader in domain adaptation often reintroduces the challenge of class imbalance, particularly when one domain has substantially more data than the other. In our case, the source dataset contains about 15,000 data instances, whereas the target has approximately 7,000. Consequently, one domain's data runs out before the other's within an epoch. To address this, we define multiple loading modes:

- **same_size:** Truncates the larger dataset to match the number of batches in the smaller one, ensuring balanced exposure to source and target data during training. Although easy to implement, it wastes a portion of the annotated data.
- **cycle_smaller:** Cycles through the smaller dataset repeatedly until the larger dataset is fully consumed within an epoch. This prevents underrepresentation of the smaller domain but risks overfitting to the target data if it is much smaller.
- **sample_larger:** At each iteration, randomly samples a subset from the larger dataset to match the smaller dataset's size. This allows usage of all annotated

data without a strict truncation while avoiding imbalance. But, this comes at a cost of instability in training.

In practice, each loading mode is a trade-off. Hence, we treat the choice of mode as another hyperparameter, comparing their effects in an array of experiments.

Sequential vs. Random Ordering of Batches. Most existing domain adaptation methods utilize pipelines that process batches from source and target domains sequentially. This method ensures a consistent order and size of data from each domain within an epoch. While this approach simplifies batch-wise operations (e.g., batch normalization, loss computation), it may bias the model toward a particular ordering or a fixed sample distribution per domain.

As an alternative, randomly interleaving source and target batches could reduce potential biases associated with fixed ordering. However, since losses for the target and source batches need to be calculated separately to maintain the a zero-shot scenario in unsupervised domain adaptation, this method complicates data-loading logistics and interrupts standard batch processing routines. Given that sequential processing is the standard method in existing literature, we adhere to this convention. Yet, we acknowledge the potential benefits of exploring a random mixing strategy in future research.

High-Level Data Pipeline Description. Below is an outline of the steps we follow when preparing and loading data:

1. **Load Source and Target Datasets:** We use either classical embeddings (e.g., ECFP4, k-mer) or advanced pretrained model embeddings (e.g., ChemBERTa, ProtT5XL), with the option to fine-tune them.

2. **Split Data into Train/Validation/Test:** Both source and target datasets are divided into training and validation subsets to monitor domain-specific performance. The target data are used only to train the feature extractor, accommodating a 0-shot performance evaluation setting on the target domain.

3. **(Optional) Compute Global Statistics for Normalization:** If normalization is needed (especially for pretrained embeddings), we combine the source and target training subsets to compute global means and standard deviations.

4. **(Optional) Normalize Datasets:** Apply computed statistics across all data splits if normalization is enabled.

5. **Assemble MultiDomain Loaders:** Build combined train and validation loaders via the `MultiDataLoader`, specifying how batches from the two domains are drawn (based on one of the modes above).

3.5.4.2 Data Normalization

We do not apply normalization to classical embeddings such as ECFP4 because each bit in these vectors indicates the presence or absence of a chemical substructure, and z-score scaling would compromise interpretability.

For embeddings derived from pretrained models (e.g., ChemBERTa [21] or ProtT5XL [12]), we treat normalization as a hyperparameter. To avoid data leakage, if normalization is enabled, we compute the mean and standard deviation exclusively from the combined source and target training subsets; we then apply these statistics to all data splits for consistent scaling.

3.5.5 Framework for Domain Adaptation in DTI utilizing MMD Loss

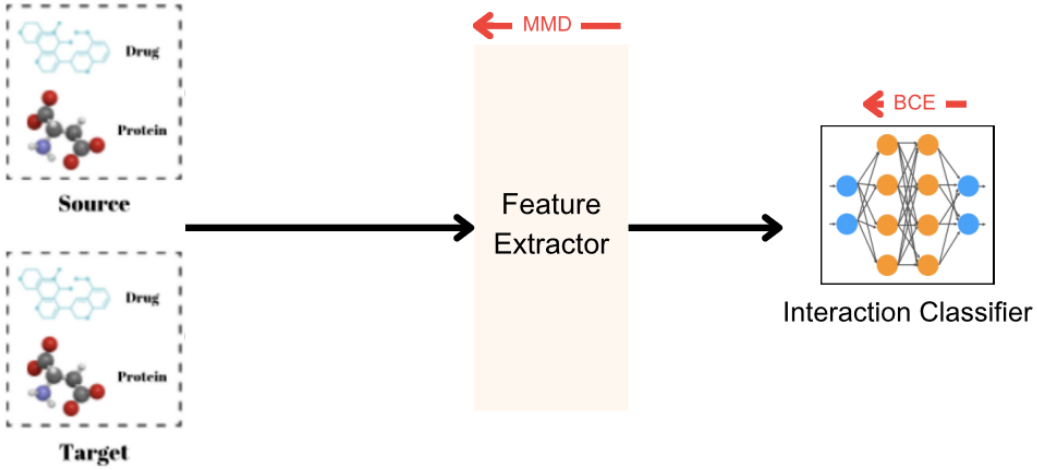


Figure 3.9: Overall cross-domain framework utilizing MMD Loss.

We incorporate MMD loss as a regularization term in the training objective, alongside the primary interaction task, as depicted in Figure 3.9. For the interaction classification task, we employ Binary Cross Entropy (BCE) Loss, as in any typical standard binary classification settings. Concurrently, MMD Loss is calculated on the features extracted from the source and target data to facilitate domain alignment. Both MMD and BCE Losses are utilized to update the feature extractor, aiming at a domain-invariant feature extractor. For the interaction classifier, however, we only use the error signal from the interaction prediction on the target data.

As explained above, two approaches to alignment are possible: marginal and conditional alignment. The final loss function is based on the choice of conditional or marginal alignment, as shown below:

1. **Loss Function for Marginal Alignment:**

$$\mathcal{L} = \mathcal{L}_{\text{interaction}}(X_{source}, Y_{source}) + \lambda \cdot \mathcal{L}_{\text{MMD}}(F(X_{source}), F(X_{target}))$$

2. **Loss Function For Conditional Alignment:**

$$\mathcal{L} = \mathcal{L}_{\text{interaction}}(X_{source}, Y_{source}) + \lambda \cdot \mathcal{L}_{\text{MMD}}(F(X_{source}|Y), F(X_{target}|Y))$$

where $\mathcal{L}_{\text{interaction}}$ is the binary cross-entropy loss measured on the source domain's labeled pairs, and $\mathcal{L}_{\text{MMD}}$ measures the distributional discrepancy between source and target features. The parameter λ controls the strength of the MMD alignment relative to the interaction task.

3.5.5.1 MMD Training Steps

1. **Feature Extraction:** Given a batch of source and target samples, we pass them through the feature extractor to obtain source features $F(X_{\text{source}})$ and target features $F(X_{\text{target}})$.

2. **Interaction Loss Computation:** We feed only the source features to the interaction classifier, computing the loss in terms of binary cross entropy. The interaction predictions for the target data, on the other hand, are used only for performance-monitoring purposes (i.e., no gradient flows from target labels to the interaction classifier). This setting accounts for a strictly zero-shot paradigm for target-domain interaction classification.

3. **MMD Loss Computation:** The MMD Loss is calculated based on the choice of domain alignment. If marginal alignment is targeted, we measure the distance between $F(X_{\text{source}})$ and $F(X_{\text{target}})$. If conditional MMD is employed, we compute the loss class-wise, i.e., aligning $P\big(F(X) \,\big|\, Y = 0\big)$ and $P\big(F(X) \,\big|\, Y = 1\big)$ across the source and target domains.

4. **Backpropagation and λ-Scheduling:** First, the source interaction loss is backpropagated, updating both the feature extractor and the interaction classifier. Subsequently, the MMD loss is backpropagated only to the feature extractor, ensuring that the classifier receives no gradient from the target data.

   This setting comes with another hyperparameter λ to balance the influence of domain alignment relative to classification. Instead of directly treating λ as another hyperparameter to tune, we employ a scheduling algorithm. Given the feature extractor and interaction classifier networks are being trained from scratch, we chose to introduce a warm-up stage to let the models focus on the main task. After the warmup period ends, we gradually increase λ to strengthen domain alignment. Here, we hold that the feature extractor will generate better

embeddings for the interaction classifier to classify and introducing a gradual complexity for domain alignment seem logical.

$$p(\text{epoch}, \text{epochs}) = \frac{\text{epoch}}{\text{epochs}},$$

$$\lambda(\text{epoch}, \text{epochs}) = \max\left(\frac{2}{1 + e^{-10 \cdot p(\text{epoch},\text{epochs})}}, 0.01\right),$$

This scheduling helps keep alignment pressure low initially, then steadily increases it as the model stabilizes.

5. **Optimization and Early Stopping:** We use an early-stopping criterion based on validation metrics, monitoring performance on both source and target data to avoid overfitting and ensure a stable alignment process. The final model is selected from the epoch yielding the best classification performance on target-domain performance in terms of MCC.

3.5.6 Framework for Domain Adaptation in DTI utilizing Adversarial Setup

Figure 3.8, above, provides an overview of the adversarial training framework in the cross-domain setting. In this architecture, drug-protein pairs from both source and target domains are fed into the feature extractor concurrently. The feature extractor can utilize classical statistical representations (e.g., k-mer frequencies for proteins, ECFP4 fingerprints for drugs) or more advanced embeddings (utilizing pre-trained models, such as ChemBERTa and ProtT5XL). After extracting features for both modalities, we fuse them using one of the previously described fusion methods: simple concatenation, self-attention on concatenated features, or cross-attention.

The fused representation is then passed to two separate components:

1. **Interaction Classifier:** A binary classifier that predicts whether the drug-protein pair interacts.

2. **Domain Discriminator:** A binary classifier that predicts which domain (source or target) the input pair is from.

Both the interaction classifier and domain discriminator produce binary outputs and hence are trained with Binary Cross-Entropy (BCE) losses. Consequently, we end up with four losses, outlined below:

- **Source Interaction Loss:** Computed from source domain to train the interaction classifier.
- **Target Interaction Loss:** Computed from target domain examples for monitoring the performance on the target, but not backpropagated to avoid supervision from target interaction to test 0-shot performance of the classifier on the target domain.
- **Source Domain Loss:** Domain classification loss for source samples.
- **Target Domain Loss:** Domain classification loss for target samples.

In practice, we only backpropagate the source interaction loss, alongside the domain losses. The target interaction loss is recorded merely for evaluation and tracking progress.

The gradient flow through the network is structured as follows:

1. **Interaction Loss Backpropagation:** Calculated solely on source data and used for supervision to the interaction classifier and domain discriminator, serving for the primary DTI prediction task.
2. **Domain Loss Backpropagation:** The discriminator is updated based on the domain discriminator loss, ensuring better discrimination in the next epoch. However, these are passed through the gradient reversal layer (GRL), before arriving at the feature extractor. In the GRL, the calculated gradients are flipped in the opposite direction in an attempt to fool the discriminator in the next iteration of training. That is, such reversed updated will result in an embedding that will be actually harder for the discriminator to distinguish. In short, the reversed gradients work for the adversarial objective in that the feature extractor is guided to fool the discriminator, by aligning the distributions of source and target data.

This joint training setup guides the feature extractor to learn embeddings that are indistinguishable across domains, but still useful for the main interaction task.

3.5.7 Training with DDAN vs CDAN

For the adversarial domain adaptation approach, we employ both Domain-Adversarial Neural Networks (DANN)[36] and Conditional Domain Adversarial Networks (CDAN) [5]. The training procedure is outlined below:

Key steps in the CDAN training loop:

1. **Feature Extraction:** Extract the features of both domains, $F(X_{\text{source}})$ and $F(X_{\text{target}})$, using the featurization method and one of the fusion methods discussed.

2. **Get the Source Interaction Prediction & Compute Interaction Loss:** Get the interaction predictions and losses for both domains in terms of Binary Cross Entropy. Those for the target domain is merely calculated for performance-monitoring purposes and not backpropagated.

3. **Input for the Discriminator:** Based on the alignment opted for (i.e., aligning marginal or conditional distributions), the input for the domain discriminator is constructed. In particular, if marginal alignment is targeted, the constructed features are directly forwarded to the domain discriminator. In contrast, if we are aiming for a conditional alignment, we employ the outer product to create a joint representation of feature embeddings and class probabilities. The outer product between the feature embedding vector and the class probability vector results in a matrix that captures the correlations between each feature and each potential class outcome. This matrix representation enriches the input to the discriminator with class-specific contextual information, encouraging conditional alignment (i.e., aligning $P(X|Y = c)$ across domains) rather than just marginal alignment.

4. **Domain Discrimination and Gradient Reversal:** The obtained representation (either direct embeddings or combined with class predictions for conditioning)

is then forwarded to the domain discriminator, passing through the gradient reversal layer (GRL). In the forward pass, the GRL acts as an identity function. The domain discriminator predicts the domains (either as source or target), resulting in a loss measured in terms of binary cross-entropy. The domain discriminator's parameters are updated based on the gradients calculated from this loss, ensuring better predictions in the next epoch. The gradients, upon passing through the gradient reversal layer, are flipped in the opposite direction when flowing back to the feature extractor. This causes the parameters of the feature extractor to be updated in an opposite manner, i.e., producing embeddings that will be harder for the domain discriminator to distinguish. This setup accounts for the adversarial training and the learning of a domain-invariant feature extractor.

5. **Loss Functions:** The CDAN/DDAN training optimizes two main losses:

   - **Interaction Loss (source only):** Measures the error on the primary task of interaction classification in terms of binary cross entropy. Though calculated for both domains for performance monitoring purposes, only the supervision from the source data is used for updating the model's (interaction classifier's) parameters.
   - **Domain Loss (source + target):** Measures the domain classifier's performance in terms of binary cross entropy. The gradient reversal layer, flipping the calculated gradients based on this loss, ensures an adversarial training objective, encouraging the feature extractor to become more domain-invariant.

6. **Warm-up and Scheduling:** Since all networks are trained from scratch, we introduce a warm-up stage where only the feature extractor and interaction classifier are trained, while the domain discriminator is temporarily frozen. This stage allows the model to focus on the main task, developing an initial understanding of which features are most relevant for interaction prediction. The adversarial stage introduces another hyperparameter α that determines the magnitude of the reversed gradient vectors, thereby controlling the power of the adversarial objective. To determine this value, we experimented with both constant α values and different scheduling strategies. Specifically, our trials

included using a constant value and logarithmic scheduling, and exponential scheduling. Though we did not perform an extensive evaluation of these strategies to determine the value of α, based on the observed performance of both the discriminator and the source/target classifiers on the preliminary trials, exponential scheduling proved to be superior. This approach gradually increases α as training progresses. Such a strategy is viable because, as training proceeds, both the feature extractor and domain discriminator become stronger and, consequently, require a progressively stronger regularization (in the form of reversed gradients) to effectively align the domains.

3.5.8 Evaluation and Hyperparameter Tuning for Cross-Domain Settings

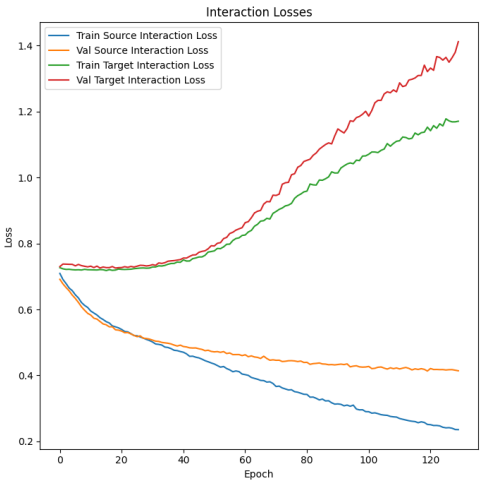
3.5.9 Choosing Between Marginal and Conditional Alignment

As discussed earlier, a key decision in a domain adaptation setup is whether to align the marginal or the conditional distributions. Although our initial premise favored conditional alignment—since aligning features across interacting and non-interacting classes seemed counterintuitive—we explored both approaches for completeness.

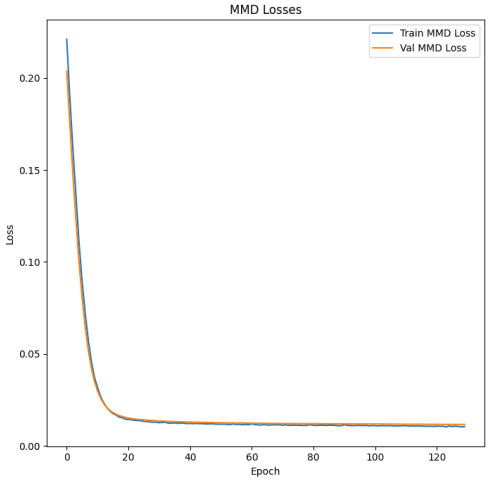
In this preliminary phase, we did not carry out an extensive hyperparameter search. Instead, we performed a series of experiments to prune the search space and exclude the suboptimal alignment type (i.e., conditional or marginal). We trained several models with different hyperparameter configurations, using both adversarial and MMD losses for marginal alignment. We present our findings employing marginal alignment in the following paragraphs.

Using MMD Loss for Marginal Alignment: Figure 3.10 illustrates the training loss curves employing marginal alignment with Maximum Mean Discrepancy (MMD) Loss [3]. Specifically, Figure 3.10a shows the evolution of interaction performance (in terms of source and target interaction losses for both training and validation sets) over the course of training, while Figure 3.10b tracks the changes in the MMD loss across both training and validation datasets. Notably, the MMD loss consistently decreases during training for both training and validation datasets. However, while

interaction prediction performance converges for the source data, no similar improvement is observed in the target data, as indicated by increasing loss curves (depicted in red and green). This suggests that despite the visible convergence of marginal distribution alignment (evidenced by the decreasing MMD loss), it does not translate into improved interaction prediction performance on the target data.



(a) Interaction loss curves over epochs, showing Train Source Interaction Loss (blue), Validation Source Interaction Loss (orange), Train Target Interaction Loss (green), and Validation Target Interaction Loss (red).



(b) MMD loss curves for training (blue) and validation (orange) sets over epochs.

Figure 3.10: Change in performance in terms of losses during training under marginal alignment with MMD.

Using Domain-Adversarial Neural Networks (DDAN) with Marginal Alignment: Figure 3.11 shows how the performance (accuracy, precision, recall, F1 score, and area under the curve) changes on the source (blue) and target (orange) validation sets over epochs when an adversarial objective with marginal alignment is employed with DDAN[36].

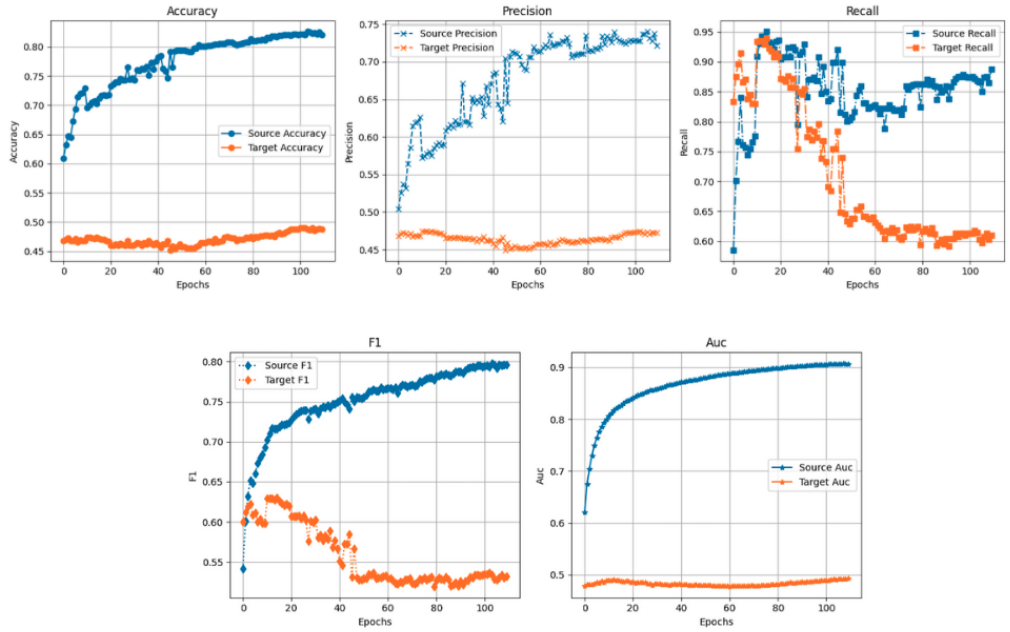


Figure 3.11: Performance over epochs on the source (blue) and target (orange) validation sets in terms of accuracy, precision, recall, F1 score, and AUC.

All these plots reveal a discrepancy between the source and target performance across all metrics, indicating that the model overfits to the source data.

We also monitored the validation domain loss to assess the performance of the discriminator. Its value remains close to that of a random classifier (BCE ≈ 0.65)[2] , which is the desired behavior for adversarial domain discrimination. However, the interaction classification performance on target data is far from ideal.

To gain additional insights, we monitored the discriminator's performance, which we ideally expect to converge at that of a random predictor (i.e., 0.5 accuracy). Figure 3.12 shows the domain discriminator's performance in terms of F1 Score, MCC, and Accuracy. We do not observe a clear convergence to random performance. Around the 40th epoch, the discriminator's performance is closest to random, but this does not correspond to any improvement in interaction prediction (see Figure 3.11).

[2] A BCE value of around 0.65 indicates near-random classifier performance. Using the formula BCE $= -y\log(p) - (1-y)\log(1-p)$ with $p = 0.5$, we find BCE $= \log(2) \approx 0.693$.

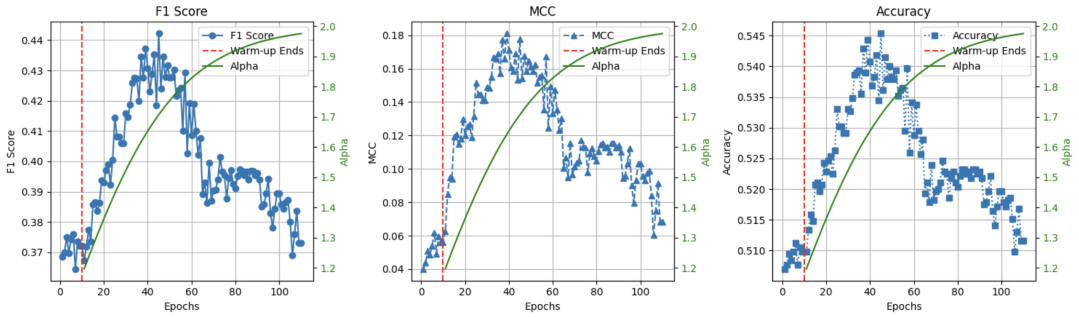


Figure 3.12: Domain discriminator performance over epochs in terms of F1 Score, MCC, and Accuracy. The red dotted line indicates where the warm-up period ends, and the green curve indicates the α value.

In a further attempt gain insight regarding the domain discriminator's performance, we tracked the number of instances classified as source or target over training. Figure 3.13 plots these counts: the left subplot shows the number of instances labeled as source, and the right one shows those labeled as target. In both, the red dotted line indicates the end of the warm-up period, while the green curve depicts the change in the parameter α (which controls the gradient reversal scale). We observe a pronounced bias toward classifying instances as source, even from the beginning.

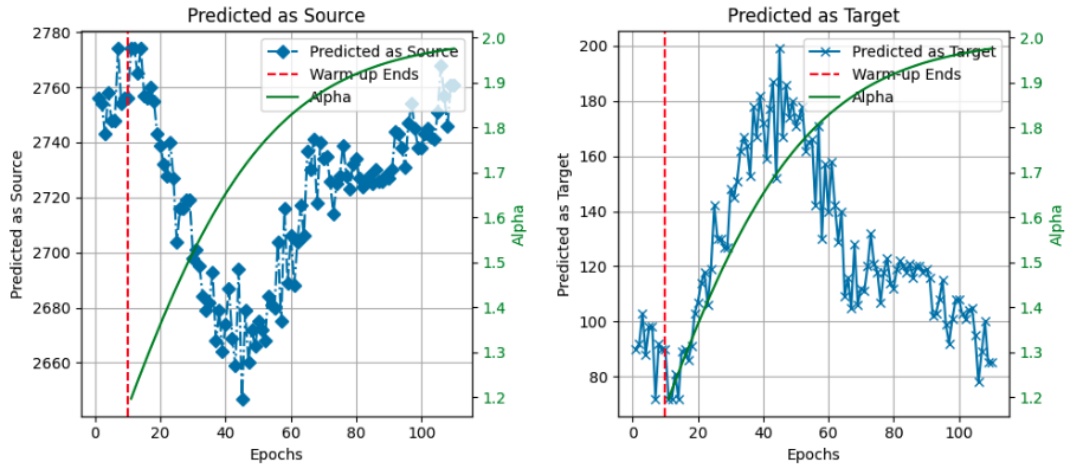


Figure 3.13: Evolution of the domain discriminator's predictions over training for DANN with marginal alignment. (Left) Number of instances predicted as source. (Right) Number of instances predicted as target. The red dotted line marks the end of the warm-up period, and the green curve shows the α scheduling. A strong bias toward predicting source is clearly visible.

Despite efforts such as hyperparameter tuning, class weighting, and other mitigation strategies, we were unable to correct these learning curves or improve target classification performance under marginal alignment. Consequently, **we decided to adopt conditional alignment for the remainder of this work**, consistent with our initial premise.

3.5.10 Hyperparameter Tuning Methodology

In this section, we describe our strategy for exploring various hyperparameters and design choices for both CDAN- and CMMD (Conditional MMD)-based domain adaptation. We focus on two main factors: loader modes and decision threshold calibration, which we treat as hyperparameters alongside the network-specific parameters. Due to resource constraints, we do not conduct an exhaustive grid search; instead, we rely on a randomized selection of hyperparameter configurations within reasonable ranges.

3.5.10.1 Loader Modes

As explained in Section 3.5.4, we employ a custom `MultiDataLoader` to handle batches from both source and target domains within one training loop. However, the source dataset (approximately 15,000 samples) is more than twice the size of the target dataset (about 7,000 samples), leading to class imbalance or under-/over-sampling issues depending on how we iterate over data (within one epoch).

To address this, we define three loading modes, each with its pros and cons (See Section 3.5.4):

- **same_size:** Truncate the larger dataset to match the batch count of the smaller one.
- **cycle_smaller:** Repeatedly cycle through the smaller dataset until the larger dataset is fully consumed in an epoch.
- **sample_larger:** Randomly sample subsets from the larger dataset to match the size of the smaller dataset in each iteration.

As we believed the loading mode might have a substantial effect on the overall model performance, we experimented all these modes with each hyperparameter configuration.

3.5.10.2 Hyperparameter Tuning for Conditional MMD-Based Training

We define the following search space for hyperparameter optimization for MMD-based DTI Prediction:

```
param_space = {
    'batch_size': [32, 64, 128, 256],
    'normalize': [True, False],
    'lr_fe': [1.0e-4, 1.0e-5, 1.0e-6],
    'lr_cls': [1.0e-3, 1.0e-4, 1.0e-5],
    'warmup_epochs': [0, 5],
```

```
    'bigger_classifier': [True, False],
    'classifier_dropout': [0.3, 0.5],
    'classifier_hidden_dim': [128, 256],
    'kernel_type': ['rbf', 'laplacian', 'linear', 'polynomial']
}
```

Key Parameters

- **Kernel Choice:** We experiment with four kernel types: RBF, Laplacian, Linear, and Polynomial.
- **Separate Optimizers & Learning Rate:** Only feature extractor and classifier networks are trained, each with its optimizer and learning rate.

We again rely on random sampling. The selected configurations are trained and evaluated with each of the loader modes, and results are saved to identify overall trends.

3.5.10.3 Hyperparameter Tuning for CDAN-Based Training

CDAN introduces adversarial domain alignment through a domain discriminator, feature extractor, and an interaction classifier that are jointly trained. We define the following hyperparameter search space for this setting:

```
param_space = {
    'batch_size': [32, 64, 128, 256],
    'normalize': [True, False],
    'lr_fe': [1.0e-4, 1.0e-5, 1.0e-6],
    'lr_cls': [1.0e-3, 1.0e-4, 1.0e-5],
    'lr_disc': [1.0e-3, 1.0e-4, 1.0e-5],
    'warmup_epochs': [0, 5, 20],
    'bigger_discrim': [True, False],
    'bigger_classifier': [True, False],
    'classifier_dropout': [0.3, 0.5],
```

```
    'classifier_hidden_dim': [128, 256],
}
```

Calibration of the Decision Threshold: In binary interaction classification, a threshold is required to convert predicted probabilities into 0, 1 labels. Rather than setting this threshold directly at 0.5, we employ a calibration strategy that sets it to the point maximizing the $tpr - fpr$ metric, which optimizes the trade-off between true and false positive rates using the validation set.

Key Components.

- **Learning Rates:** Separate optimizers for the feature extractor, interaction classifier, and domain discriminator, each with its own learning rate (`lr_fe`, `lr_cls`, `lr_disc`).

- **Warmup Epochs:** During warmup, we disable the domain alignment loss to let the network focus on the primary interaction-classification objective. We test {0, 5, 20} warmup epochs. (We include no warmup strategy into our experiments).

- **Classifier and Discriminator Capacity:** The complexities of networks, especially those of the discriminator and feature extractor, have a significant impact on overall performance in such adversarial settings.

Given the large number of possible combinations, we employ **random sampling** of 25 configurations from the above space. For each configuration, we train the model under all three modes(Section 3.5.10.1) to assess the direct effect of the data-loading strategy. This leads to 100 model evaluations.

The model performance results (employing CDAN) on validation sets are presented under Section 4.2.

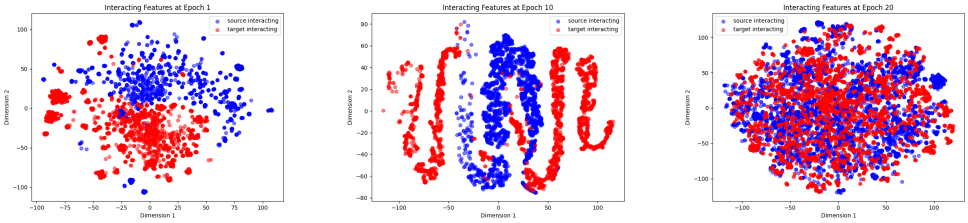
3.5.11 Feature Alignment Visualization

We visually inspected how well the learned feature representations are aligned between the source and target domains using t-SNE projection of the extracted features. Specifically, we periodically record intermediate feature embeddings from our model (e.g., at epoch 1, epoch 10, and epoch 20), apply t-SNE to reduce the dimensionality to two components, and then plot source and target samples.

Figures 3.14 and 3.15 illustrate the resulting 2D projections for interacting and non-interacting pairs, respectively. Examining these plots over epochs, we observe that although the feature distributions are initially clustered in a domain-specific manner, the source and target samples become increasingly intertwined as training progresses, indicating improved domain alignment.

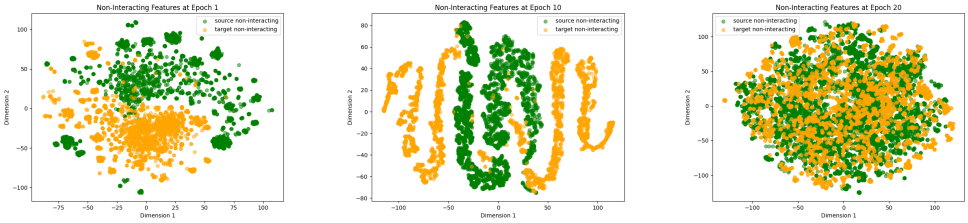
For this, we used the ECFP4 fingerprints for drugs and k-mer representations of proteins, as they provide easier embeddings, with characteristics easier to capture via dimensionality reduction methods.

Implementation Details. We implement a custom `Visualizer` class that (1) collects feature embeddings and label information for both domains, (2) uses t-SNE (with `n_components=2` and a fixed random seed for reproducibility) to project these embeddings, and (3) plots the resulting 2D points. Separate plots are generated to distinguish interacting pairs from non-interacting pairs, and to differentiate source vs. target data. As shown in Figures 3.14 and 3.15, comparing the t-SNE projections at different epochs allows us to visually confirm whether the source and target data clusters become more closely aligned over time.



(a) Epoch 1 (b) Epoch 10 (c) Epoch 20

Figure 3.14: t-SNE projections of interacting source and target samples at different epochs. Notice how source and target clusters become more aligned over time. Blue dots represent source and red ones are the target data.



(a) Epoch 1 (b) Epoch 10 (c) Epoch 20

Figure 3.15: t-SNE projections of non-interacting source and target samples at different epochs. As with the interacting samples, source and target data for non-interacting pairs also become more intertwined as training proceeds. Green dots represent source and orange ones are the target data.

CHAPTER 4

RESULTS

In this chapter, we present and analyze our experimental findings for both in-domain and cross-domain drug–target interaction (DTI) prediction. Our work investigates two key aspects: (1) the effectiveness of various input fusion strategies, and (2) the role of domain adaptation techniques in enhancing cross-domain performance.

We begin by evaluating three fusion methods in an in-domain setting:

1. Simple concatenation,
2. Concatenation followed by self-attention, and
3. Cross-attention.

For these experiments, we report training and validation loss curves (Figure 4.1) and monitor the evolution of key performance metrics such as accuracy, precision, recall, F1 score, and Area Under the Receiver Operating Characteristic curve (AUROC) (Figure 4.2). We conclude the in-domain evaluation with test set performance metrics summarized in Table 4.1.

Next, we extend our analysis to the cross-domain setting using two domain adaptation (DA) methods:

1. Conditional Adversarial Domain Adaptation (CDAN), and
2. Maximum Mean Discrepancy (MMD) Loss-based Domain Adaptation.

For each DA approach, we evaluate models on both the validation and test sets using a comprehensive array of metrics (accuracy, precision, recall, F1, AUROC, and

Matthews correlation coefficient (MCC)). The results reported in Tables 4.2 and 4.3 correspond to the models that achieved the highest MCC during training on the target validation set. In addition, we monitor intermediate performance indicators—such as loss curves and domain alignment trends—to gain deeper insights into the training dynamics of these methods.

Rather than detailing all 200 model configurations, we focus on four promising configurations:

- Two configurations from CDAN (Configs 3 and 16), and
- Two configurations from MMD (Config 19, evaluated under two different data loading modes).

To obtain these configurations, we randomly sampled possible hyperparameter sets defined in Sections 3.5.10.3 (for CDAN) and 3.5.10.2 (for MMD), rather than performing an exhaustive grid search. Finally, in Section 4.8, we provide insights into the runtimes of representative configurations and the hardware setup used throughout our experiments.

To offer a full picture of the broader hyperparameter space, we include the complete validation and test performance results for all 25 configurations in **Appendix A**.

The structure of this chapter is as follows:

- **Section 4.1** presents the performance evaluation in the in-domain setting.
- **Section 4.2** reports the validation performance of CDAN-based models.
- **Section 4.3** discusses the intermediate training dynamics for CDAN.
- **Section 4.4** details the validation performance of MMD-based alignment.
- **Section 4.5** reflects on the training dynamics observed with MMD-based domain alignment.
- **Section 4.6** presents the test set performance and benchmarking results against state-of-the-art baselines.

- **Section 4.7** provides further discussion and concluding remarks on the benchmarking outcomes.

- **Section 4.8** details the runtime and hardware information for representative experiments.

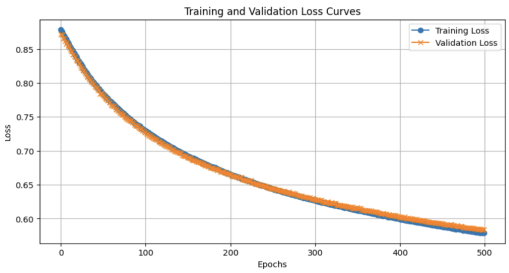
4.1 Evaluation & Hyperparameter Tuning for In-Domain Setting

We have avoided extensive hyperparameter tuning for in-domain setting and selected parameters that provided stable training and acceptable baseline performance based on preliminary trials to assess the effectiveness of fusion strategies.

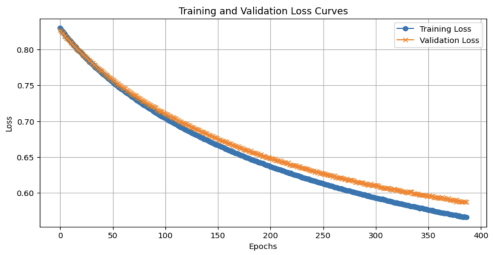
In these baseline in-domain experiments, we utilized a straightforward classifier network with 2 layers, alongside a batch size of 256 and a learning rate of 1.0×10^{-6}, along with a binary cross-entropy loss function.

Upon training, we employed a typical early stopping mechanism to avoid overfitting automatically, where the training is stopped if validation performance (measured by loss) does not improve for a certain number of epochs (patience).

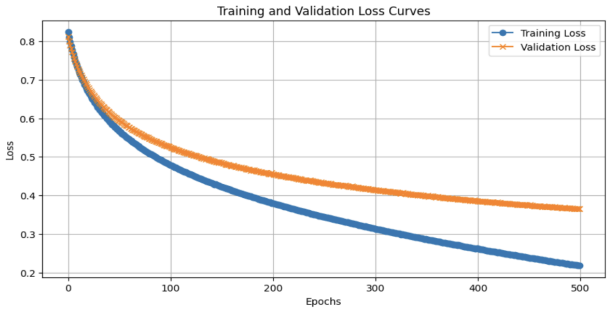
Figure 4.1 demonstrates training and validation loss curves for different fusion methods. Here, the Figure 4.1a corresponds to simple concatenation with a multi-layer perceptron (MLP), Figure 4.1b to concatenation plus self-attention, and the Figure 4.1c. The blue curves represent training losses and the orange ones represent validation losses.



(a) Concatenation with MLP



(b) Concatenation with self-attention

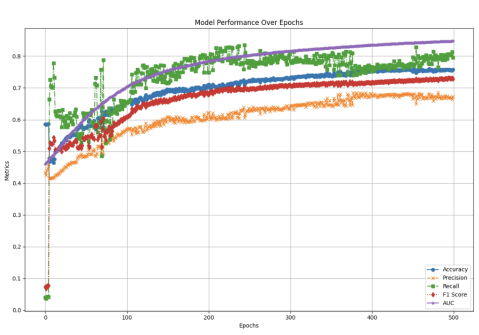


(c) Cross-attention

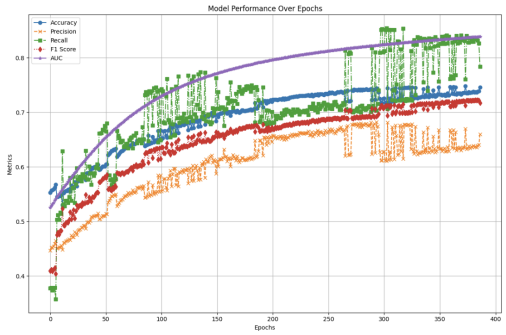
Figure 4.1: Training and validation losses for in-domain experiments across different fusion strategies.

The plots in Figure 4.1 illustrate the learning processes of networks that have not yet converged (the learning is halted based on the maximum number of epochs we set), particularly the first two methods (concatenation and self-attention). Though all these fusion methods seem to pose promising learning curves, cross-attention appears to converge more rapidly. This faster convergence can be attributed to its explicit capability in assessing the inter-modal interactions.

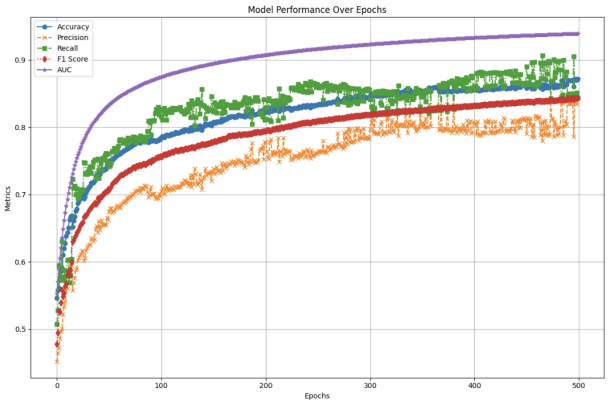
In addition, Figure 4.2 shows performance metrics measured on the validation set over epochs, where Figure4.2a corresponds to simple concatenation, Figure4.2b to self-attention, and Figure4.2c to cross-attention. The metrics displayed include accuracy (green), precision (orange), recall (green), F1-score (red), and AUROC (purple). All these align with the learning curves above, i.e., they appear promising.



(a) Performance of Concatenation



(b) Performance of Self-Attention



(c) Performance of Cross-Attention

Figure 4.2: Validation performance metrics over epochs for different in-domain models.

The performance of various fusion methods on the test set is detailed in Table 4.1, with cross-attention significantly outperforming the other methods. The first two methods show nearly identical results across all metrics. In contrast, cross-attention not only leads in overall performance but also shows a particularly strong advantage in precision. It is hard to attribute this observation to anything specific, however, the other methods apparently pose an imbalance in the precision-recall trade-off, favoring recall. That is, they clearly have a general tendency to classify instances as positive.

Table 4.1: Performance Metrics for Different Input Fusion Methods

Input Fusion	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC**
Concat + MLP	0.74	0.66	0.78	0.71	0.84
Concat + SA + MLP	0.74	0.66	0.78	0.71	0.83
Cross Attention	0.86	0.83	0.83	0.84	0.94

Still, provided that the networks are sufficiently complex, the other fusion methods theoretically can match the learning capabilities of cross-attention, as suggested by the Universal Approximation Theorem. This is evident from the learning curves, which indicate that these models are still underfit and have the potential to achieve performance on par with that of cross-attention. Still, cross-attention's more explicit representation ability allows it to converge more rapidly. That's why we have decided to use **cross attention for input fusion for the remainder of our work.**

4.2 Performance Results on the Validation Set for CDAN

In this section, we focus on the most promising configurations among the 100 experiments conducted using CDAN for domain alignment. Table 4.2 presents the validation performance in terms of accuracy, precision, recall, F1 score, Area Under the Receiver Operating Characteristic curve (AUROC), and Matthews correlation coefficient (MCC).

Table 4.2: Validation performance of selected configurations using CDAN.

Config	**Load Mode**	**Accuracy**	**Precision**	**Recall**	**F1**	**AUROC**	**MCC**
3	same_size	0.60	0.54	0.93	0.68	0.65	0.30
16	cycle_smaller	0.63	0.60	0.63	0.62	0.68	0.27

Alongside these final metrics on the validation set, we also provide intermediate training details for a deeper understanding of the learning process, depicted in Figure 4.3 and Figure 4.4:

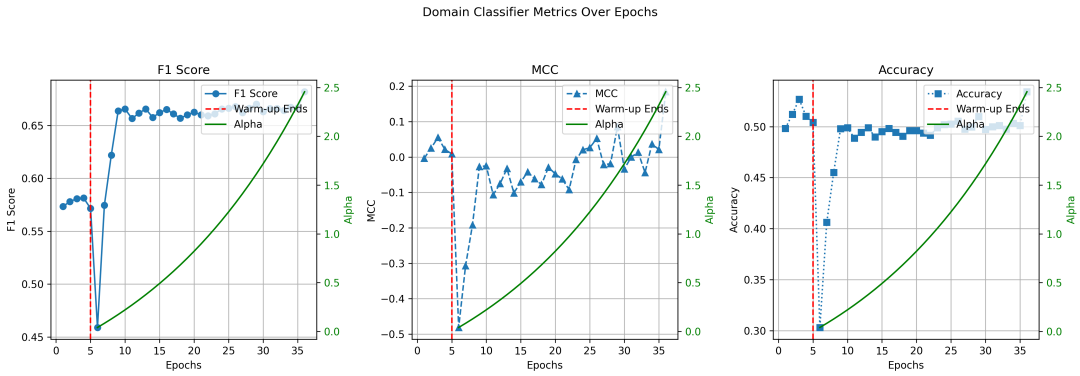
- **Interaction Classification Metrics:** Plots the change in accuracy, precision, recall, F1 score, AUROC, and MCC for both the source (in orange) and the target (in blue) validation sets in the course of training. The vertical red dotted line marks the end of the warmup period, and the green curve represents the changing α value (determining the magnitude of adversarial objective) over training. (Depicted in Figure 4.3c and Figure 4.4c.)

- **Loss Curves:** Presented in Figure 4.3b and Figure 4.4b

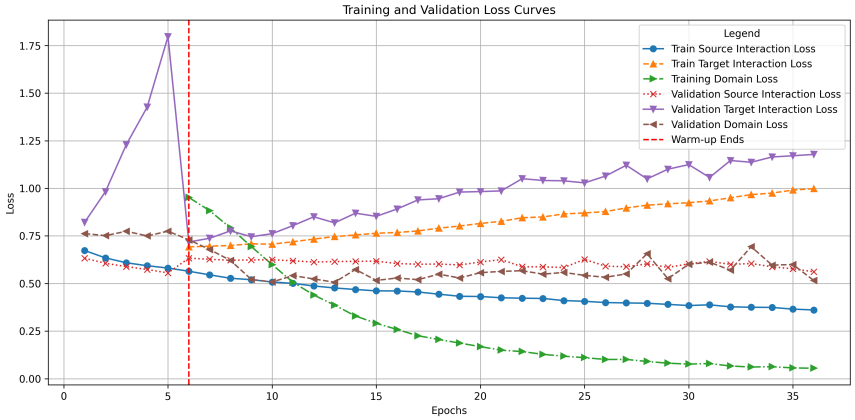
    1. Training source interaction loss (blue with circles),
    2. Training target interaction loss (orange with triangles),
    3. Training domain loss (green with triangles),
    4. Validation source interaction loss (red with crosses),
    5. Validation target interaction loss (purple with triangles),
    6. Validation domain loss (brown with triangles).

    The same red dotted line indicates the end of the warmup period.

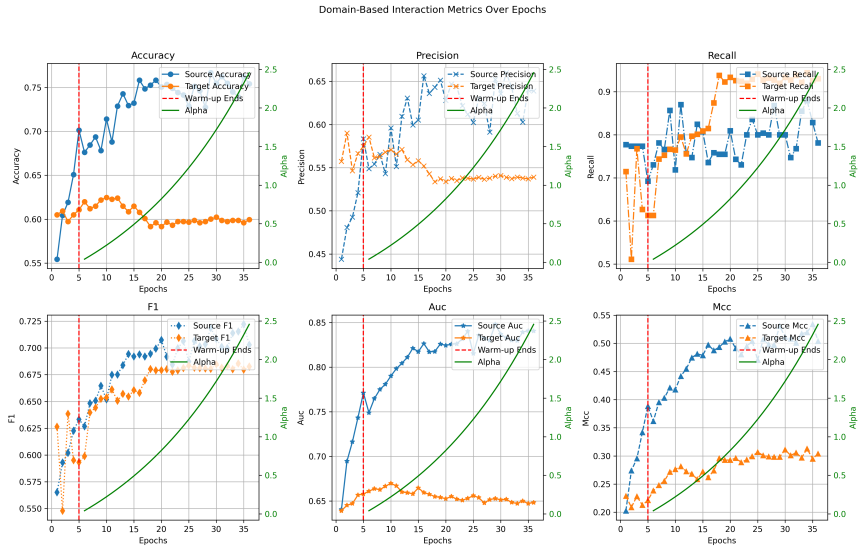
- **Domain Discriminator Metrics:** Metrics for the domain discriminator, including MCC, F1 score, and accuracy. These plots also highlight the end of the warmup period and depict the changing α value. We expect these to be at random performance (i.e., 0 for MCC, 0.5 for F1 score and accuracy). These are shown in Figure 4.3a and Figure 4.4a.



(a) Domain metrics for Configuration 3.

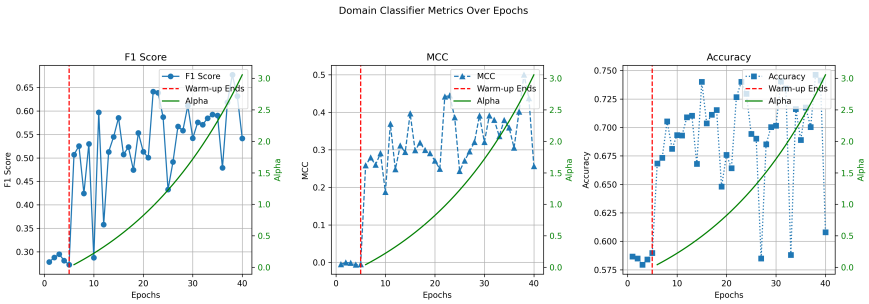


(b) Loss curves for Configuration 3.

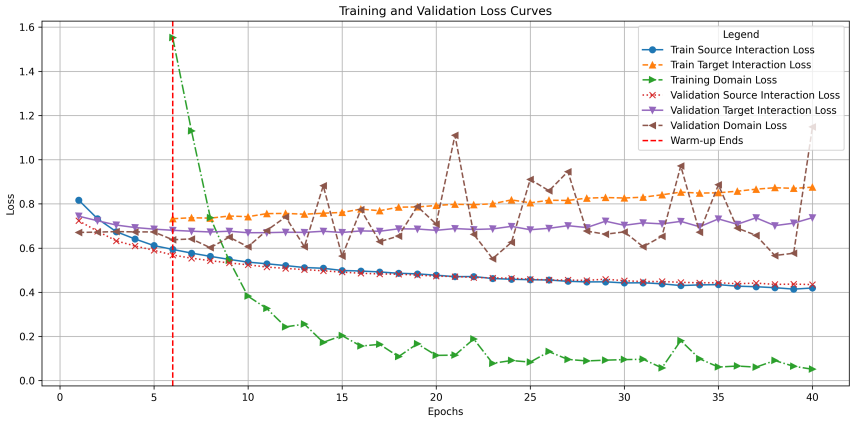


(c) Classification metrics for Configuration 3.

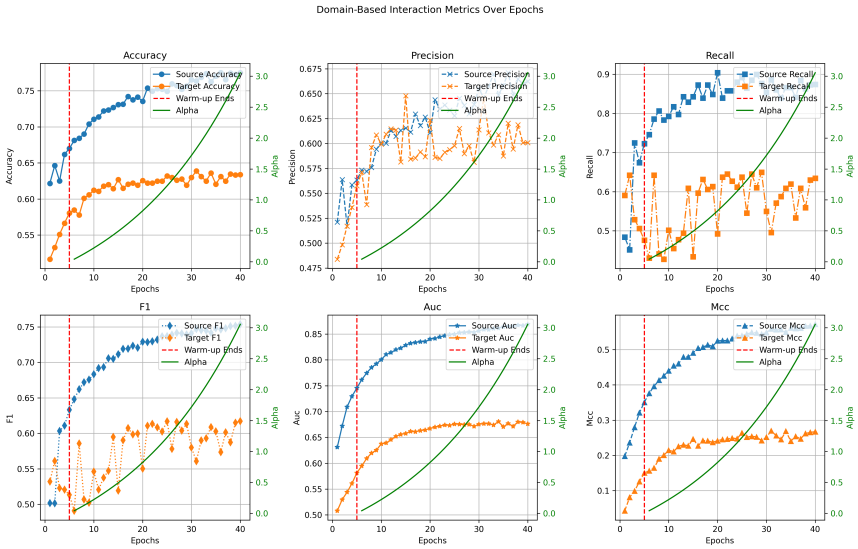
Figure 4.3: Evaluation metrics for Configuration 3.



(a) Domain metrics for Configuration 16.



(b) Loss curves for Configuration 16.



(c) Classification metrics for Configuration 16.

Figure 4.4: Evaluation metrics for Configuration 16.

4.3 Discussion on the Intermediate Performance Results Achieved Using CDAN

Reflection on the Final Performance of Config 3 on Validation Set Table 4.2 highlights the most promising validation performance on target set encountered during training. Below, we compare these highlighted configurations:

- **Config 3** has a higher MCC (0.30) but slightly lower AUROC compared to Config 16. In terms of F1 score, Config 3 yields a considerably superior performance to 16, which can be attributed to the imbalance between precision and recall. In particular, Config 3 has a considerably high recall and low precision values, indicating a positive bias, which might not be optimal.
- **Config 16** achieves a better AUC (0.68) but a slightly lower MCC (0.27). Though, it has a lower F1 score (0.62), it has a higher precision (0.60) and a lower recall (0.63) compared to Config 3. This setting appears to have managed a better trade-off in between precision and recall.

Reflection on the Performance of Config 3 Throughout Training Figure 4.3 depicts detailed performance curves, with domain discriminator performance (Figure 4.3a), various losses (Figure 4.3b), and how the classification performance evolves (Figure 4.3c).

The domain metrics in Figure 4.3a hover near random performance, except for the F1 score, which is slightly better than random. During training, both MCC (expected to be 0) and Accuracy (expected to be 0.5) remain close to random levels, which aligns with our adversarial training objective. Specifically, our aim is to learn a feature extractor that outputs embeddings which the discriminator cannot distinguish. Achieving random discriminator performance indicates success in this aspect.

Figure 4.3b shows a substantial drop in the validation target interaction loss (purple curve) at the end of the warmup period (red dotted line), indicating the effect of domain alignment introduced at that epoch. However, as training progresses, the validation loss for the target (purple) begins to rise, suggesting overfitting to the source data. Still, we do not observe a substantial drop in the source interaction loss (red)

either, though its supervision is directly used for training the classifier. This shows the adversarial method works, as the source interaction loss does not drop as expected. However, this pattern underscores the challenges of the adversarial objective. The persisting gap between source and target interaction losses may be attributed to our scheduling of α (determining the strength of adversarial routine) or the complexity disparity between the classifier and the discriminator. Specifically, while we achieve the desired random performance in the domain discriminator, the classifier appears to still focus on distinctive, non-transferrable features specific to the source, a common issue in such adversarial settings.

Figure 4.3c shows good performance in terms of recall, which also boosts the F1 score, despite a notable decline in precision. Though there is an increasing trend in MCC, overall, the training metrics show inconsistency, as each focuses on different objectives for evaluation. In adversarial training scenarios, such fluctuations are common, as training multiple networks with conflicting objectives is an inherently challenging task.

Reflection on the Performance of Config 16 Throughout Training: Figure 4.4 depicts performance curves similar to those in Config 3 but shows a different trajectory.

Figure 4.4a shows that the domain discriminator performs better than random, suggesting the alignment is slightly less adversarially balanced than in Config 3.

The classifier metrics in Figure 4.4c show more consistency among each other, compared to those in Config 3, despite fluctuations in precision, recall, and F1 score. This time, we see an opposite trend in terms of precision-recall trade-off in that we see higher precision than recall. It is hard to attribute these behavioral changes to anything specific in the model configuration. However, we infer that a more consistent behavior across all metrics suggest a more robust approach compared to Config 3.

The loss curves (Figure 4.4b) still reflect typical adversarial training behaviors, though the validation target interaction loss (purple line) does not drop as sharply following the warmup period, indicating that domain alignment may be less immediately effective for target data in this configuration.

Remarks on Domain Alignment Using Adversarial Objective: Overall, these discrepancies highlight the complexity of adversarial training. Factors such as hyperparameter tuning, length of warmup period, α scheduling, and the capacities of the domain discriminator and classifier have a significant effect on performance. Additionally, although we conducted experiments across all defined loading modes for each model configuration, we did not observe any consistent advantages or disadvantages associated with the choice of loading.

4.4 Performance Results on the Validation Set for CMMD

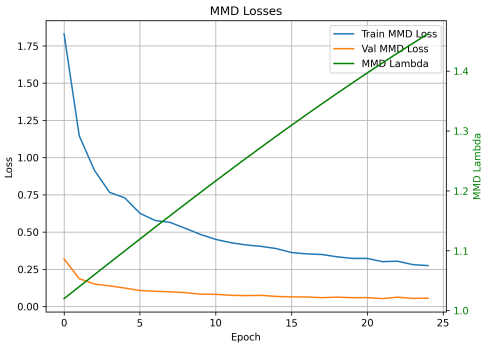
We next evaluate using MMD-based domain alignment. Table 4.3 shows two top-performing configurations on the validation set, comparing accuracy, precision, recall, F1 score, AUC, and MCC.

Table 4.3: Validation performance of top configurations using MMD-based alignment.

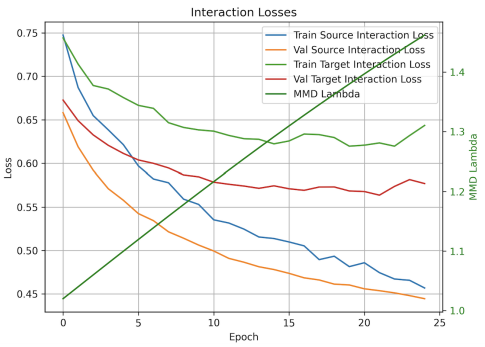
Config	Load Mode	Accuracy	Precision	Recall	F1	AUROC	MCC
19	cycle_smaller	0.65	0.69	0.57	0.62	0.73	0.31
19	sample_larger	0.67	0.66	0.70	0.68	0.77	0.35

In addition to these final metrics, we also present intermediate training details for deeper insight:

- **MMD Loss Curves:** We plot both training (blue) and validation (green) MMD losses alongside the λ (MMD weight in the total loss) per epoch.
- **Interaction Loss Curves:**
    1. Training source interaction loss (blue),
    2. Validation source interaction loss (orange),
    3. Training target interaction loss (green),
    4. Validation target interaction loss (red),
    5. MMD λ value (green line).

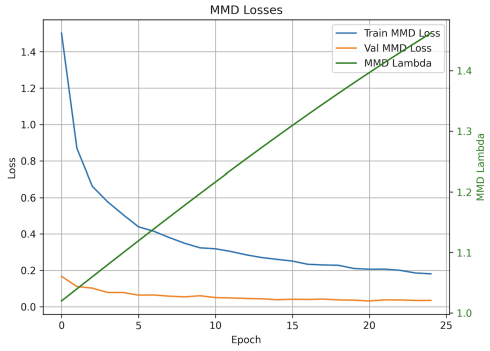


(a) MMD Loss for config19 (sample_larger).

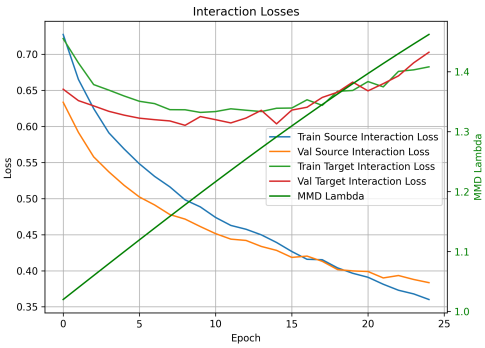


(b) Interaction Loss for config19 (sample_larger).

Figure 4.5: MMD alignment (sample_larger). The left plot shows MMD losses, while the right plot shows interaction losses.



(a) MMD Loss for config19 (cycle_smaller).



(b) Interaction Loss for config19 (cycle_smaller).

Figure 4.6: MMD alignment (cycle_smaller). The left plot shows MMD losses, while the right plot shows interaction losses.

4.5 Discussion on the Intermediate Performance Results Achieved Using MMD

Similar to the results in Table 4.2, Table 4.3 shows the most promising validation metrics observed during MMD-based training. We again see trade-offs between precision and recall, but these are notably less pronounced than those reported with CDAN. Overall, the MMD-based approach yields higher AUROC and MCC values, suggest-

ing more robust performance relative to the CDAN configurations.

Figures 4.5a and 4.6a both show a steady decline in MMD losses for training and validation, indicating successful domain alignment over time. In addition, Figures 4.5b and 4.6b illustrate broadly similar trends in source and target interaction losses, further supporting our interpretation of successful domain alignment.

Nevertheless, in Figure 4.6b, we observe an increase in the target interaction loss roughly after epoch 15, while the source interaction loss continues to decrease. This indicates an overfitting to the source domain and triggers our early stopping mechanism (with a defined patience window).

4.6 Performance Results on the Test Set & Benchmarking

Table 4.4 compares our models' performance on the test set against two existing state-of-the-art methods, DrugBAN [40] and CAT-DTI [41], for cross-domain DTI prediction. Here, the performance metrics of the existing methods are presented in the first two rows in terms of AUROC, F1 score, and Accuracy. Because these baselines do not provide the trained models, codebases, or details for precision, recall, and MCC metrics, we are limited to partial comparisons. Nonetheless, we include all metrics we monitor (precision, recall, MCC) to give a more comprehensive view of each model's strengths and limitations.

In addition to our methods where we employ domain adapatation methods, we also include a baseline configuration, Ours no DA, where the model is trained only on the source dataset without any adaptation to the target domain. This serves as an ablation-like study, showing how much the use of domain adaptation improves performance.

We report four main configurations of our approach with domain adaptation: two using CDAN (Ours1 corresponds to Config 3, Ours2 to Config 16) and two using MMD-based alignment (Ours3 corresponds to Config 19 with a "cycle_smaller" loading mode, and Ours4 to the same config with a "sample_larger" loading mode).

Table 4.4: Performance Metrics for Different Models on the Test Set

Model	**AUROC**	**F1**	**Accuracy**	**Precision**	**Recall**	**MCC**
DrugBAN	0.61	**0.69**	0.55	-	-	-
CAT-DTI	0.68	**0.69**	0.57	-	-	-
Ours no DA	0.58	0.58	0.54	0.51	0.68	-
Ours1 (CDAN1)	0.64	0.68	0.60	0.54	0.93	0.30
Ours2 (CDAN2)	0.64	0.58	0.60	0.57	0.59	0.21
Ours3 (MMD1)	0.74	0.61	0.66	**0.66**	0.56	0.32
Ours4 (MMD2)	**0.75**	0.65	**0.67**	0.63	**0.68**	**0.33**

4.7 Discussion on Benchmarking Results

One notable concern with the reported results from the state-of-the-art models is that their F1 scores are substantially higher than their accuracies. Mathematically, this suggests an imbalance between precision and recall. Since the baselines do not provide these individual metrics, it is likely that their models achieve very high values for one metric at the expense of the other.

Our baseline configuration, **Ours no DA**, demonstrates the performance we achieve without any domain adaptation; it obtains an AUROC of 0.58 and an F1 of 0.58. Compared to this ablation, the domain-adapted models show clear gains in AUROC, accuracy, and in many cases recall and MCC, highlighting the overall benefits of our DA strategies.

Our best MMD-based configuration (Ours4) outperforms the baselines in terms of AUROC (0.75 vs. 0.61/0.68) and accuracy (0.67 vs. 0.55/0.57). However, it slightly falls behind in terms of F1 compared to DrugBAN and CAT-DTI (0.65 vs. 0.69). Because of the absence of recall and precision scores from both baselines, we are not able to discuss and reflect further on the reasons why.

Moreover, Ours4 achieves a recall of 0.68 and an MCC of 0.33, reflecting a stronger

correlation between predictions and labels. In contrast, our CDAN-based models (Ours1, Ours2) yield higher recall in one instance (0.93 for Ours1), but suffer from lower precision and AUROC, indicating a trade-off that favors true positives at the cost of additional false positives.

These results suggest that MMD-based alignment, when used as a regularizing term, can provide more stable and robust performance than adversarial training for domain adaptation tasks. It is challenging to attribute the advantages and disadvantages of these deep learning-based approaches to specific conditions. However, the relatively inferior performance of CDAN, despite having greater expressive power, can be linked to the complexities it introduces, such as conflicting objectives of learning networks (inherent to adversarial tasks), a larger hyperparameter space, and the need for careful balancing.

In addition, our models achieve on par or superior performance in most classification metrics, even though the baseline methods employ custom feature extractors for proteins and drugs. While learning custom feature extractors can enhance representation quality specialized for the remainder of the networks in the DTI prediction and domain alignment pipeline, it also requires training a large number of parameters. Given the complexity of the feature extractors (with the use of transformer-based architectures) and the limited size of our dataset (approximately 30K drug–protein pairs), the potential benefits of end-to-end feature learning is likely constrained. In fact, our models relying on pre-trained models (ChemBERTa and ProtT5XL) for feature extraction—even without fine-tuning—appear to offer comparable or superior performance in this setting.

4.8 Runtimes and Hardware Setup

All experiments were conducted on a machine equipped with an Intel Xeon CPU and an **NVIDIA Quadro GV100 GPU with 32 GB of memory** (Driver Version: 535.183.01, CUDA Version: 12.2). We primarily examine two methods of domain alignment: MMD-based training (Section 3.5.10.2) and CDAN-based training (Sections 3.5.10.3). Rather than running an exhaustive grid search, we randomly sample

from a predefined hyperparameter space and train a model under each sampled configuration.

Below, we report the runtime for one representative configuration from each method. Note that different hyperparameter settings (e.g., batch size, learning rates, and network architectures) can lead to substantial variations in training time.

- **Conditional MMD (CMMD)**: Using the following configuration:

```
{
    "batch_size": 128,
    "normalize": true,
    "lr_fe": 1e-06,
    "lr_cls": 0.0001,
    "warmup_epochs": 5,
    "bigger_classifier": false,
    "classifier_dropout": 0.3,
    "classifier_hidden_dim": 256,
    "kernel_type": "laplacian",
    "mode": "cycle_smaller"
}
```

  the total training time was approximately **2465.59 seconds**.

- **CDAN**: For the representative configuration shown below:

```
batch_size=256
normalize=False
mode="cycle_smaller"
fine_tune=False
lr_fe=1.0e-5
lr_cls=1.0e-4
lr_disc=1.0e-5
max_num_epochs=250
```

```
warmup_epochs=10
patience=10
use_da=True
weight_interaction=0.5
weight_domain=1
```

the total training time was approximately **843.59 seconds**.

The more pronounced runtime of the Conditional MMD approach (CMMD) can primarily be attributed to the fact that kernel-based losses are less amenable to GPU parallelization compared to adversarial training losses. Although GPUs generally excel at parallel processing, the kernel-trick computations in MMD-based alignment can introduce additional overhead, thus prolonging training time.

CHAPTER 5

CONCLUSION & FUTURE DIRECTIONS

In this thesis, we introduce a domain adaptation framework specifically tailored for predicting drug–target interactions (DTI), aimed at addressing the distribution shift commonly seen between training and inference data. We conceptualize DTI prediction as a binary classification task utilizing multimodal inputs, namely drugs and proteins. We evaluate various fusion methods, with cross-attention emerging as a particularly effective technique for integrating these modalities. To tackle domain shift, we utilize both adversarial-based and Maximum Mean Discrepancy (MMD)-based methods to align the source and target distributions. Notably, our work marks the first reported use of MMD-based alignment in the DTI domain, to our knowledge. Our findings suggest that MMD-based alignment provides a more consistent training process compared to the more hyperparameter-sensitive adversarial methods.

Additionally, our research demonstrates that leveraging pretrained models (ChemBERTa for drug representations and ProtT5XL for protein representations) without fine-tuning can achieve comparable or superior performance across key metrics. This approach substantially reduces both the training complexity and computational costs while maintaining robust model performance.

Future work could investigate the use of more advanced or multimodal featurizers, such as SaProt [18] for protein representations or SELFormer [22] for drug representations. Instead of relying solely on pretrained models for feature extraction, one could also consider fine-tuning these models to capture more task-specific features. Such enhancements have the potential to further refine performance. Additionally, there is a vast hyperparameter space yet to be explored, offering opportunities for even greater performance gains.

Moreover, the dataset used for our cross-domain setting was constructed using hierarchical clustering-based splits on proteins and drugs separately, thereby ignoring any existing drug–protein interaction relations in the splitting process. More sophisticated approaches that consider network-based splits and capture the actual relationship between proteins and drugs have been proposed [45]. Such more challenging datasets could be employed in future work to provide additional insights into the robustness and real-world applicability of domain-adapted DTI models.

Ultimately, the stability of MMD-based domain adaptation, coupled with the effectiveness of pretrained feature extraction, positions our work as a promising foundation for continued research and real-world applications in DTI prediction.

REFERENCES

[1] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," *arXiv preprint arXiv:1711.03213*, 2017.

[2] P. Bai, F. Miljković, Y. Ge, N. Greene, B. John, and H. Lu, "Hierarchical clustering split for low-bias evaluation of drug-target interaction prediction," in *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 641–644, 2021.

[3] L. Ouyang and A. Key, "Maximum mean discrepancy for generalization in the presence of distribution and missingness shift," 2022.

[4] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 1180–1189, PMLR, 2015.

[5] M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Conditional adversarial domain adaptation," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1640–1650, 2018.

[6] B. Buchfink, K. Reuter, and H.-G. Drost, "Sensitive protein alignments at tree-of-life scale using diamond," *Nature Methods*, vol. 18, no. 4, pp. 366–368, 2021.

[7] L. S. Johnson, S. R. Eddy, and E. Portugaly, "Hidden markov model speed heuristic and iterative hmm search procedure," *BMC Bioinformatics*, vol. 11, no. 431, pp. 1–8, 2010.

[8] J. Mistry, S. Chuguransky, L. Williams, and et al., "Pfam: The protein families database in 2021," *Nucleic Acids Research*, vol. 49, no. D1, pp. D412–D419, 2021.

[9] E. Asgari and M. R. K. Mofrad, "Continuous distributed representation of bi-

ological sequences for deep proteomics and genomics," *PLOS ONE*, vol. 10, no. 11, p. e0141287, 2015.

[10] R. Rao, J. Liu, R. Verkuil, *et al.*, "MSA Transformer," 2021. bioRxiv: 2021.02.12.430858.

[11] A. Elnaggar, M. Heinzinger, C. Dallago, and et al., "Prottrans: Toward understanding the language of life through self-supervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, 2022.

[12] A. Elnaggar, M. Heinzinger, C. Dallago, and et al., "Prottrans: Toward understanding the language of life through self-supervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, 2022. Alternate reference/DOI: 10.1109/TPAMI.2021.3095381.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 4171–4186, 2019.

[14] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[15] Z. Lin, H. Akin, R. Rao, and et al., "Evolutionary-scale prediction of atomic-level protein structure with a language model," *Science*, vol. 379, no. 6637, pp. 1123–1130, 2023.

[16] T. Hayes, R. Rao, H. Akin, *et al.*, "Simulating 500 million years of evolution with a language model," 2024. bioRxiv: 2024.07.01.600583.

[17] M. Heinzinger, K. Weissenow, J. G. Sanchez, and et al., "ProstT5: Bilingual language model for protein sequence and structure," 2023. bioRxiv: 2023.07.23.550085.

[18] J. Su, C. Han, Y. Zhou, *et al.*, "SaProt: Protein language modeling with structure-aware vocabulary," 2023. bioRxiv: 2023.10.01.560349.

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

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

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

[22] A. Yüksel, E. Ulusoy, A. Ünlü, and T. Doğan, "Selformer: molecular representation learning via selfies language models," *Machine Learning: Science and Technology*, vol. 4, no. 2, p. 025035, 2023.

[23] Y. Wang, Z. Li, and A. Barati Farimani, *Graph Neural Networks for Molecules*, vol. 36 of *Challenges and Advances in Computational Chemistry and Physics*. Cham: Springer, 2023.

[24] Y. Bai, H. Ding, S. Qiao, A. Marinovic, Y. Gu, T. Sun, and W. Wang, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," in *NeurIPS*, 2019.

[25] J. Ngiam, A. Khosla, M. Y. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th International Conference on Machine Learning*, pp. 689–696, 2011.

[26] J. Zhang, Q. Li, *et al.*, "A survey of multimodal learning in biomedical applications and challenges: A comprehensive analysis," *Information Fusion*, vol. 86, pp. 123–140, 2022.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[28] J. Lu, J. Yang, D. Batra, and D. Parikh, "Hierarchical question-image co-attention for visual question answering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 289–297, 2016.

[29] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6077–6086, 2018.

[30] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[31] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, p. 9, 2016.

[32] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3320–3328, 2014.

[33] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pp. 242–264, IGI Global, 2009.

[34] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339, 2018.

[35] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 97–105, 2015.

[36] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," 2016.

[37] R. Chen, X. Liu, S. Jin, J. Lin, and J. Liu, "Machine learning for drug-target interaction prediction," *Molecules*, vol. 23, no. 9, p. 2208, 2018.

[38] C. Cai, S. Wang, Y. Xu, W. Zhang, K. Tang, Q. Ouyang, L. Lai, and J. Pei, "Transfer learning for drug discovery," *Journal of Medicinal Chemistry*, vol. 63, no. 16, pp. 8683–8694, 2020.

[39] A. Dalkıran, A. Atakan, A. S. Rifaioğlu, M. J. Martin, R. Çetin Atalay, A. C. Acar, T. Doğan, and V. Atalay, "Transfer learning for drug–target interaction prediction," *Bioinformatics*, vol. 39, no. Supplement_1, pp. i103–i110, 2023.

[40] P. Bai, F. Miljković, B. John, and H. Lu, "Interpretable bilinear attention network with domain adaptation improves drug-target prediction," *Nature Machine Intelligence*, vol. 5, no. 2, pp. 126–136, 2023.

[41] X. Zeng, W. Chen, and B. Lei, "CAT-DTI: cross-attention and Transformer network with domain adaptation for drug-target interaction prediction," *BMC Bioinformatics*, vol. 25, p. 141, 2024.

[42] W. Chen and H. Hu, "Unsupervised domain adaptation via discriminative classes-center feature learning in adversarial network," *Neural Processing Letters*, vol. 52, pp. 467–483, 2020.

[43] K.-Y. Gao, A. Fokoue, H. Luo, A. Iyengar, D. Dey, and P. Zhang, "Interpretable drug target prediction using deep neural representation," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3371–3377, 2018.

[44] Z. Zhang, M. Li, and J. Yu, "On the convergence and mode collapse of gan," in *SIGGRAPH Asia 2018 Technical Briefs*, SA '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[45] H. Atas Guvenilir and T. Doğan, "How to approach machine learning-based prediction of drug/compound–target interactions," *Journal of Cheminformatics*, vol. 15, no. 1, p. 16, 2023.

APPENDICES

A CDAN Performance with Different Hyperparamter Configurations

A.1 CDAN Configurations

Table 5.1: **CDAN Configurations.** This table lists the randomly chosen hyperparameter configurations used for CDAN. Columns include: **Conf**: configuration number, **N**: whether input features are normalized, **lr_fe**: learning rate for the feature extractor, **lr_cls**: learning rate for the classifier, **lr_d**: learning rate for the discriminator, **wp**: number of warmup epochs, **wi**: weight assigned to the interaction classification loss, **wd**: weight assigned to the domain discriminator loss, **big_d**: whether a larger discriminator network is used, **big_c**: whether a larger classifier network is used, **dropout**: dropout probability in the classifier, **hid**: hidden dimension of the classifier.

Conf	bs	N	lr_fe	lr_cls	lr_d	wp	wi	wd	big_d	big_c	drop	hid
1	256	T	1e-05	1e-05	0.001	5	0.5	1.2	T	T	0.3	256
2	64	F	1e-05	1e-05	1e-05	5	0.5	1	F	F	0.3	128
3	32	T	1e-06	0.0001	0.0001	5	0.5	0.7	F	F	0.3	256
4	128	F	0.0001	1e-05	1e-05	0	0.5	0.3	F	T	0.5	256
5	256	T	1e-05	1e-05	1e-05	0	0.5	1.5	T	F	0.3	128
6	32	T	1e-06	1e-05	1e-05	5	0.5	0.3	T	T	0.5	256
7	128	F	1e-06	0.0001	0.001	20	0.5	1.2	T	T	0.3	128
8	32	T	1e-06	1e-05	0.001	20	0.5	1.5	F	T	0.3	256
9	256	T	1e-06	0.0001	0.0001	0	0.5	1.5	F	T	0.5	128
10	256	T	1e-06	0.001	0.0001	20	0.5	1	F	T	0.5	256
11	64	T	1e-05	0.001	1e-05	0	0.5	0.7	F	T	0.5	256
12	128	F	1e-06	1e-05	0.001	0	0.5	0.5	F	T	0.5	256

Table 5.1 – *Continued from previous page*

Conf	bs	N	lr_fe	lr_cls	lr_d	wp	wı	w_d	big_d	big_c	drop	hid
13	32	T	1e-05	0.001	0.001	20	0.5	0.5	T	T	0.5	256
14	128	T	0.0001	0.0001	0.001	5	0.5	1.5	T	F	0.5	256
15	64	F	0.0001	0.001	0.001	0	0.5	1.2	T	T	0.3	128
16	128	F	1e-06	0.0001	0.001	5	0.5	1.2	T	F	0.3	128
17	32	F	1e-05	0.001	0.0001	5	0.5	1.2	F	T	0.3	256
18	32	F	1e-05	1e-05	0.0001	0	0.5	1.5	T	F	0.5	128
19	32	T	1e-05	0.001	0.0001	5	0.5	1.5	F	T	0.3	128
20	128	F	0.0001	1e-05	1e-05	0	0.5	1	T	T	0.5	128
21	32	T	1e-06	0.001	1e-05	5	0.5	1.5	F	T	0.3	128
22	256	F	1e-06	0.001	0.0001	0	0.5	0.5	T	T	0.5	128
23	32	T	1e-06	1e-05	0.001	5	0.5	1.2	F	T	0.5	128
24	128	F	1e-05	0.0001	1e-05	20	0.5	0.7	F	F	0.5	256
25	64	F	0.0001	0.001	1e-05	5	0.5	0.5	F	T	0.5	256

A.2 CDAN Performance on Validation Set with Different Hyperparameter Configurations

Table 5.2: **CDAN Validation Performance.** Each row shows the validation metrics (accuracy, precision, recall, F1 score, AUC, and MCC) for a given CDAN configuration from Table 5.1 paired with a specified loader mode.

Config	load_mode	val_acc	val_prec	val_recall	val_f1	val_auc	val_mcc
config_1	cycle_smaller	0.59	0.54	0.93	0.68	0.64	0.27
config_1	sample_larger	0.57	0.52	0.79	0.63	0.62	0.18
config_1	same_size	0.60	0.60	0.50	0.54	0.63	0.19
config_2	cycle_smaller	0.56	0.55	0.50	0.52	0.55	0.11
config_2	sample_larger	0.61	0.57	0.74	0.65	0.65	0.23
config_2	same_size	0.60	0.61	0.53	0.57	0.63	0.19
config_3	cycle_smaller	0.57	0.53	0.85	0.65	0.59	0.18
config_3	sample_larger	0.57	0.53	0.89	0.67	0.58	0.20

Table 5.2 – *Continued from previous page*

Config	load_mode	val_acc	val_prec	val_recall	val_f1	val_auc	val_mcc
config_3	same_size	0.60	0.54	0.93	0.68	0.65	0.30
config_4	cycle_smaller	0.59	0.54	0.93	0.69	0.58	0.26
config_4	sample_larger	0.54	0.51	0.59	0.54	0.55	0.09
config_4	same_size	0.56	0.53	0.76	0.63	0.55	0.13
config_5	cycle_smaller	0.59	0.57	0.69	0.62	0.59	0.19
config_5	sample_larger	0.57	0.53	0.86	0.66	0.60	0.21
config_5	same_size	0.59	0.53	0.89	0.66	0.64	0.26
config_6	cycle_smaller	0.60	0.57	0.66	0.61	0.64	0.21
config_6	sample_larger	0.60	0.59	0.54	0.56	0.63	0.19
config_6	same_size	0.61	0.58	0.73	0.64	0.66	0.24
config_7	cycle_smaller	0.56	0.52	0.84	0.64	0.61	0.20
config_7	sample_larger	0.63	0.58	0.75	0.65	0.67	0.27
config_7	same_size	0.59	0.53	0.81	0.64	0.64	0.22
config_8	cycle_smaller	0.57	0.54	0.85	0.66	0.60	0.19
config_8	sample_larger	0.61	0.56	0.77	0.65	0.66	0.25
config_8	same_size	0.61	0.56	0.80	0.66	0.67	0.26
config_9	cycle_smaller	0.57	0.52	0.88	0.65	0.63	0.22
config_9	sample_larger	0.61	0.58	0.60	0.59	0.66	0.22
config_9	same_size	0.57	0.55	0.54	0.54	0.59	0.14
config_10	cycle_smaller	0.59	0.55	0.78	0.64	0.63	0.22
config_10	sample_larger	0.59	0.58	0.52	0.55	0.58	0.17
config_10	same_size	0.61	0.59	0.60	0.59	0.64	0.21
config_11	cycle_smaller	0.54	0.51	0.96	0.66	0.54	0.18
config_11	sample_larger	0.56	0.54	0.51	0.52	0.57	0.11
config_11	same_size	0.59	0.56	0.69	0.62	0.60	0.20
config_12	cycle_smaller	0.55	0.54	0.48	0.51	0.56	0.10
config_12	sample_larger	0.52	0.51	0.53	0.52	0.51	0.04
config_12	same_size	0.52	0.49	0.61	0.54	0.51	0.06
config_13	cycle_smaller	0.58	0.54	0.91	0.68	0.58	0.23
config_13	sample_larger	0.51	0.48	0.67	0.56	0.49	0.04

Table 5.2 – *Continued from previous page*

Config	load_mode	val_acc	val_prec	val_recall	val_f1	val_auc	val_mcc
config_13	same_size	0.59	0.56	0.63	0.59	0.59	0.19
config_14	cycle_smaller	0.60	0.56	0.73	0.63	0.64	0.23
config_14	sample_larger	0.56	0.53	0.90	0.67	0.58	0.20
config_14	same_size	0.63	0.60	0.74	0.66	0.63	0.27
config_15	cycle_smaller	0.55	0.52	0.93	0.67	0.53	0.18
config_15	sample_larger	0.55	0.53	0.53	0.53	0.57	0.10
config_15	same_size	0.60	0.57	0.59	0.58	0.63	0.20
config_16	cycle_smaller	0.63	0.60	0.63	0.62	0.68	0.27
config_16	sample_larger	0.60	0.63	0.41	0.50	0.63	0.20
config_16	same_size	0.62	0.58	0.65	0.61	0.65	0.24
config_17	cycle_smaller	0.56	0.53	0.84	0.65	0.58	0.16
config_17	sample_larger	0.58	0.55	0.70	0.61	0.61	0.17
config_17	same_size	0.57	0.53	0.70	0.60	0.58	0.15
config_18	cycle_smaller	0.59	0.58	0.46	0.52	0.61	0.18
config_18	sample_larger	0.61	0.58	0.71	0.64	0.63	0.22
config_18	same_size	0.54	0.51	0.92	0.65	0.58	0.17
config_19	cycle_smaller	0.60	0.56	0.68	0.61	0.62	0.21
config_19	sample_larger	0.53	0.50	0.90	0.64	0.57	0.15
config_19	same_size	0.58	0.56	0.61	0.59	0.57	0.15
config_20	cycle_smaller	0.50	0.49	1.00	0.66	0.48	0.09
config_20	sample_larger	0.56	0.55	0.56	0.55	0.57	0.12
config_20	same_size	0.59	0.53	0.83	0.65	0.61	0.23
config_21	cycle_smaller	0.52	0.51	0.70	0.59	0.53	0.07
config_21	sample_larger	0.54	0.51	0.70	0.59	0.56	0.11
config_21	same_size	0.63	0.61	0.61	0.61	0.67	0.25
config_22	cycle_smaller	0.58	0.52	0.84	0.64	0.62	0.23
config_22	sample_larger	0.59	0.55	0.86	0.67	0.64	0.22
config_22	same_size	0.60	0.56	0.61	0.58	0.64	0.20
config_23	cycle_smaller	0.61	0.55	0.92	0.69	0.67	0.30
config_23	sample_larger	0.61	0.61	0.52	0.56	0.64	0.22

Table 5.2 – *Continued from previous page*

Config	**load_mode**	**val_acc**	**val_prec**	**val_recall**	**val_f1**	**val_auc**	**val_mcc**
config_23	same_size	0.57	0.53	0.96	0.68	0.61	0.25
config_24	cycle_smaller	0.59	0.58	0.67	0.62	0.61	0.17
config_24	sample_larger	0.64	0.71	0.36	0.48	0.67	0.27
config_24	same_size	0.61	0.62	0.50	0.56	0.65	0.22
config_25	cycle_smaller	0.55	0.53	0.91	0.67	0.56	0.14
config_25	sample_larger	0.54	0.52	0.90	0.66	0.55	0.13

A.3 CDAN Performance on Test Set with Different Hyperparameter Configurations

Table 5.3: **CDAN Test Performance.** Each row shows the test metrics (accuracy, precision, recall, F1 score, AUC, and MCC) for a given CDAN configuration from Table 5.1 paired with a specified loader mode.

Config	load_mode	test_acc	test_prec	test_recall	test_f1	test_auc	test_mcc
config_1	cycle_smaller	0.56	0.52	0.90	0.66	0.63	0.21
config_1	sample_larger	0.57	0.53	0.78	0.63	0.63	0.19
config_1	same_size	0.58	0.55	0.49	0.52	0.61	0.14
config_2	cycle_smaller	0.56	0.53	0.48	0.50	0.56	0.11
config_2	sample_larger	0.57	0.52	0.71	0.60	0.63	0.15
config_2	same_size	0.58	0.55	0.52	0.53	0.61	0.15
config_3	cycle_smaller	0.53	0.50	0.82	0.62	0.57	0.12
config_3	sample_larger	0.53	0.50	0.87	0.63	0.55	0.13
config_3	same_size	0.60	0.54	0.93	0.68	0.64	0.30
config_4	cycle_smaller	0.53	0.50	0.91	0.64	0.53	0.15
config_4	sample_larger	0.54	0.50	0.59	0.54	0.55	0.09
config_4	same_size	0.53	0.49	0.74	0.59	0.54	0.09
config_5	cycle_smaller	0.56	0.52	0.65	0.58	0.59	0.13
config_5	sample_larger	0.57	0.52	0.84	0.64	0.62	0.19
config_5	same_size	0.58	0.53	0.88	0.66	0.62	0.24
config_6	cycle_smaller	0.57	0.54	0.64	0.58	0.64	0.16
config_6	sample_larger	0.57	0.54	0.52	0.53	0.62	0.14
config_6	same_size	0.59	0.54	0.71	0.61	0.63	0.19
config_7	cycle_smaller	0.56	0.51	0.84	0.64	0.60	0.17
config_7	sample_larger	0.59	0.54	0.72	0.62	0.65	0.20
config_7	same_size	0.56	0.52	0.78	0.62	0.62	0.17
config_8	cycle_smaller	0.55	0.51	0.84	0.64	0.60	0.17
config_8	sample_larger	0.59	0.54	0.76	0.63	0.64	0.21
config_8	same_size	0.60	0.55	0.78	0.64	0.66	0.23
config_9	cycle_smaller	0.56	0.52	0.84	0.64	0.64	0.18

Table 5.3 – *Continued from previous page*

Config	load_mode	test_acc	test_prec	test_recall	test_f1	test_auc	test_mcc
config_9	sample_larger	0.58	0.55	0.57	0.56	0.63	0.17
config_9	same_size	0.56	0.53	0.53	0.53	0.59	0.12
config_10	cycle_smaller	0.58	0.53	0.77	0.63	0.62	0.19
config_10	sample_larger	0.55	0.52	0.46	0.49	0.55	0.10
config_10	same_size	0.60	0.56	0.59	0.58	0.64	0.19
config_11	cycle_smaller	0.52	0.49	0.95	0.65	0.53	0.16
config_11	sample_larger	0.54	0.51	0.50	0.50	0.55	0.08
config_11	same_size	0.58	0.54	0.68	0.60	0.59	0.18
config_12	cycle_smaller	0.54	0.51	0.49	0.50	0.56	0.08
config_12	sample_larger	0.50	0.47	0.52	0.49	0.51	0.01
config_12	same_size	0.53	0.49	0.61	0.55	0.54	0.07
config_13	cycle_smaller	0.55	0.51	0.86	0.64	0.57	0.17
config_13	sample_larger	0.49	0.46	0.66	0.55	0.49	0.00
config_13	same_size	0.55	0.52	0.57	0.54	0.55	0.11
config_14	cycle_smaller	0.58	0.53	0.70	0.61	0.62	0.18
config_14	sample_larger	0.54	0.50	0.90	0.64	0.56	0.16
config_14	same_size	0.61	0.57	0.73	0.64	0.64	0.25
config_15	cycle_smaller	0.52	0.49	0.93	0.64	0.55	0.15
config_15	sample_larger	0.54	0.50	0.52	0.51	0.55	0.07
config_15	same_size	0.58	0.55	0.55	0.55	0.61	0.15
config_16	cycle_smaller	0.60	0.57	0.59	0.58	0.64	0.21
config_16	sample_larger	0.61	0.62	0.43	0.51	0.64	0.21
config_16	same_size	0.60	0.56	0.65	0.60	0.64	0.21
config_17	cycle_smaller	0.52	0.49	0.82	0.61	0.55	0.09
config_17	sample_larger	0.57	0.53	0.72	0.61	0.61	0.17
config_17	same_size	0.56	0.52	0.69	0.59	0.59	0.14
config_18	cycle_smaller	0.59	0.56	0.48	0.52	0.62	0.16
config_18	sample_larger	0.58	0.54	0.69	0.61	0.62	0.18
config_18	same_size	0.53	0.50	0.93	0.65	0.57	0.17
config_19	cycle_smaller	0.58	0.54	0.68	0.60	0.61	0.18

Table 5.3 – *Continued from previous page*

Config	load_mode	test_acc	test_prec	test_recall	test_f1	test_auc	test_mcc
config_19	sample_larger	0.51	0.48	0.87	0.62	0.53	0.08
config_19	same_size	0.57	0.53	0.58	0.56	0.57	0.14
config_20	cycle_smaller	0.47	0.47	0.99	0.64	0.49	0.05
config_20	sample_larger	0.55	0.51	0.52	0.52	0.56	0.09
config_20	same_size	0.59	0.53	0.85	0.66	0.61	0.23
config_21	cycle_smaller	0.52	0.49	0.69	0.57	0.55	0.06
config_21	sample_larger	0.53	0.49	0.65	0.56	0.55	0.08
config_21	same_size	0.61	0.58	0.55	0.57	0.65	0.21
config_22	cycle_smaller	0.56	0.52	0.83	0.64	0.62	0.19
config_22	sample_larger	0.57	0.53	0.85	0.65	0.64	0.21
config_22	same_size	0.59	0.56	0.63	0.59	0.62	0.19
config_23	cycle_smaller	0.58	0.53	0.91	0.67	0.65	0.25
config_23	sample_larger	0.60	0.57	0.52	0.54	0.64	0.18
config_23	same_size	0.54	0.50	0.95	0.66	0.60	0.20
config_24	cycle_smaller	0.57	0.53	0.69	0.60	0.61	0.17
config_24	sample_larger	0.60	0.62	0.33	0.43	0.63	0.18
config_24	same_size	0.59	0.57	0.47	0.52	0.62	0.17
config_25	cycle_smaller	0.51	0.48	0.92	0.63	0.55	0.11
config_25	sample_larger	0.50	0.48	0.89	0.62	0.53	0.07

B MMD Performance with Different Hyperparamter Configurations

B.1 MMD Configurations

Table 5.4: **MMD Configurations.** This table lists the randomly chosen hyperparameter configurations used for MMD. Columns include: **Conf**: configuration number, **bs**: batch size, **N**: whether input features are normalized, **lr_fe**: learning rate for the feature extractor, **lr_cls**: learning rate for the classifier, **wp**: number of warmup epochs, **big_cls**: whether a larger classifier network is used, **cls_drp**: dropout probability in the classifier network, **cls_hid**: hidden dimension of the classifier, **kernel**: kernel function used for MMD (e.g., *rbf*, *laplacian*, or *linear*).

Conf	bs	N	lr_fe	lr_cls	wp	big_cls	cls_drp	cls_hid	kernel
Config1	32	false	1e-06	1e-05	0	true	0.3	128	laplacian
Config2	128	false	1e-05	0.0001	0	false	0.5	256	laplacian
Config3	64	true	1e-06	1e-05	5	false	0.3	256	laplacian
Config4	256	false	1e-05	0.001	0	false	0.3	256	rbf
Config5	128	true	1e-06	0.0001	0	false	0.5	128	linear
Config6	64	false	1e-06	1e-05	5	true	0.5	256	rbf
Config7	256	false	1e-05	0.0001	0	true	0.5	256	linear
Config8	128	false	1e-06	0.0001	0	false	0.5	128	rbf
Config9	128	false	1e-06	1e-05	0	true	0.5	128	rbf
Config10	64	false	1e-06	1e-05	0	true	0.5	128	polynomial
Config11	256	true	0.0001	1e-05	5	false	0.3	128	polynomial
Config12	256	false	1e-06	0.001	5	false	0.3	128	polynomial
Config13	256	true	1e-05	0.001	0	false	0.5	256	linear
Config14	256	true	0.0001	1e-05	0	false	0.5	128	laplacian
Config15	32	true	0.0001	0.0001	5	false	0.3	128	polynomial
Config16	32	false	0.0001	0.001	0	false	0.3	256	polynomial
Config17	32	true	1e-06	0.001	5	true	0.5	128	rbf
Config18	32	true	1e-05	1e-05	0	false	0.3	256	rbf
Config19	256	true	0.0001	0.0001	0	false	0.5	256	linear
Config20	256	true	1e-06	0.001	0	true	0.3	256	polynomial

Table 5.4 – *Continued from previous page*

Conf	**bs**	**N**	**lr_fe**	**lr_cls**	**wp**	**big_cls**	**cls_drp**	**cls_hidd**	**kernel**
Config21	128	false	1e-05	0.0001	5	true	0.3	128	laplacian
Config22	128	false	1e-06	1e-05	5	false	0.5	128	laplacian
Config23	256	true	1e-06	0.0001	0	true	0.3	256	rbf
Config24	256	true	0.0001	1e-05	5	false	0.5	256	laplacian
Config25	64	false	1e-06	0.001	0	true	0.5	256	rbf

B.2 MMD Performance on Validation Set with Different Hyperparameter Configurations

Table 5.5: **MMD Validation Performance.** Each row shows the validation metrics (accuracy, precision, recall, F1 score, AUC, and MCC) for a given MMD configuration from Table 5.4 paired with a specified loader mode.

Config	load_mode	val_acc	val_prec	val_recall	val_f1	val_auc	val_mcc
config_1	cycle_smaller	0.62	0.62	0.49	0.55	0.66	0.23
config_1	sample_larger	0.57	0.57	0.36	0.44	0.60	0.13
config_1	same_size	0.54	0.54	0.31	0.40	0.56	0.07
config_2	cycle_smaller	0.63	0.62	0.57	0.60	0.69	0.25
config_2	sample_larger	0.62	0.63	0.51	0.56	0.65	0.24
config_2	same_size	0.62	0.64	0.43	0.52	0.68	0.24
config_3	cycle_smaller	0.62	0.59	0.60	0.60	0.68	0.23
config_3	sample_larger	0.62	0.61	0.53	0.57	0.67	0.24
config_3	same_size	0.56	0.56	0.62	0.59	0.58	0.12
config_4	cycle_smaller	0.57	0.54	0.57	0.55	0.60	0.14
config_4	sample_larger	0.60	0.59	0.55	0.57	0.63	0.19
config_4	same_size	0.59	0.60	0.48	0.53	0.63	0.18
config_5	cycle_smaller	0.60	0.60	0.50	0.54	0.67	0.20
config_5	sample_larger	0.59	0.57	0.53	0.55	0.66	0.18
config_5	same_size	0.54	0.50	0.52	0.51	0.57	0.07
config_6	cycle_smaller	0.58	0.58	0.43	0.50	0.61	0.15
config_6	sample_larger	0.53	0.53	0.27	0.36	0.53	0.06
config_6	same_size	0.53	0.53	0.27	0.36	0.56	0.06
config_7	cycle_smaller	0.58	0.56	0.60	0.58	0.64	0.16
config_7	sample_larger	0.59	0.58	0.52	0.55	0.65	0.18
config_7	same_size	0.60	0.61	0.49	0.54	0.63	0.20
config_8	cycle_smaller	0.59	0.58	0.47	0.52	0.64	0.18
config_8	sample_larger	0.54	0.53	0.43	0.48	0.57	0.07
config_8	same_size	0.58	0.57	0.42	0.48	0.60	0.15
config_9	cycle_smaller	0.55	0.54	0.45	0.49	0.58	0.10

Table 5.5 – *Continued from previous page*

Config	load_mode	val_acc	val_prec	val_recall	val_f1	val_auc	val_mcc
config_9	sample_larger	0.53	0.51	0.29	0.37	0.52	0.04
config_9	same_size	0.51	0.48	0.26	0.34	0.52	0.00
config_10	cycle_smaller	0.53	0.51	0.23	0.32	0.52	0.04
config_10	sample_larger	0.55	0.53	0.26	0.35	0.56	0.07
config_10	same_size	0.54	0.53	0.29	0.37	0.54	0.07
config_11	cycle_smaller	0.58	0.56	0.58	0.57	0.60	0.17
config_11	sample_larger	0.58	0.54	0.63	0.58	0.61	0.16
config_11	same_size	0.57	0.59	0.48	0.53	0.60	0.14
config_12	cycle_smaller	0.62	0.60	0.50	0.54	0.66	0.23
config_12	sample_larger	0.61	0.59	0.52	0.55	0.65	0.20
config_12	same_size	0.57	0.52	0.57	0.54	0.60	0.13
config_13	cycle_smaller	0.63	0.62	0.57	0.59	0.67	0.25
config_13	sample_larger	0.64	0.64	0.52	0.57	0.71	0.27
config_13	same_size	0.51	0.49	0.58	0.53	0.56	0.03
config_14	cycle_smaller	0.60	0.58	0.60	0.59	0.69	0.20
config_14	sample_larger	0.63	0.63	0.59	0.61	0.69	0.25
config_14	same_size	0.58	0.56	0.59	0.57	0.61	0.16
config_15	cycle_smaller	0.52	0.49	0.54	0.51	0.54	0.04
config_15	sample_larger	0.55	0.53	0.52	0.52	0.58	0.09
config_15	same_size	0.60	0.56	0.68	0.61	0.64	0.21
config_16	cycle_smaller	0.63	0.60	0.61	0.61	0.67	0.25
config_16	sample_larger	0.61	0.57	0.64	0.60	0.66	0.22
config_16	same_size	0.59	0.55	0.51	0.53	0.62	0.17
config_17	cycle_smaller	0.59	0.58	0.54	0.56	0.65	0.18
config_17	sample_larger	0.58	0.57	0.45	0.50	0.62	0.15
config_17	same_size	0.58	0.55	0.60	0.57	0.64	0.16
config_18	cycle_smaller	0.59	0.60	0.55	0.57	0.64	0.19
config_18	sample_larger	0.63	0.65	0.54	0.59	0.68	0.27
config_18	same_size	0.57	0.57	0.41	0.47	0.57	0.13
config_19	cycle_smaller	0.65	0.69	0.57	0.62	0.73	0.31

Table 5.5 – *Continued from previous page*

Config	load_mode	val_acc	val_prec	val_recall	val_f1	val_auc	val_mcc
config_19	sample_larger	0.67	0.66	0.70	0.68	0.77	0.35
config_19	same_size	0.55	0.53	0.45	0.49	0.54	0.09
config_20	cycle_smaller	0.54	0.53	0.45	0.49	0.58	0.08
config_20	sample_larger	0.52	0.50	0.47	0.48	0.56	0.04
config_20	same_size	0.59	0.55	0.63	0.58	0.64	0.18
config_21	cycle_smaller	0.58	0.53	0.57	0.55	0.63	0.15
config_21	sample_larger	0.61	0.62	0.52	0.56	0.66	0.22
config_21	same_size	0.59	0.57	0.55	0.56	0.64	0.18
config_22	cycle_smaller	0.52	0.49	0.31	0.38	0.54	0.01
config_22	sample_larger	0.55	0.54	0.29	0.38	0.58	0.07
config_22	same_size	0.51	0.45	0.18	0.26	0.49	-0.02
config_23	cycle_smaller	0.63	0.61	0.53	0.57	0.68	0.24
config_23	sample_larger	0.58	0.58	0.49	0.53	0.62	0.15
config_23	same_size	0.57	0.56	0.55	0.55	0.61	0.14
config_24	cycle_smaller	0.64	0.62	0.61	0.62	0.71	0.29
config_24	sample_larger	0.63	0.62	0.58	0.60	0.67	0.26
config_24	same_size	0.61	0.63	0.47	0.54	0.64	0.22
config_25	cycle_smaller	0.62	0.61	0.57	0.59	0.67	0.23
config_25	sample_larger	0.57	0.52	0.57	0.54	0.63	0.14
config_25	same_size	0.61	0.62	0.49	0.55	0.65	0.22

B.3 MMD Performance on Test Set with Different Hyperparameter Configurations

Table 5.6: **MMD Test Performance.** Each row shows the test metrics (accuracy, precision, recall, F1 score, AUC, and MCC) for a given MMD configuration from Table 5.4 paired with a specified loader mode.

Config	load_mode	test_acc	test_prec	test_recall	test_f1	test_auc	test_mcc
config_1	cycle_smaller	0.59	0.57	0.48	0.52	0.62	0.16
config_1	sample_larger	0.57	0.56	0.38	0.45	0.60	0.13
config_1	same_size	0.57	0.56	0.34	0.43	0.58	0.13
config_2	cycle_smaller	0.60	0.58	0.52	0.55	0.65	0.19
config_2	sample_larger	0.60	0.59	0.50	0.54	0.64	0.20
config_2	same_size	0.60	0.60	0.40	0.48	0.65	0.18
config_3	cycle_smaller	0.58	0.55	0.55	0.55	0.64	0.16
config_3	sample_larger	0.60	0.57	0.52	0.54	0.64	0.19
config_3	same_size	0.56	0.52	0.63	0.57	0.59	0.12
config_4	cycle_smaller	0.56	0.52	0.54	0.53	0.59	0.11
config_4	sample_larger	0.58	0.55	0.53	0.54	0.62	0.15
config_4	same_size	0.60	0.59	0.48	0.53	0.62	0.19
config_5	cycle_smaller	0.60	0.58	0.52	0.55	0.66	0.20
config_5	sample_larger	0.60	0.58	0.50	0.54	0.65	0.19
config_5	same_size	0.55	0.51	0.55	0.53	0.57	0.09
config_6	cycle_smaller	0.59	0.57	0.45	0.50	0.62	0.16
config_6	sample_larger	0.53	0.48	0.28	0.35	0.55	0.02
config_6	same_size	0.54	0.50	0.27	0.35	0.55	0.05
config_7	cycle_smaller	0.60	0.57	0.60	0.58	0.65	0.20
config_7	sample_larger	0.60	0.57	0.54	0.55	0.65	0.18
config_7	same_size	0.59	0.57	0.46	0.51	0.63	0.16
config_8	cycle_smaller	0.60	0.58	0.47	0.52	0.63	0.19
config_8	sample_larger	0.55	0.52	0.46	0.49	0.58	0.09
config_8	same_size	0.58	0.56	0.43	0.48	0.61	0.14
config_9	cycle_smaller	0.57	0.54	0.49	0.52	0.58	0.13

Table 5.6 – *Continued from previous page*

Config	load_mode	test_acc	test_prec	test_recall	test_f1	test_auc	test_mcc
config_9	sample_larger	0.53	0.49	0.30	0.37	0.52	0.03
config_9	same_size	0.51	0.46	0.28	0.35	0.51	0.00
config_10	cycle_smaller	0.54	0.51	0.22	0.31	0.53	0.05
config_10	sample_larger	0.55	0.52	0.31	0.39	0.55	0.07
config_10	same_size	0.53	0.50	0.26	0.34	0.55	0.03
config_11	cycle_smaller	0.55	0.52	0.55	0.53	0.58	0.11
config_11	sample_larger	0.57	0.54	0.62	0.57	0.61	0.15
config_11	same_size	0.57	0.54	0.45	0.49	0.59	0.12
config_12	cycle_smaller	0.61	0.60	0.46	0.52	0.65	0.20
config_12	sample_larger	0.60	0.57	0.52	0.55	0.64	0.19
config_12	same_size	0.57	0.54	0.60	0.57	0.62	0.15
config_13	cycle_smaller	0.61	0.59	0.51	0.55	0.66	0.21
config_13	sample_larger	0.64	0.64	0.53	0.58	0.69	0.27
config_13	same_size	0.50	0.47	0.57	0.52	0.54	0.01
config_14	cycle_smaller	0.63	0.60	0.62	0.61	0.69	0.26
config_14	sample_larger	0.63	0.61	0.56	0.58	0.68	0.25
config_14	same_size	0.59	0.55	0.60	0.57	0.60	0.17
config_15	cycle_smaller	0.52	0.48	0.55	0.51	0.53	0.04
config_15	sample_larger	0.57	0.53	0.52	0.53	0.59	0.13
config_15	same_size	0.59	0.55	0.68	0.61	0.64	0.20
config_16	cycle_smaller	0.62	0.60	0.57	0.59	0.67	0.24
config_16	sample_larger	0.60	0.57	0.59	0.58	0.64	0.20
config_16	same_size	0.59	0.56	0.50	0.53	0.62	0.16
config_17	cycle_smaller	0.56	0.53	0.48	0.51	0.60	0.12
config_17	sample_larger	0.58	0.55	0.46	0.51	0.62	0.14
config_17	same_size	0.58	0.54	0.61	0.58	0.62	0.17
config_18	cycle_smaller	0.59	0.56	0.54	0.55	0.62	0.18
config_18	sample_larger	0.61	0.59	0.51	0.54	0.65	0.20
config_18	same_size	0.55	0.53	0.38	0.44	0.57	0.09
config_19	cycle_smaller	0.66	0.66	0.56	0.61	0.74	0.32

Table 5.6 – *Continued from previous page*

Config	load_mode	test_acc	test_prec	test_recall	test_f1	test_auc	test_mcc
config_19	sample_larger	0.67	0.63	0.68	0.65	0.75	0.33
config_19	same_size	0.52	0.48	0.43	0.46	0.52	0.03
config_20	cycle_smaller	0.55	0.52	0.43	0.47	0.58	0.09
config_20	sample_larger	0.54	0.51	0.46	0.49	0.56	0.08
config_20	same_size	0.58	0.54	0.59	0.56	0.63	0.15
config_21	cycle_smaller	0.58	0.55	0.55	0.55	0.62	0.15
config_21	sample_larger	0.59	0.57	0.48	0.52	0.63	0.17
config_21	same_size	0.62	0.59	0.58	0.58	0.65	0.23
config_22	cycle_smaller	0.52	0.47	0.32	0.38	0.52	0.01
config_22	sample_larger	0.54	0.50	0.29	0.37	0.54	0.04
config_22	same_size	0.51	0.43	0.18	0.26	0.48	-0.03
config_23	cycle_smaller	0.61	0.60	0.50	0.54	0.63	0.21
config_23	sample_larger	0.58	0.56	0.47	0.51	0.60	0.15
config_23	same_size	0.58	0.54	0.56	0.55	0.61	0.15
config_24	cycle_smaller	0.63	0.61	0.59	0.60	0.68	0.26
config_24	sample_larger	0.61	0.58	0.56	0.57	0.66	0.22
config_24	same_size	0.61	0.60	0.50	0.54	0.63	0.22
config_25	cycle_smaller	0.62	0.59	0.56	0.58	0.65	0.23
config_25	sample_larger	0.58	0.55	0.61	0.57	0.63	0.17
config_25	same_size	0.59	0.57	0.46	0.51	0.63	0.17