GCondNet: A Novel Method for Improving Neural Networks on Small High-Dimensional Tabular Data

Andrei Margeloiu
University of Cambridge
am2770@cam.ac.uk

Nikola Simidjievski
University of Cambridge
ns779@cam.ac.uk

Pietro Lio
University of Cambridge
pl219@cam.ac.uk

Mateja Jamnik
University of Cambridge
mj201@cam.ac.uk

Abstract

Neural network models often struggle with high-dimensional but small sample-size tabular datasets. One reason is that current weight initialisation methods assume independence between weights, which can be problematic when there are insufficient samples to estimate the model's parameters accurately. In such small data scenarios, leveraging additional structures can improve the model's performance and training stability. To address this, we propose GCondNet, a general approach to enhance neural networks by leveraging implicit structures present in tabular data. We create a graph between samples for each data dimension, and utilise Graph Neural Networks (GNNs) for extracting this implicit structure, and for conditioning the parameters of the first layer of an underlying predictor network. By creating many small graphs, GCondNet exploits the data's high-dimensionality, and thus improves the performance of an underlying predictor network. We demonstrate the effectiveness of our method on 9 real-world datasets, where GCondNet outperforms 15 standard and state-of-the-art methods. The results show that GCondNet is a versatile framework for injecting graph-regularisation into various types of neural networks, including MLPs and tabular Transformers.

1 Introduction

Tabular datasets are ubiquitous in scientific fields such as medicine [1, 2, 3], physics [4, 5], and chemistry [6, 7]. These datasets often have a limited number of samples but a large number of features for each sample. This is because collecting many samples is often costly or infeasible, but collecting many features for each sample is relatively easy. For example, in medicine [8, 9, 10, 11, 12, 13, 14, 15], it may be difficult to enrol a large number of patients in a clinical trial for a rare disease. However, it is relatively common to collect many features, such as measuring thousands of gene expression patterns, for each patient enrolled in the study. The resulting tabular datasets have a much larger number of features (D) than the number of samples (N).

When faced with high-dimensional tabular data, neural network models struggle to achieve strong performance [16, 17], partly because they encounter increased degrees of freedom, which results in overfitting, particularly in scenarios involving small datasets. Despite transfer learning's success in image and language tasks [18], a general transfer learning protocol is lacking for tabular data [19], and current methods [20] assume large upstream datasets and shared features, which is unsuitable for our scenarios. Consequently, we focus on improving training neural networks from scratch.

37th Table Representation Learning Workshop at NeurIPS 2023

Previous successful approaches for training models on small sample-size and high-dimensional data constrained the model's parameters to ensure that similar features have similar coefficients, as initially proposed in [21] for linear regression and later extended to neural networks [22]. In the context of applications in biomedical domains, such constraints can lead to more interpretable identification of genes (features) that are biologically relevant [21]. However, these methods require access to external application-specific knowledge graphs (e.g., gene regulatory networks) to obtain feature similarities, which in turn provide "explicit relationships" between features. But numerous tasks do not have access to such application-specific graphs. We aim to integrate a similar inductive bias, posing that performance is enhanced when similar features have similar coefficients. We accomplish this *without* relying on "explicit relationships" defined in external application-specific graphs.

We propose a novel method GCondNet (**G**raph-**Cond**itioned **Net**works) to enhance the performance of neural network predictors, such as Multi-layer Perceptrons (MLPs). The key innovation of GCondNet lies in exploiting the "implicit relationships" between *samples* by performing "soft parameter-sharing" to constrain the model's parameters in a principled manner, thereby reducing overfitting. By "implicit relationships" between samples (in tabular data), we refer to possible associations that are *not* explicitly present in the dataset. To identify these potential implicit relationships between samples, we construct many graphs between samples, one graph for each feature. We then use Graph Neural Networks (GNNs) to extract any implicit structure and condition the parameters of the first layer of an underlying predictor MLP network. By parameterising the first layer, our method ensures that *similar features have similar weights at the beginning of training*, as we also propose a decaying mechanism which allows it to revert to the MLP's performance when incorrect relationships between samples are specified. Note that GCondNet still considers the samples as independent and identically distributed (iid) at both train-time and test-time because the information from the graphs is encapsulated within the model parameters and is not used directly for prediction (extended discussion in Sec. 2.2).

We introduce two similarity-based approaches for constructing such graphs from any tabular dataset. Both approaches generate a graph for each feature in the dataset (resulting in D graphs), with each node representing a sample (totalling N nodes per graph). For instance, in a gene expression dataset, we create a unique graph of patients for each gene. Unlike other methods [22, 21, 23] that require external knowledge for constructing the graphs, our graphs can be constructed from any tabular dataset.

By constraining the model's parameters to ensure similar features have similar coefficients at the beginning of training, we show that our approach yields improved downstream performance and enhanced model robustness. One reason is that creating many small graphs effectively "transposes" the problem and makes neural network optimisation more effective because we leverage the high-dimensionality of the data to our advantage by generating many small graphs, which serve as a large training set for the GNN, which in turn computes the parameters of the MLP predictor. In addition, our approach also models a different aspect of the problem – the implicit structure between samples – which we show reduces overfitting. Our contributions are summarised as follows:

1. We propose a novel method, GCondNet, for leveraging implicit relationships between samples into neural networks (Sec. 2) to improve predictive performance on small sample-size and high-dimensional tabular data. Our method is general and can be applied to any such tabular dataset, unlike other methods that rely on external application-specific knowledge graphs.
2. We demonstrate the effectiveness of our method in a series of experiments on 9 real-world biomedical datasets. We show that GCondNet consistently outperforms an MLP with the same architecture and, in fact, outperforms all 15 state-of-the-art methods we evaluate (Sec. 3.1).
3. We show that GCondNet is robust to various graph construction methods and reverts to the MLP's performance when incorrect relationships are present (Sec. 3.2).
4. We investigate the inductive biases of GCondNet and show that they serve as an additional regularisation mechanism for reducing overfitting (Sec. 3.3).

2 Method

Problem formulation. We study tabular classification problems (although the method can be directly applied to regression too), where the data matrix $\boldsymbol{X} := [\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}]^\top \in \mathbb{R}^{N \times D}$ comprises N samples $\boldsymbol{x}^{(i)} \in \mathbb{R}^D$ of dimension D, and the labels are $\boldsymbol{y} := [y_1, ..., y_N]^\top$.

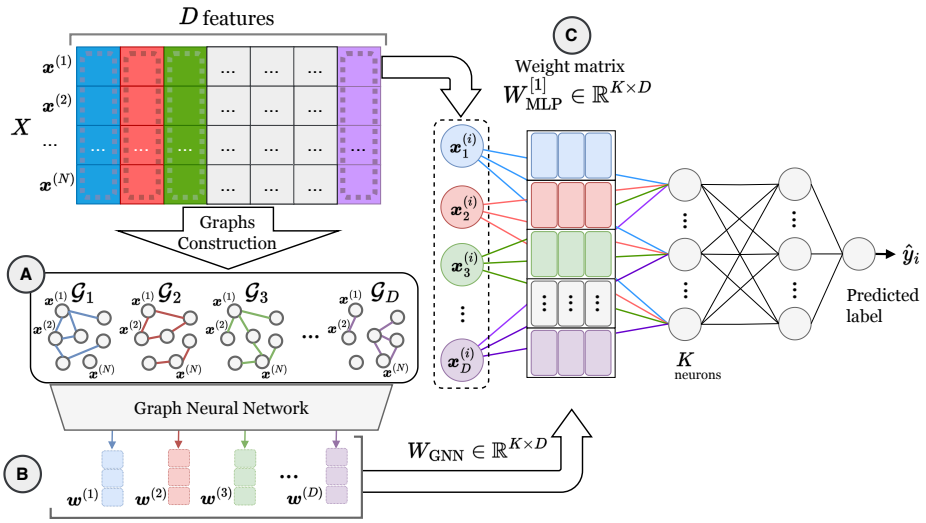

Figure 1: GCondNet is a general method for leveraging *implicit* relationships between samples to improve the performance of a standard MLP predictor network on tabular data. **(A)** Given a tabular dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, we generate a graph $\mathcal{G}_j$ for each feature in the dataset (results in D graphs), with each node representing a sample (totalling N nodes per graph). **(B)** The resulting graphs are passed to a Graph Neural Network (GNN) to extract graph embeddings $\boldsymbol{w}^{(j)} \in \mathbb{R}^K$ from each graph. We concatenate the graph embeddings into a matrix $\boldsymbol{W}_{\text{GNN}} = [\boldsymbol{w}^{(1)}, ..., \boldsymbol{w}^{(D)}]$. **(C)** We use $\boldsymbol{W}_{\text{GNN}}$ to parameterise the first layer $\boldsymbol{W}^{[1]}_{\text{MLP}}$ of the MLP predictor as a convex combination $\boldsymbol{W}^{[1]}_{\text{MLP}} = \alpha \boldsymbol{W}_{\text{GNN}} + (1-\alpha)\boldsymbol{W}_{\text{scratch}}$, where $\boldsymbol{W}_{\text{scratch}}$ is initialised to zero.

Method overview. Figure 1 presents our proposed method, which has two components: (i) a predictor network (e.g., an MLP) that takes as input a sample $\boldsymbol{x}^{(i)} \in \mathbb{R}^D$ and outputs the predicted label y_i; and (ii) a Graph Neural Network (GNN) that takes as input *fixed* graphs (D of them) and generates the parameters $\boldsymbol{W}^{[1]}_{\text{MLP}}$ for the first input layer of the predictor MLP network. The same GNN model is used for all graphs. Since these graphs are fixed for all inputs $\boldsymbol{x}^{(i)}$, GCondNet maintains the dimensionality of the input space (which remains D). Appx. C presents the pseudocode of GCondNet.

In particular, we parameterise the MLP's first layer $\boldsymbol{W}^{[1]}_{\text{MLP}}$ as a convex combination of a weight matrix $\boldsymbol{W}_{\text{GNN}}$ predicted by the GNN (by extracting the implicit structure between samples), and a weight matrix $\boldsymbol{W}_{\text{scratch}}$ initialised to zero:

$$\boldsymbol{W}^{[1]}_{\text{MLP}} = \alpha \boldsymbol{W}_{\text{GNN}} + (1-\alpha)\boldsymbol{W}_{\text{scratch}} \tag{1}$$

The mixing coefficient α determines how much the model should be conditioned on the relationships between samples learnt by the GNN. We schedule α to linearly decay $1 \rightarrow 0$ over n_α training steps, as further motivated in Sec. 2.2.

Computing $\boldsymbol{W}_{\text{GNN}}$. We train the GNN model concurrently with the MLP predictor to compute the weight matrix $\boldsymbol{W}_{\text{GNN}}$. To do this, we use the training split of $\boldsymbol{X}$ to generate a graph $\mathcal{G}_j = (\mathcal{V}_j, \mathcal{E}_j)$ for each feature in the dataset (resulting in D graphs), with each node representing a sample (totalling N nodes per graph). For example, in a gene expression dataset, one graph of patients is created for each gene. This way, we take advantage of high-dimensional data by creating many graphs to train the GNN. The graphs stay *fixed* during training, and the graph embeddings $\boldsymbol{w}^{(j)}$ are computed in parallel. We describe the graph construction in Sec. 2.1 and investigate the impact of this choice in Sec. 3.2.

At each training iteration, we use the GNN to extract graph embeddings from these graphs, as presented in Algorithm 1. For each of the D graphs, we first apply the GNN to obtain node embeddings of size K for all nodes. We then compute graph embeddings $\boldsymbol{w}^{(j)} \in \mathbb{R}^K$ by using a permutation invariant function f_{agg} to aggregate the node embeddings. Thus, the graph em-

Algorithm 1 Computing $\boldsymbol{W}_{\text{GNN}}$.

for each feature $j = 1, 2, ..., D$ **do**
  node_embeddings = GNN($\mathcal{V}_j, \mathcal{E}_j$) {$\mathcal{V}_j$ represents the nodes, and $\mathcal{E}_j$ represents the edges of the j-th feature graph}
  $\boldsymbol{w}^{(j)}$ = f_{agg}(node_embeddings) {Aggregate all node embeddings to obtain the graph embedding $\boldsymbol{w}^{(j)} \in \mathbb{R}^K$}
end for
$\boldsymbol{W}_{\text{GNN}} = [\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, ..., \boldsymbol{w}^{(D)}]$ {Concatenation}

beddings are also of size K, which is independent of the number of nodes in the graphs. These embeddings are then concatenated horizontally to form the weight matrix $\boldsymbol{W}_{\text{GNN}} = [\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, ..., \boldsymbol{w}^{(D)}]$. Finally, we use the resulting matrix $\boldsymbol{W}_{\text{GNN}}$ to parameterise the first layer of the underlying MLP predictor network that outputs the final prediction, as shown in Equation 1.

Test-time inference. The GNN and the associated graphs are only used during training. Upon training completion, the predictor MLP retains its final weights, rendering the GNN and graphs unnecessary for test inference. Post-training, test samples are processed solely by the predictor MLP.

2.1 Graphs Construction from Tabular Data

For each feature j, we create a graph $\mathcal{G}_j = (\mathcal{V}_j, \mathcal{E}_j)$ using *only* the values $\boldsymbol{X}_{:,j}$ of that feature. This enables the use of simple distance metrics between nodes and eliminates the need to work in a high-dimensional space, where distances can be inaccurate. The graph construction time is negligible, taking five seconds to generate all graphs for each task.

Node features. The nodes $\mathcal{V}_j$ of each graph represent the N training samples. The node features are one-hot encoded vectors, with the feature value for a sample located in the corresponding position in the one-hot encoding. For instance, if the feature values of three training samples are $\boldsymbol{X}_{:,j} = [0.2, 0.4, -0.5]$, then the first node's features would be $[0.2, 0, 0]$, the second node's features would be $[0, 0.4, 0]$, and the third node's features would be $[0, 0, -0.5]$.

Edges. We propose two similarity-based methods for constructing the edges between samples from tabular data, which assume that similar samples should be connected: (i) **KNN graphs** connect each node with the closest K neighbours (we set $K = 5$ in this paper), enabling GCondNet to scale linearly time-wise and memory-wise w.r.t. the sample size N; (ii) **Sparse Relative Distance (SRD) graphs** connect a sample to all samples with a feature value within a specified distance. This process creates a network topology where nodes with common feature values have more connections, thus we use an accept-reject step to sparsify the graph (all details are included in Appx. B).

2.2 Rationale for Model Architecture

What are the inductive biases of GCondNet? By parameterising the first layer, our method ensures that *similar features have similar weights at the beginning of training*. For example, if two features i and j are expressed similarly across samples, their graph embeddings $\boldsymbol{w}^{(i)}$ and $\boldsymbol{w}^{(j)}$ will be similar. As the training progresses and α decreases, the network adapts and all weights in $\boldsymbol{W}^{[1]}_{\text{MLP}}$ are updated independently, providing more flexibility in the learning process.

Why is GCondNet appropriate when $D >> N$? On small sample-size and high-dimensional data, conventional neural approaches (such as an MLP) tend to exhibit unstable training behaviour and/or converge poorly – one reason is a large degree of freedom for the small datasets. This happens because; (i) the number of parameters in the first layer is proportional to the number of features; and (ii) modern weight initialisation techniques [24, 25] assume independence between the parameters within a layer. Although the independence assumption may work well with large datasets, as it allows for flexibility, it can be problematic when there are too few samples to estimate the model's parameters accurately (as we show in Sec. 3.2). GCondNet is designed to mitigate these training instabilities: (i) by constraining the model's degrees of freedom via an additional GNN that outputs the model's first layer, which includes most of the learning parameters; and (ii) by providing a more principled weight initialisation on the model's first layer (because at the start we have $\boldsymbol{W}^{[1]}_{\text{MLP}} = \boldsymbol{W}_{\text{GNN}}$). GCondNet indirectly addresses the curse of dimensionality by enhancing model optimisation rather than directly reducing the input space dimensionality, which remains D.

Why do we create many small graphs? The conventional approach would be to have one large graph where nodes are features and let $\boldsymbol{w}^{(j)}$ be node embeddings. In contrast, we generate many small graphs and compute $\boldsymbol{w}^{(j)}$ as graph embeddings. Our approach offers several advantages: (i) Having multiple graphs "transposes" the problem and uses the high-dimensionality of the data to our advantage by generating many small graphs which serve as a large training set for the GNN. (ii) It allows using simple distance metrics because the nodes contain only scalar values; in contrast, taking distances between features would require working in a high-dimensional space and encountering the curse of dimensionality. (iii) Flexibility, as it can incorporate external knowledge graphs, if available, by forming hyper-graphs between similar feature graphs. (iv) The computation is efficient because the graphs are small due to small-size datasets.

Does GCondNet still consider the samples as iid? Yes, at both train-time and test-time. Recall that samples are iid if they are independent, conditioned on the model's parameters θ [26], so that $p\left(y_1, y_2 \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \theta\right) = p\left(y_1 \mid \boldsymbol{x}^{(1)}, \theta\right) \cdot p\left(y_2 \mid \boldsymbol{x}^{(2)}, \theta\right)$. As all information extracted from our sample-wise graphs is encapsulated within the model parameters, the above iid equation holds for GCondNet. Note that our graphs are *not* used directly to make predictions, such as distance-based models (e.g., KNN) do; to make a prediction, input samples are exclusively processed by the predictor MLP, which uses the same model parameters across all samples.

Why decay the mixing coefficient α? Decaying α introduces flexibility in the learning process, by enabling the model to start training initialised with the GNN-extracted structure and later adjust the weights more autonomously as training advances. Since the true relationships between samples are unknown, the GNN-extracted structure may be noisy or suboptimal for parameterising the model. At the start, $\alpha = 1$ and the first layer is entirely determined by the GNN ($\boldsymbol{W}^{[1]}_{\text{MLP}} = \boldsymbol{W}_{\text{GNN}}$). After α becomes 0, the model trains as a standard MLP (the GNN is disabled), but its parameters have been impacted by our proposed method and will reach a distinct minimum (evidenced in Sec. 3.1 by GCondNet consistently outperforming an equivalent MLP). In contrast to our decaying of the mixing coefficient α, maintaining α fixed (similar to PLATO's [22] inductive bias) leads to unstable training (see our experiments in Sec. 3.3). Moreover, if α were fixed, it would need optimisation like other hyperparameters, while by decaying α, we avoid this time-consuming tuning.

3 Experiments

First, we quantitatively evaluate our model against 15 benchmark models (Sec. 3.1). We then evaluate the impact of the graph type on GCondNet (Sec. 3.2). Finally, we analyse the inductive biases of our method and their effect on optimisation (Sec. 3.3).

Datasets. We focus on classification tasks using small-sample and high-dimensional datasets and consider 9 real-world tabular biomedical datasets ranging from $100 - 200$ samples, $3312 - 19993$ features, and $2 - 4$ classes. We specifically keep the datasets small to mimic practical scenarios where data is limited. See Appx. D for details on the datasets.

Evaluation. We evaluate all models using a 5-fold cross-validation repeated 5 times, resulting in 25 runs per model. We report the mean $\pm$ std of the test balanced accuracy averaged across all 25 runs. To ensure comprehensive analysis, we perform both a paired t-test and a Wilcoxon signed-rank test, and we correct the variance for the multiple repeats and cross-validation (see Appx. E for details and p-values). To summarise the results in the manuscript, we rank the methods by their mean relative performance on each dataset, which is similar to signed-rank statistical tests (such as Wilcoxon). If two methods have the same accuracy (rounded to two decimals), they obtain the same per-dataset rank. The final rank for each method is the average of their per-dataset ranks.

GCondNet architecture and settings. The predictor MLP is a 3-layer feed-forward neural network with $100, 100, 10$ neurons. The GNN within GCondNet is a two-layer Graph Convolutional Network (GCN) [27]. The permutation invariant function f_{agg} for computing graph embeddings is global average pooling. We decay the mixing coefficient α over $n_\alpha = 200$ training steps, although we found that GCondNet is robust to the number of steps n_α, as supported by the statistical tests in Appx. F. We choose the graph construction method (between KNN and SRD) using the validation balanced accuracy (Appx. G presents the validation and test accuracy for different graph types). Due to limited space, we provide complete reproducibility details in Appx. H, including the training protocol and final settings for all methods. Training times for GCondNet are in Appx. I.

Benchmark methods. We evaluate 15 benchmark models, encompassing a standard MLP and modern methods typically employed for small sample-size and high-dimensional datasets, such as DietNetworks [28], FsNet [29], SPINN [17], DNP [16], and WPFS [30], all of which use the same architecture as GCondNet for a fair comparison. We also include contemporary neural methods for tabular data, like TabNet [31], TabTransformer [32], Concrete Autoencoders (CAE) [33], and LassoNet[1] [34], and standard methods such as ElasticNet [35], Random Forest [36] and LightGBM [37].

We also compare the performance of GCondNet with GNNs on tabular data where relationships between samples are not explicitly provided. In particular, we evaluate Graph Convolutional Network (GCN) [27] and Graph Attention Network v2 (GATv2) [38]. To ensure fairness, we employ a

[1]We discuss LassoNet training instabilities in Appx. H.

Dataset	lung	toxicity	metabric-p50	metabric-dr	prostate	cll	smk	tcga-survival	tcga-tumor	Avg. rank
MLP	94.20 ± 4.9	93.21 ± 6.1	94.31 ± 5.3	59.56 ± 5.5	88.76 ± 5.5	78.30 ± 8.9	64.42 ± 8.4	56.28 ± 6.7	48.19 ± 7.7	$6.88\star$
DietNetworks	90.43 ± 6.2	82.13 ± 7.4	95.02 ± 4.7	56.98 ± 8.7	81.71 ± 11.0	68.84 ± 9.2	62.71 ± 9.3	53.62 ± 5.4	46.69 ± 7.1	$10.62\star$
FsNet	91.75 ± 3.0	60.26 ± 8.1	83.86 ± 8.1	56.92 ± 10.1	84.74 ± 9.8	66.38 ± 9.2	56.27 ± 9.2	53.83 ± 7.9	45.94 ± 9.8	$11.75\star$
DNP	92.83 ± 5.6	93.50 ± 6.1	93.56 ± 5.5	55.79 ± 7.0	90.25 ± 5.9	85.12 ± 5.4	66.89 ± 7.6	58.13 ± 8.2	44.71 ± 5.9	7.38
SPINN	92.26 ± 6.6	93.50 ± 4.8	93.56 ± 5.5	56.13 ± 7.2	89.27 ± 5.9	85.34 ± 5.4	68.43 ± 7.9	57.70 ± 7.0	44.28 ± 6.8	7.19
WPFS	94.83 ± 4.2	88.29 ± 5.2	95.96 ± 4.1	59.05 ± 8.6	89.15 ± 6.7	79.14 ± 4.4	66.89 ± 6.2	59.54 ± 6.9	55.91 ± 8.5	4.00
TabNet	77.65 ± 12.9	40.06 ± 11.3	83.60 ± 11.4	49.18 ± 9.6	65.66 ± 14.7	57.81 ± 9.9	54.57 ± 8.7	51.58 ± 9.9	39.34 ± 7.9	$14.88\star$
TabTransformer	94.03 ± 4.7	87.67 ± 6.1	93.82 ± 4.7	52.49 ± 9.0	85.96 ± 11.5	76.81 ± 6.8	64.00 ± 9.2	56.91 ± 5.6	40.70 ± 6.9	$9.62\star$
CAE	85.00 ± 5.0	60.36 ± 11.2	95.78 ± 3.6	57.35 ± 9.3	87.60 ± 7.8	71.94 ± 13.4	59.96 ± 10.9	52.79 ± 8.3	40.69 ± 7.3	$9.31\star$
LassoNet	25.11 ± 9.8	26.67 ± 8.6	48.81 ± 10.8	48.88 ± 5.7	54.78 ± 10.5	30.63 ± 8.6	51.04 ± 8.5	46.08 ± 9.2	33.49 ± 7.5	$16.00\star$
ElasticNet	95.19 ± 3.7	94.32 ± 4.8	95.98 ± 2.6	58.23 ± 9.6	91.36 ± 6.1	84.35 ± 7.3	70.36 ± 8.5	55.88 ± 5.7	50.73 ± 7.9	4.06
Random Forest	91.81 ± 6.9	80.75 ± 6.7	89.11 ± 6.5	51.38 ± 3.7	90.78 ± 7.1	82.06 ± 6.5	68.16 ± 7.5	61.30 ± 6.0	50.93 ± 8.4	6.62
LightGBM	93.42 ± 5.9	82.40 ± 6.4	94.97 ± 5.1	58.23 ± 8.5	91.38 ± 5.7	85.59 ± 6.5	65.70 ± 7.4	57.08 ± 7.8	49.11 ± 10.3	5.06
GCN	93.29 ± 4.6	76.13 ± 7.0	91.12 ± 8.6	58.28 ± 7.3	82.59 ± 12.4	71.99 ± 8.3	65.62 ± 8.0	58.31 ± 5.7	51.01 ± 8.1	$7.75\star$
GATv2	93.33 ± 6.2	76.65 ± 11.2	86.95 ± 8.2	54.71 ± 7.1	83.23 ± 10.5	57.74 ± 14.1	66.06 ± 8.2	53.60 ± 6.8	45.45 ± 9.3	$11.25\star$
GCondNet (ours)	95.34 ± 4.4	95.25 ± 4.5	96.37 ± 3.9	59.34 ± 8.9	90.37 ± 5.5	80.69 ± 5.4	68.08 ± 7.3	56.36 ± 9.4	51.69 ± 8.8	**3.62**

Table 1: Classification performance of GCondNet and 15 benchmark models on 9 real-world datasets. We report the mean $\pm$ std of the test balanced accuracy averaged over 25 runs, and also compute each method's average rank across datasets, where a higher rank implies higher accuracy in general. The $\star$ means that GCondNet statistically outperforms the method based on the Wilcoxon signed-rank test at $\alpha = 0.05$ (see Appx. E). Overall, GCondNet ranks best compared to the other benchmark models.

similar setup to GCondNet, constructing a KNN-based graph in which each node represents a sample connected to its five nearest samples based on cosine similarity, which is well-suited for high-dimensional data. Both GCN and GATv2 are trained in a transductive setting, incorporating test sample edges during training while masking nodes to prevent data leakage.

3.1 How Accurate is GCondNet?

Our experiments in Table 1 show that GCondNet outperforms all 15 benchmark models on average, achieving a better overall rank across 9 real-world datasets – suggesting its effectiveness across diverse datasets. GCondNet is followed by WPFS, a method employed for small sample-size and high-dimensional datasets, although GCondNet consistently outperforms it on seven out of nine tasks, providing improvements of up to 7%. Standard methods like ElasticNet, Random Forest, and LightGBM are competitive; however, their relative performance is sometimes highly dataset-dependent.

GCondNet demonstrates consistent improvements compared to a standard MLP, outperforming it with statistical significance. Even though, in some cases, these improvements might be modest, they prove valuable for the problems we tackle, specifically learning from small sample-size and high-dimensional data, underscoring the effectiveness of our method.

We also find that GCondNet outperforms other methods specialised for this data scenario by a large margin, such as DietNetworks, FsNet, SPINN, DNP, and more complex neural architectures for tabular data such as TabNet, TabTransformer, CAE and LassoNet. This finding aligns with recent research [39], suggesting that well-regularised MLPs are more effective at handling tabular data compared to intricate architectures. Because the MLP baseline we used was already well-regularised, we attribute GCondNet's performance improvement to its additional regularisation effect which reduces overfitting (examined in Sec. 3.2 and 3.3) and its robustness to the graph construction method (Sec. 3.2).

Finally, we compare against GNNs on tabular data where relationships between samples are not explicitly provided. Despite the advantage of GCN and GATv2 of being trained in a transductive setting, GCondNet outperforms both methods across all tasks. The performance gap ranges from 23% on 'cll' and 19% on 'toxicity' to more than 5% on three other tasks. This indicates that models heavily reliant on *latent* structure present in tabular data, such as GNNs, are particularly sensitive to misspecifications during model construction. In contrast, GCondNet demonstrates resilience against such misspecifications, which we analyse in the following section.

3.2 The Influence of the Weight Initialisation in MLP and the Improvement via GCondNet

Our central hypothesis is that, on small datasets, a more effective weight initialisation scheme – such as the one introduced in GCondNet – improves the accuracy of neural networks. We believe that the independence assumptions in typical weight initialisation schemes, such as the Kaiming initialisation [25], may favour overfitting on small datasets because they allow for too much flexibility.

$\triangleright$ *How does the graph type influence the performance of GCondNet relative to an equivalent MLP?*

In addition to the SRD and KNN methods explained in Sec. 2.1, we also consider random graphs between samples for the GNN in GCondNet (full details for creating random graphs are in Appx. J).

	MLP	MLP with specialised initialisations			GCondNet with different graph types		
		PCA	NMF	WL	SRD	KNN	Random
lung	94.20 ± 4.9	96.04 ± 4.0	95.05 ± 4.0	94.56 ± 5.9	95.34 ± 4.4	94.68 ± 4.2	94.86 ± 4.5
toxicity	93.21 ± 6.1	92.58 ± 5.4	89.11 ± 5.9	92.49 ± 5.7	95.25 ± 4.5	95.22 ± 3.9	95.06 ± 6.1
metabric-p50	94.31 ± 5.3	94.70 ± 4.9	95.09 ± 4.8	95.81 ± 5.0	96.26 ± 3.7	96.37 ± 3.9	95.86 ± 4.2
metabric-dr	59.56 ± 5.5	55.75 ± 8.2	59.36 ± 6.8	58.69 ± 7.3	58.24 ± 6.3	59.34 ± 8.9	57.89 ± 8.7
prostate	88.76 ± 5.5	91.04 ± 5.0	89.36 ± 6.4	89.97 ± 5.9	89.95 ± 6.1	90.37 ± 5.5	89.56 ± 6.3
cll	78.30 ± 8.9	79.92 ± 6.4	78.59 ± 6.6	79.98 ± 6.5	81.54 ± 7.1	80.69 ± 5.4	81.36 ± 5.7
smk	64.42 ± 8.4	66.79 ± 10.8	65.87 ± 7.3	64.47 ± 8.1	68.08 ± 7.3	65.92 ± 8.6	66.13 ± 8.1
tcga-survival	56.28 ± 6.7	55.22 ± 7.3	60.08 ± 6.1	54.79 ± 8.2	56.36 ± 9.4	58.61 ± 7.0	58.31 ± 7.8
tcga-tumor	48.19 ± 7.7	50.95 ± 10.5	51.49 ± 9.7	49.67 ± 8.8	52.42 ± 7.5	51.69 ± 8.8	51.57 ± 9.1
Average rank	5.77	4.22	4.33	5.11	**2.33**	**2.66**	3.55

Table 2: We evaluate the robustness of GCondNet on various graph types and compare it to an identically structured MLP initialised with three weight initialisation methods emulating the inductive biases of GCondNet, but without training a GNN. We report the mean $\pm$ std of the test balanced accuracy averaged over 25 runs and include statistical tests in Appx. E. Results show that GCondNet is robust to the graph type, and consistently outperforms a standard MLP across all graph types. Further, GCondNet often outperforms other initialisation methods, highlighting the effectiveness of the GNN-extracted latent structure.

We train GCondNet on 25 different data splits, and for each split, we sample random graphs five times – resulting in 125 trained models on random graphs.

Table 2 shows that GCondNet, with *any* of the three types of graphs, outperforms an MLP of the same architecture on eight out of nine datasets, and the improvement is statistically significant (Appx. E). This supports our hypothesis that incorporating sample-wise structure improves performance.

$\triangleright$ *Does GCondNet surpass the performance of other initialisation schemes that do not use GNNs?*

We investigate other weight initialisation schemes for imbuing the inductive biases of similar features having similar coefficients (as proposed by 21, 22). To the best of our knowledge, all such existing initialisation methods necessitate external knowledge, like 21, 22. Consequently, *we propose* three weight initialisation schemes incorporating a similar inductive bias, but without training GNNs. See Appx. J for details on these initialisation methods. While the specialised initialisation schemes (PCA, NMF, WL) occasionally outperform a standard MLP (with Kaiming initialisation), we generally find that GCondNet further enhances these results. For instance, GCondNet with KNN graphs consistently surpasses the graph-based Weisfeiler-Lehman (WL) initialisation across all datasets, with performance increases reaching 3.5%. This underscores the efficacy of training GNNs to distil structure from sample-wise relationships.

$\triangleright$ *How robust is GCondNet to different graphs?*

The three graph types exhibit similar absolute performance with statistical significance (see Appx. E), making GCondNet resilient to misspecifications during graph construction. However, the optimal graph construction method is task-dependent: SRD excels in four datasets, KNN in another four, and Random graphs in one. We believe one *limitation* of our work is the lack of an optimal graph construction method. However, we anticipated this outcome because: (i) diverse tasks may necessitate modelling distinct types of structure; and (ii) the GNN's oversmoothing issue [40] can lead to computing just an "average" of the feature values.

3.3 The Impact of GCondNet's Inductive Biases

We analyse the inductive biases of our method, which ensures that similar features have similar weights at the beginning of training. To isolate this effect, we train two versions of GCondNet: (i) with *decaying* α, and (ii) a modified version with a *fixed* mixing coefficient α throughout the training process. As $\alpha \to 0$, the model becomes equivalent to an MLP, and as $\alpha \to 1$, the first layer is conditioned on the GNN-extracted structure.

We find that incorporating structure into the model serves as a regularisation mechanism that can prevent overfitting. Figure 2 shows that an MLP (equivalent to $\alpha = 0$) begins overfitting at around the $4{,}000^{\text{th}}$ iteration. In contrast, all models incorporating the structure between samples (i.e., $\alpha > 0$) avoid this issue and attain better validation loss.

We also find that using a fixed α during training leads to a test-time performance drop of at least 2% than the decaying version (on 'toxicity', presented in Figure 2), and to training instability, as seen by the high variance in the training loss. We posit this occurs because the model is overly constrained on

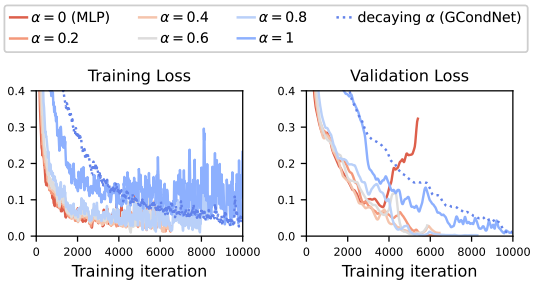

Figure 2: The impact of varying the mixing coefficient α is illustrated through the training and validation loss curves (averaged over 25 runs) on 'toxicity'. We train GCondNet with linearly *decaying* α and with *fixed* α. Two observations are notable: (i) GCondNet exhibits less overfitting than an MLP ($\alpha = 0$), which overfits at the $4{,}000^{\text{th}}$ iteration; (ii) decaying α enhances the training stability while improving the test-time accuracy by at least 2%.

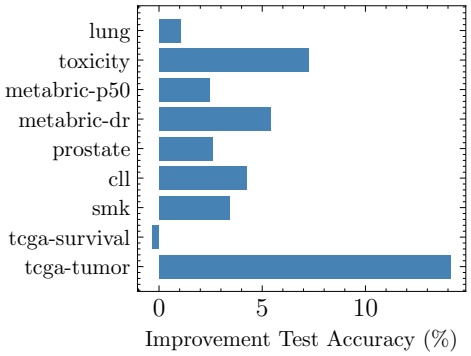

Figure 3: GCondNet is versatile and can enhance various models beyond just MLPs. When applied to TabTransformer, GCondNet consistently improves performance by up to 14% (averaged across 25 runs).

potentially incorrect graphs without sufficient learning capacity for other weights. By decaying α from $1 \rightarrow 0$, the model gains flexibility, achieving better generalisation and increased stability.

Extension to Transformers: We highlight that GCondNet is a general framework for injecting graph-regularisation into various types of neural networks, and it can readily be applied to other architectures beyond an MLP. As a proof-of-principle, we apply GCondNet to TabTransformer [32], leading to consistent performance improvements by up to 14% (Figure 3).

4 Related Work

"Diet" methods such as DietNetworks [28], FsNet [29], and WPFS [30] use auxiliary networks to reduce the number of parameters in the first layer of MLPs. Although designed for small and high-dimensional tabular datasets, the performance of these methods is highly sensitive to the choice of feature embedding. In contrast, GCondNet harnesses data's implicit sample relationships to derive feature embeddings through learned graph embeddings $\boldsymbol{w}^{(i)}$. Of these methods, GCondNet shares similarities with PLATO [22], which also uses GNNs as an auxiliary network. Key differences include: (i) PLATO requires an external domain-specific knowledge graph, while GCondNet is general and can be applied to any tabular dataset; and (ii) PLATO uses one graph between features, while GCondNet builds multiple graphs between samples and leverages the implicit sample-wise relationships. Refer to Appendix A for an extended discussion on the difference to PLATO.

Graph-based approaches for tabular data, including semi-supervised approaches, construct a graph between samples to capture the underlying relationships. Our method differs from recent GNN-based techniques in three key ways. (i) We apply GNNs *only during training* to improve an MLP predictor, unlike other methods [41, 42, 43, 44, 45] that use GNNs for inference. (ii) GCondNet generates many graphs between samples (one for each feature) and extracts graph embeddings to parameterise a predictor network. This approach is novel and differs from other work [41, 42, 43], which structure graphs differently. (iii) Common graph-based methods make additional, potentially suboptimal assumptions. For instance, [46, 47, 44] depend on the *smoothness assumption*, which can be ill-suited for high-dimensional data. Moreover, [45] requires an additional large meta-training, while [44] focuses on datasets with just up to 30 features. See Appx. A for an extended discussion on related work.

5 Conclusion

We introduce GCondNet, a general method to improve neural network predictors on small and high-dimensional tabular datasets. The key innovation of GCondNet lies in exploiting the "implicit relationships" between *samples* by performing "soft parameter-sharing" to constrain the model's parameters. Our method is general and can be applied to any tabular dataset. We evaluate 9 classification tasks from biomedical datasets – in real applications, this could mean identifying biomarkers for different diseases – and show that GCondNet outperforms 15 benchmark methods while being robust to different graph construction methods.

Acknowledgements

The authors would like to thank Mateo Espinosa Zarlenga, Ramon Viñas, Pietro Barbiero, Iulia Duta and Cristian Bodnar for their valuable comments and insightful discussions on earlier versions of this manuscript. We acknowledge the support from the Cambridge ESRC Doctoral Training Partnership for AM, and from the Cancer Research UK Cambridge Centre [CTRQQR-2021/100012] for NS.

References

[1] Lisiane B Meira, Antonio MC Reis, David L Cheo, Dorit Nahari, Dennis K Burns, and Errol C Friedberg. Cancer predisposition in mutant mice defective in multiple genetic pathways: uncovering important genetic interactions. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, 477(1-2):51–58, 2001.

[2] Rubika Balendra and Adrian M Isaacs. C9orf72-mediated als and ftd: multiple pathways to disease. *Nature Reviews Neurology*, 14(9):544–558, 2018.

[3] Matthew Kelly and Christopher Semsarian. Multiple mutations in genetic cardiovascular disease: a marker of disease severity? *Circulation: Cardiovascular Genetics*, 2(2):182–190, 2009.

[4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):4308, 2014.

[5] Gregor Kasieczka, Benjamin Nachman, David Shih, Oz Amram, Anders Andreassen, Kees Benkendorfer, Blaz Bortolato, Gustaaf Brooijmans, Florencia Canelli, Jack H Collins, et al. The lhc olympics 2020 a community challenge for anomaly detection in high energy physics. *Reports on progress in physics*, 84(12):124201, 2021.

[6] Zenan Zhai, Christian Druckenbrodt, Camilo Thorne, Saber A Akhondi, Dat Quoc Nguyen, Trevor Cohn, and Karin Verspoor. Chemtables: a dataset for semantic classification on tables in chemical patents. *Journal of Cheminformatics*, 13(1):1–20, 2021.

[7] John A Keith, Valentin Vassilev-Galindo, Bingqing Cheng, Stefan Chmiela, Michael Gastegger, Klaus-Robert Muller, and Alexandre Tkatchenko. Combining machine learning and computational chemistry for predictive insights into chemical systems. *Chemical reviews*, 121(16): 9816–9872, 2021.

[8] Wanjuan Yang, Jorge Soares, Patricia Greninger, Elena J Edelman, Howard Lightfoot, Simon Forbes, Nidhi Bindal, Dave Beare, James A Smith, I Richard Thompson, et al. Genomics of drug sensitivity in cancer (gdsc): a resource for therapeutic biomarker discovery in cancer cells. *Nucleic acids research*, 41(D1):D955–D961, 2012.

[9] Hui Gao, Joshua M Korn, Stéphane Ferretti, John E Monahan, Youzhen Wang, Mallika Singh, Chao Zhang, Christian Schnell, Guizhi Yang, Yun Zhang, et al. High-throughput screening using patient-derived tumor xenografts to predict clinical trial drug response. *Nature medicine*, 21(11):1318–1325, 2015.

[10] Francesco Iorio, Theo A Knijnenburg, Daniel J Vis, Graham R Bignell, Michael P Menden, Michael Schubert, Nanne Aben, Emanuel Gonçalves, Syd Barthorpe, Howard Lightfoot, et al. A landscape of pharmacogenomic interactions in cancer. *Cell*, 166(3):740–754, 2016.

[11] Mathew J Garnett, Elena J Edelman, Sonja J Heidorn, Chris D Greenman, Anahita Dastur, King Wai Lau, Patricia Greninger, I Richard Thompson, Xi Luo, Jorge Soares, et al. Systematic identification of genomic markers of drug sensitivity in cancer cells. *Nature*, 483(7391):570–575, 2012.

[12] Avrum Spira, Jennifer E Beane, Vishal Shah, Katrina Steiling, Gang Liu, Frank Schembri, Sean Gilman, Yves-Martine Dumas, Paul Calner, Paola Sebastiani, et al. Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. *Nature medicine*, 13(3):361–366, 2007.

[13] Gagan Bajwa, Ralph J DeBerardinis, Baomei Shao, Brian Hall, J David Farrar, and Michelle A Gill. Cutting edge: Critical role of glycolysis in human plasmacytoid dendritic cell antiviral responses. *The Journal of Immunology*, 196(5):2004–2009, 2016.

[14] Christina Curtis, Sohrab P Shah, Suet-Feung Chin, Gulisa Turashvili, Oscar M Rueda, Mark J Dunning, Doug Speed, Andy G Lynch, Shamith Samarajiwa, Yinyin Yuan, et al. The genomic and transcriptomic architecture of 2,000 breast tumours reveals novel subgroups. *Nature*, 486 (7403):346–352, 2012.

[15] Katarzyna Tomczak, Patrycja Czerwińska, and Maciej Wiznerowicz. The cancer genome atlas (tcga): an immeasurable source of knowledge. *Contemporary oncology*, 19(1A):A68, 2015.

[16] Bo Liu, Ying Wei, Yu Zhang, and Qiang Yang. Deep neural networks for high dimension, low sample size data. In *International Joint Conference on Artificial Intelligence*, pages 2287–2293, 2017.

[17] Jean Feng and Noah Simon. Sparse-input neural networks for high-dimensional nonparametric regression and classification. *arXiv preprint arXiv:1711.07592*, 2017.

[18] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*, pages 270–279. Springer, 2018.

[19] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[20] Roman Levin, Valeriia Cherepanova, Avi Schwarzschild, Arpit Bansal, C. Bayan Bruss, Tom Goldstein, Andrew Gordon Wilson, and Micah Goldblum. Transfer learning with deep tabular models. *International Conference on Learning Representations*, 2023.

[21] Caiyan Li and Hongzhe Li. Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24(9):1175–1182, 2008.

[22] Camilo Ruiz, Hongyu Ren, Kexin Huang, and Jure Leskovec. Tabular deep learning when $d >> n$ by using an auxiliary knowledge graph. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

[23] Paul Scherer, Maja Trebacz, Nikola Simidjievski, Ramon Viñas, Zohreh Shams, Helena Andres Terre, Mateja Jamnik, and Pietro Liò. Unsupervised construction of computational graphs for gene expression data with explicit structural inductive biases. *Bioinformatics*, 38(5):1320–1327, 2022.

[24] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[26] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[27] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *Internation Conference on Learning Representa*, 2017.

[28] Adriana Romero, Pierre Luc Carrier, Akram Erraqabi, Tristan Sylvain, Alex Auvolat, Etienne Dejoie, Marc-André Legault, Marie-Pierre Dubé, Julie G. Hussin, and Yoshua Bengio. Diet networks: Thin parameters for fat genomics. In *International Conference on Learning Representations*, 2017.

[29] Dinesh Singh, Héctor Climente-González, Mathis Petrovich, Eiryo Kawakami, and Makoto Yamada. Fsnet: Feature selection network on high-dimensional biological data. *arXiv preprint arXiv:2001.08322*, 2020.

[30] Andrei Margeloiu, Nikola Simidjievski, Pietro Lio, and Mateja Jamnik. Weight predictor network with feature selection for small sample tabular biomedical data. *AAAI Conference on Artificial Intelligence*, 2023.

[31] Sercan O Arık and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

[32] Xin Huang, Ashish Khetan, Milan W. Cvitkovic, and Zohar S. Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv*, abs/2012.06678, 2020.

[33] Muhammed Fatih Balın, Abubakar Abid, and James Zou. Concrete autoencoders: Differentiable feature selection and reconstruction. In *International conference on machine learning*, pages 444–453. PMLR, 2019.

[34] Ismael Lemhadri, Feng Ruan, and Rob Tibshirani. Lassonet: Neural networks with feature sparsity. In *International Conference on Artificial Intelligence and Statistics*, pages 10–18. PMLR, 2021.

[35] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 2005.

[36] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[37] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30, 2017.

[38] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

[39] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in Neural Information Processing Systems*, 34, 2021.

[40] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3438–3445, 2020.

[41] Jiaxuan You, Xiaobai Ma, Yi Ding, Mykel J Kochenderfer, and Jure Leskovec. Handling missing data with graph representation learning. *Advances in Neural Information Processing Systems*, 33:19075–19087, 2020.

[42] Qitian Wu, Chenxiao Yang, and Junchi Yan. Towards open-world feature extrapolation: An inductive graph learning approach. *Advances in Neural Information Processing Systems*, 34: 19435–19447, 2021.

[43] Kounianhua Du, Weinan Zhang, Ruiwen Zhou, Yangkun Wang, Xilong Zhao, Jiarui Jin, Quan Gan, Zheng Zhang, and David Paul Wipf. Learning enhanced representations for tabular data via neighborhood propagation. *Advances in Neural Information Processing Systems*, 35, 2022.

[44] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34:22667–22681, 2021.

[45] Victor Garcia Satorras and Joan Bruna. Few-shot learning with graph neural networks. *International Conference on Learning Representations*, 2018.

[46] Anees Kazi, Luca Cosmo, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael M Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1606–1617, 2022.

[47] Kaixiong Zhou, Zirui Liu, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Table2graph: Transforming tabular data to unified weighted graph. In *International Joint Conference on Artificial Intelligence*, 2022.

[48] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[49] Liran Katzir, Gal Elidan, and Ran El-Yaniv. Net-dnf: Effective deep modeling of tabular data. In *International Conference on Learning Representations*, 2020.

[50] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *International Conference on Machine Learning*, pages 4138–4148. PMLR, 2020.

[51] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020.

[52] Yongxin Yang, Irene Garcia Morillo, and Timothy M. Hospedales. Deep neural decision trees. *International Conference in Machine Learning - Workshop on Human Interpretability in Machine Learning (WHI)*, 2018.

[53] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc-Alexander Najork. Are neural rankers still outperformed by gradient boosted decision trees? In *International Conference on Learning Representations*, 2021.

[54] Yu. V. Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, 2021.

[55] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):94, 2018.

[56] Christian Haslinger, Norbert Schweifer, Stephan Stilgenbauer, Hartmut Dohner, Peter Lichter, Norbert Kraut, Christian Stratowa, and Roger Abseher. Microarray gene expression profiling of b-cell chronic lymphocytic leukemia subgroups defined by genomic aberrations and vh mutation status. *Journal of Clinical Oncology*, 22(19):3937–3949, 2004.

[57] Arindam Bhattacharjee, William G Richards, Jane Staunton, Cheng Li, Stefano Monti, Priya Vasa, Christine Ladd, Javad Beheshti, Raphael Bueno, Michael Gillette, et al. Classification of human lung carcinomas by mrna expression profiling reveals distinct adenocarcinoma subclasses. *Proceedings of the National Academy of Sciences*, 98(24):13790–13795, 2001.

[58] Dinesh Singh, Phillip G Febbo, Kenneth Ross, Donald G Jackson, Judith Manola, Christine Ladd, Pablo Tamayo, Andrew A Renshaw, Anthony V D'Amico, Jerome P Richie, et al. Gene expression correlates of clinical prostate cancer behavior. *Cancer cell*, 1(2):203–209, 2002.

[59] Arthur Liberzon, Chet Birger, Helga Thorvaldsdóttir, Mahmoud Ghandi, Jill P Mesirov, and Pablo Tamayo. The molecular signatures database hallmark gene set collection. *Cell systems*, 1 (6):417–425, 2015.

[60] Yoshua Bengio Claude Nadeau. Inference for the generalization error. *Machine Learning, 52, 239–281*, 2003.

[61] Remco R. Bouckaert and Eibe Frank. Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2004.

[62] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

[63] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016.

[64] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[65] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[66] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[67] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

[68] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[69] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[70] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:5470–5477, 2020.

[71] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[72] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Internation Conference on Learning Representations*, 2015.

[73] Junchen Yang, Ofir Lindenbaum, and Yuval Kluger. Locally sparse neural networks for tabular biomedical data. *arXiv:2106.06468v2*, 2021.

[74] Hyunsoo Kim and Haesun Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12): 1495–1502, 2007.

[75] Leo Taslaman and Björn Nilsson. A framework for regularized non-negative matrix factorization, with application to the analysis of gene expression data. *PloS one*, 7(11):e46331, 2012.

[76] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Peter J Campbell, and Michael R Stratton. Deciphering signatures of mutational processes operative in human cancer. *Cell reports*, 3(1):246–259, 2013.

[77] Boris Weisfeiler. On construction and identification of graphs. 1976.

Appendix for submission "GCondNet: A Novel Method for Improving Neural Networks on Small High-Dimensional Tabular Data"

A Extended Related Work

"Diet" networks: Our focus is learning problems on high-dimensional tabular datasets with a limited number of samples. As such, our method takes inspiration from "diet" methods such as DietNetworks [28], FsNet [29] and WPFS [30], which rely on auxiliary networks to predict (and in turn reduce) the number of learnable parameters in the first layer of an underlying feed-forward network. However, GCondNet differs from "diet" methods in two important ways: (i) "diet" methods require a well-defined strategy for computing feature embeddings, and their performance is highly sensitive to this choice. In contrast, GCondNet defines graphs between samples and uses a GNN to learn the feature embeddings; (ii) GCondNet provides a different inductive bias which leverages the implicit relationships between samples (via the learned graph embeddings $\boldsymbol{w}^{(i)}$).

Out of all "diet" methods, GCondNet is most closely related to PLATO [22], as both methods employ GNNs as auxiliary networks to parameterise the predictor network. However, the similarities end there, and we highlight two key differences: (i) PLATO relies on domain knowledge, making it inapplicable when such information is unavailable. In contrast, GCondNet is more general, as it does not require domain knowledge but can still utilise it when available. (ii) PLATO constructs a single graph between features, whereas GCondNet creates multiple graphs between samples. This distinction is crucial, as PLATO leverages the relationships among features, while our method focuses on leveraging the relationships between samples (in addition to the relationships between features learnt by the MLP predictor itself). GCondNet differs from PLATO [22] in three important ways:

1. **Method's applicability:** PLATO relies on domain knowledge, making it inapplicable when such information is unavailable. In contrast, GCondNet is more general, as it can be applied to any tabular dataset without requiring domain knowledge.
2. **Graph type and what is learnt:** PLATO relies on *a single graph between features* – which is application-specific and must be provided a priori – and learns node embeddings to parameterise the first layer $\boldsymbol{W}^{[1]}_{\text{MLP}}$ of the predictor MLP. In contrast, GCondNet is task-agnostic: it constructs *multiple graphs between samples* from any tabular datasets, and it learns graph embeddings to parameterise $\boldsymbol{W}^{[1]}_{\text{MLP}}$. Our approach takes a distinctive perspective from PLATO, which allows for exploiting richer inductive biases captured in the complete structure of the data: (i) the relationships between samples through the GNN and (ii) the relationships between features through the predictor MLP.
3. **The "flexibility" of the inductive bias:** PLATO assumes that similar features have similar weights throughout the *entire* training. This is because in PLATO the first layer of the MLP is always computed using the GNN. In contrast, GCondNet is less conservative than PLATO; in the case of GCondNet, similar features have similar weights, but *only at the beginning* of the training. As the training progresses and α decreases to 0, the model adjusts the weights independently. This distinction is crucial for providing flexibility in the learning process. Our experiments in Sec. 3.3 show that keeping α fixed (similar to the inductive bias of PLATO) leads to unstable training.

Graph-based approaches for tabular data, including semi-supervised approaches, construct a graph between samples to capture the underlying relationships. The graphs are created using either a user-defined metric or by learning a latent graph between samples. Recent methods apply GNNs to these graphs, and our work distinguishes itself from such tabular data approaches in three ways.

1. We use GNNs indirectly and *only during training* to improve an underlying MLP predictor. Once trained, we store the MLP predictor's final weights, eliminating the need for GNNs during inference. Test input samples are subsequently processed exclusively through the predictor MLP. In contrast, GNN approaches to tabular data [41, 42, 43, 44, 45] directly employ GNNs *for inference* on new inputs, including making predictions [41, 43, 44, 45] and performing feature imputation [41, 42].
2. The graph structure is different. GCondNet generates many graphs between samples (one for each feature) and then extracts graph embeddings $\boldsymbol{w}^{(j)}$ to parameterise a predictor network.

This approach is novel and clearly distinguishes it from other work such as [41, 42, 43], which generate graphs connecting features and samples. Both [41, 42] construct a bipartite graph between samples and features, while [43] creates a hyper-graph where each sample is a node linked to corresponding feature nodes (specifically for discrete data).

3. Graph-based approaches for tabular data often introduce additional assumptions that may be suboptimal or inapplicable. For example, [46, 47, 44] create a graph between samples and rely on the *smoothness assumption*, which posits that neighbouring instances share the same labels. As demonstrated in Sec. 3.1, such assumptions can be suboptimal for high-dimensional data. Concerning *dataset assumptions*, [45] addresses few-shot learning, which requires a substantial meta-training set comprising similar tasks. In contrast, our approach focuses on learning from small datasets without assuming the presence of an external meta-training set. The work of [44] infers a latent graph, focusing on either images or tabular data with a maximum of 30 features. In comparison, our research explores tabular datasets containing up to 20000 features.

Feature selection. When faced with high-dimensional data, machine learning models are presented with increased degrees of freedom, making prediction tasks more challenging, especially on small sample-size tasks. To address this issue, various feature selection methods have been proposed to reduce the dimensionality of the data [48, 17, 16, 29, 33, 30, 34]. All these methods aim to model the relationships between features (i.e., determining which features are similar or irrelevant to the task), but they do not consider the relationships between samples. In contrast, GCondNet uses a GNN to extract the relationships between samples, while the MLP predictor learns the relationships between features.

Neural networks for tabular data. More broadly, our work is related to neural network methods for tabular data. Recent methods include various inductive biases, such as taking inspiration from tree-based methods [49, 50, 51, 52], including attention-based modules [31, 32], or modelling multiplicative feature interactions [53]. For a recent review on neural networks for tabular data, refer to [19]. However, these methods are generally designed for large sample size datasets, and their performance can vary on different datasets [54], making them unsuitable for small high-dimensional tasks. In contrast, our method is specifically designed for small high-dimensional tasks.

B Sparse Relative Distance (SRD) Graph Construction

Table B.1: Key statistics of the graphs created using the Sparse Relative Distance (SRD) with $dist = 5\%$.

Dataset	Node degree	Edges of full graph (%)
lung	5.16 ± 9.46	7.37
toxicity	3.1 ± 5.42	5.12
metabric-p50	3.97 ± 6.73	5.56
metabric-dr	3.92 ± 6.69	5.48
prostate	11.28 ± 17.96	31.77
cll	3.88 ± 6.81	9.96
smk	3.94 ± 6.61	5.93
tcga-survival	7.7 ± 12.03	10.77
tcga-tumor	7.45 ± 11.84	10.42

We propose a novel similarity-based method, Sparse Relative Distance (SRD), for creating edges between node (representing samples) in a graph. It assumes that similar samples should be connected and use this principle to create edges $\mathcal{E}_j$ in the jth graph. The method also includes an accept-reject step to sparsify the graph. Specifically, SRD work as follows:

1. For each node k create a set of candidate edges $\mathbb{C}_k$ by identifying all samples l with a feature value within a certain distance, $dist$, of the corresponding feature value of sample k. Specifically, we include all samples l such that $|\boldsymbol{X}_{k,j} - \boldsymbol{X}_{l,j}| \leq dist$, where $dist$ is defined as 5% of the absolute difference between the 5th and 95th percentiles of all values of feature j (to eliminate the effect of outlier feature values).
2. Perform a Bernoulli trial with probability $size(\mathbb{C}_k)/N_{\text{train}}$ for each node k. If the trial outcome is positive, create undirected edges between node k and all nodes within the candidate set $\mathbb{C}_k$. If the outcome is negative, no new edges are created. This sampling procedure results in sparser graphs, which helps alleviate the issue of oversmoothing [40] commonly encountered in GNNs.

Oversmoothing occurs when the model produces similar embeddings for all nodes in the graph, effectively 'smoothing out' any differences in the graph structure. It is worth noting that a node can also acquire new edges as part of the candidate set of other nodes.

This process results in a network topology where nodes with larger candidate sets have a higher probability of having more connections. Intuitively, this means that 'representative' samples become the centres of node clusters in the network.

3. To further prevent oversmoothing, if a node has more than 25 connections, we randomly prune some of its edges until it has exactly 25 connections. This is because samples with highly frequent values can have an excessive number of connections, which can result in oversmoothing.

C GCondNet Pseudocode

Algorithm 2 Training GCondNet

Input: training data $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, training labels $\boldsymbol{y} \in \mathbb{R}^{N}$, classification network $f_{\theta_{\text{MLP}}}$, graph neural network $g_{\theta_{\text{GNN}}}$, node aggregation function f_{agg}, graph creation method $h(\cdot)$, steps for linear decay n_α

for each feature $d = 1, 2, ..., D$ **do**
  $\mathcal{G}_d = h(\boldsymbol{X}_{:,d})$ {Compute the graphs between samples.}
end for
$\boldsymbol{W}_{\text{scratch}} = 0$ {Initialise auxiliary weight matrix.}
for each mini-batch $B = \{(\boldsymbol{x}^{(i)}, y_i)\}_{i=1}^{b}$ **do**
  for each feature $d = 1, 2, ..., D$ **do**
    node_embeddings = $g_{\theta_{\text{GNN}}}(\mathcal{G}_d)$ {The node embeddings are K dimensional.}
    $\boldsymbol{w}^{(d)}$ = f_{agg}(node_embeddings) {Aggregate all node embeddings to obtain the graph embedding $\boldsymbol{w}^{(d)} \in \mathbb{R}^K$.}
  end for
  $\boldsymbol{W}_{\text{GNN}} = [\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, ..., \boldsymbol{w}^{(D)}]$ {Horizontally concatenate the graph embeddings.}
  $\alpha = \max(0, 1 - (i/n_\alpha))$ {Compute mixing coefficient.}
  $\boldsymbol{W}_{\text{MLP}}^{[1]} \leftarrow \alpha \boldsymbol{W}_{\text{GNN}} + (1 - \alpha)\boldsymbol{W}_{\text{scratch}}$ {Compute weights first layer.}
  Make $\boldsymbol{W}_{\text{MLP}}^{[1]}$ the weight matrix of the first layer of $f_{\theta_{\text{MLP}}}$
  for each sample $i = 1, 2, ..., b$ **do**
    $\hat{y}_i = f_{\theta_{\text{MLP}}}(\boldsymbol{x}^{(i)})$
  end for
  $\hat{\boldsymbol{y}} \leftarrow [\hat{y}_1, \hat{y}_2, ..., \hat{y}_b]$ {Concatenate all predictions.}
  Compute training loss $L = CrossEntropyLoss(\boldsymbol{y}, \hat{\boldsymbol{y}})$
  Compute the gradient of the loss L w.r.t.
    $\theta_{\text{MLP}}, \theta_{\text{GNN}}, \boldsymbol{W}_{\text{scratch}}$ using backpropagation
  Update the parameters:
    $\theta_{\text{MLP}} \leftarrow \theta_{\text{MLP}} - \nabla_{\theta_{\text{MLP}}} L$
    $\theta_{\text{GNN}} \leftarrow \theta_{\text{GNN}} - \nabla_{\theta_{\text{GNN}}} L$
    $\boldsymbol{W}_{\text{scratch}} \leftarrow \boldsymbol{W}_{\text{scratch}} - \nabla_{\boldsymbol{W}_{\text{scratch}}} L$
end for

Return: Trained models $f_{\theta_{\text{MLP}}}, g_{\theta_{\text{GNN}}}$ and $\boldsymbol{W}_{\text{MLP}}^{[1]}$

D Datasets

All nine datasets are publicly available and summarised in Table D.2. Five datasets are open-source [55] and available `https://jundongl.github.io/scikit-feature/datasets.html`online: **CLL-SUB-111** (called 'cll') [56], **lung** [57], **Prostate_GE** (called 'prostate') [58], **SMK-CAN-187** (called 'smk') [12] and **TOX-171** (called 'toxicity') [13].

We created four additional datasets following the methodology presented in [30]:

- Two datasets from the **METABRIC** [14] dataset. We combined the molecular data with the clinical label 'DR' to create the **'metabric-dr'** dataset, and we combined the molecular data with the clinical label 'Pam50Subtype' to create the **'metabric-p50'** dataset. Because the label 'Pam50Subtype' was very imbalanced, we transformed the task into a binary task of basal vs non-basal by combining the classes 'LumA', 'LumB', 'Her2', 'Normal' into one class and using

Table D.2: Details of the nine real-world biomedical datasets used for experiments. The number of features is $15 - 110$ times larger than the number of samples.

Dataset	**# Samples**	**# Features**	**# Classes**	**# Samples per class**
lung	197	3312	4	17, 20, 21, 139
toxicity	171	5748	4	39, 42, 45, 45
metabric-p50	200	4160	2	33, 167
metabric-dr	200	4160	2	61, 139
prostate	102	5966	2	50, 52
cll	111	11340	3	11, 49, 51
smk	187	19993	2	90, 97
tcga-survival	200	4381	2	78, 122
tcga-tumor	200	4381	3	25, 52, 124

the remaining class 'Basal' as the second class. For both 'metabric-dr' and 'metabric-p50' we selected the Hallmark gene set [59] associated with breast cancer, and the new datasets contain 4160 expressions (features) for each patient. We created the final datasets by randomly sampling 200 patients stratified because we are interested in studying datasets with a small sample-size.

- Two datasets from the **TCGA** [15] dataset. We combined the molecular data and the label 'X2yr.RF.Surv' to create the **'tcga-survival'** dataset, and we combined the molecular data and the label 'tumor_grade' to create the **'tcga-tumor'** dataset. For both 'tcga-survival' and 'tcga-tumor' we selected the Hallmark gene set [59] associated with breast cancer, leaving 4381 expressions (features) for each patient. We created the final datasets by randomly sampling 200 patients stratified because we are interested in studying datasets with a small sample-size.

Dataset processing. Before training the models, we apply Z-score normalisation to each dataset. Specifically, on the training split, we learn a simple transformation to make each column of $\boldsymbol{X}_{train} \in \mathbb{R}^{N_{train} \times D}$ have zero mean and unit variance. We apply this transformation to the validation and test splits during cross-validation.

E Statistical Analysis

The statistical analysis includes a paired t-test and a Wilcoxon signed-rank test. Namely, for pairwise comparisons of the runs per dataset, we perform a corrected 1-sided t-test (assuming the alternative hypothesis is right-tailed w.r.t GCondNet), adjusting the variance for the multiple repeats and cross-validation overlaps as documented in [60, 61]. To provide a more general conclusion across all 9 datasets, we also perform a Wilcoxon signed-rank test [62, 63], comparing the mean performances of the respective methods.

Table E.3: **Statistical analysis of Table 1 from the main paper**. We compare GCondNet to each of the 15 baseline methods. The results show that in general (based on the Wilcoxon test), GCondNet performs statistically significantly better (at $\alpha = 0.05$) than half of the baselines. These include several recent methods (DietDNN, FSNet, TabNet, CAE, LassoNet, TabTransformer, GCN, GATv2), where GCondNet also significantly outperforms them in many per-dataset comparisons. In the case of the MLP baseline, while GCondNet's mean rank is significantly better, the per-dataset comparisons show no statistically significant difference at $\alpha = 0.05$. On the other hand, in terms of the remaining baselines, whilst the Wilcoxon test did not show a statistically significant difference, we observed differences in several per-dataset cases (e.g. 'toxicity' and 'cll'), where GCondNet significantly outperforms some of them, which further highlights the robustness and utility of GCondNet.

GCondNet vs.	cll	lung	metabric-dr	metabric-p50	prostate	smk	tcga-survival	tcga-tumor	toxicity	Wilcoxon
MLP	1.230E-01	1.401E-01	4.627E-01	1.333E-01	6.201E-02	5.118E-02	4.881E-01	1.277E-01	1.354E-01	1.172E-02
DietDNN	1.890E-03	3.804E-03	2.720E-01	2.317E-01	2.139E-02	1.016E-01	2.055E-01	8.036E-02	6.973E-07	3.906E-03
FSNet	3.935E-05	1.908E-02	2.817E-01	7.406E-05	5.807E-02	4.172E-03	2.606E-01	5.585E-02	4.322E-12	3.906E-03
SPINN	2.902E-02	1.201E-01	1.635E-01	5.986E-02	3.689E-01	4.590E-01	3.541E-01	1.510E-02	1.917E-01	2.500E-01
DNP	2.889E-02	1.344E-01	1.598E-01	6.259E-02	4.844E-01	3.701E-01	3.183E-01	2.669E-02	2.216E-01	3.008E-01
WPFS	1.862E-01	3.831E-01	4.611E-01	4.068E-01	1.726E-01	2.814E-01	9.633E-02	9.037E-02	2.159E-04	4.258E-01
TabNet	1.801E-07	1.122E-04	1.037E-02	1.592E-04	3.860E-06	2.599E-04	1.059E-01	1.356E-04	3.897E-14	3.906E-03
TabTransformer	1.014E-01	1.197E-01	3.077E-02	8.640E-02	6.456E-02	7.856E-02	4.377E-01	6.854E-04	8.204E-05	7.813E-03
CAE	2.113E-02	6.331E-06	3.195E-01	3.582E-01	1.989E-01	3.019E-02	1.289E-01	2.574E-05	3.787E-11	3.906E-03
LassoNet	3.664E-15	1.000E-18	1.286E-03	2.347E-12	4.545E-10	9.465E-06	1.286E-02	9.806E-06	1.000E-18	3.906E-03
ElasticNet	1.503E-02	4.600E-01	3.156E-01	3.583E-01	3.105E-01	2.074E-01	4.420E-01	3.671E-01	1.880E-01	9.102E-01
Random Forest	1.913E-01	2.003E-02	4.362E-03	2.838E-04	4.955E-01	4.890E-01	3.697E-02	4.048E-01	1.426E-09	2.076E-01
LightGBM	4.773E-03	1.064E-01	3.579E-01	1.844E-01	3.302E-01	2.365E-01	3.961E-01	2.198E-01	3.297E-09	2.031E-01
GCN	3.195E-06	1.741E-01	7.657E-02	5.750E-04	2.142E-02	2.947E-01	2.056E-01	6.814E-02	1.132E-05	3.906E-03
GATv2	3.195E-06	1.741E-01	7.657E-02	5.750E-04	2.142E-02	2.947E-01	2.056E-01	6.814E-02	1.132E-05	3.906E-03

Table E.4: **Statistical analysis of Table 2 from the main paper**. We compare both GCondNet w/ SRD and KDD variants to each other, to GCondNet with Random graphs, as well as to each of the MLP initialization baselines. Regarding different graph-constructing methods, the results further support our discussion that while there is some performance difference between each method when applied to different datasets, these differences are not significant at $\alpha = 0.05$. Furthermore, we find that both GCondNet variants (SRD and KNN) perform significantly better (based on the Wilcoxon test) than the MLP baselines (and their variants). However, per-dataset comparisons mainly show no statistical difference at $\alpha = 0.05$ but rather at $\alpha = 0.1$.

Model A	vs.	Model B	cll	lung	metabric-dr	metabric-p50	prostate	smk	tcga-survival	tcga-tumor	toxicity	**Wilcoxon**
GCondNet (SRD)		MLP	7.40E-02	1.40E-01	2.70E-01	1.67E-01	3.26E-01	5.12E-02	4.88E-01	6.85E-02	1.35E-01	2.73E-02
GCondNet (SRD)		MLP (WL)	2.25E-01	3.05E-01	4.30E-01	3.63E-01	4.84E-01	6.60E-02	2.78E-01	2.34E-01	3.05E-02	2.73E-02
GCondNet (SRD)		MLP (NMF)	1.24E-01	4.20E-01	3.02E-01	2.18E-01	4.37E-01	2.24E-01	1.06E-01	3.68E-01	1.92E-04	3.59E-01
GCondNet (SRD)		MLP (SVD)	2.18E-01	2.85E-01	1.73E-01	1.11E-01	8.65E-02	3.36E-01	3.77E-01	3.39E-01	1.91E-02	9.77E-02
GCondNet (KNN)		MLP	1.23E-01	2.59E-01	4.63E-01	1.33E-01	6.20E-02	3.14E-01	1.84E-01	1.28E-01	1.49E-01	7.81E-03
GCondNet (KNN)		MLP (WL)	3.31E-01	4.59E-01	4.18E-01	3.26E-01	1.83E-01	2.87E-01	7.79E-02	2.73E-01	4.22E-02	3.91E-03
GCondNet (KNN)		MLP (NMF)	7.61E-02	3.72E-01	4.96E-01	1.88E-01	2.79E-01	4.94E-01	2.31E-01	4.75E-01	3.24E-04	2.63E-01
GCondNet (KNN)		MLP (SVD)	3.13E-01	8.37E-02	1.08E-01	6.09E-02	3.64E-01	3.87E-01	1.33E-01	4.25E-01	3.39E-02	1.64E-01
GCondNet (SRD)		GCondNet (KNN)	2.89E-01	2.30E-01	3.42E-01	4.30E-01	1.80E-01	1.82E-01	1.76E-01	3.91E-01	4.86E-01	8.33E-01
GCondNet (SRD)		GCondNet (Random)	4.47E-01	2.67E-01	4.26E-01	2.78E-01	3.78E-01	1.34E-01	2.02E-01	3.72E-01	3.93E-01	3.01E-01
GCondNet (KNN)		GCondNet (Random)	1.99E-01	3.89E-01	2.75E-01	2.09E-01	1.12E-01	3.61E-01	4.53E-01	4.87E-01	4.13E-01	5.28E-01

F Ablation Number of Steps for Decaying the Mixing Coefficient

Table F.5: We evaluate the impact of decaying the mixing coefficient α for varying number of steps n_α. We present the mean$\pm$std balanced *validation accuracy* averaged over 25 runs. We find that GCondNet is robust to the decay length n_α, and we choose $n_\alpha = 200$ steps for all experiments of this paper unless otherwise specified.

Dataset	**Steps n_α of linear decay for α**		
	100	200	400
lung	96.77 ± 5.07	97.27 ± 4.97	97.36 ± 4.76
toxicity	98.75 ± 3.12	98.25 ± 3.83	98.5 ± 3.73
metabric-p50	98.56 ± 3.70	98.56 ± 3.70	97.74 ± 5.02
metabric-dr	71.20 ± 8.80	68.36 ± 10.63	71.92 ± 8.39
prostate	95.56 ± 7.35	93.97 ± 10.22	95.11 ± 7.99
cll	88.88 ± 8.47	88.09 ± 9.34	88.33 ± 8.28
smk	76.39 ± 11.59	76.89 ± 13.10	75.78 ± 11.64
tcga-survival	69.20 ± 11.38	70.6 ± 9.35	68.53 ± 10.01
tcga-tumor	62.06 ± 13.64	62.46 ± 18.05	66.13 ± 15.17

Table F.6: Statistical analysis of the number of iterations from Table F.5. We report the p-values of a Wilcoxon signed-rank test. The results support our observation, namely that GCondNet is robust to the hyper-parameter n_α.

$n_\alpha = 100$ vs. $n_\alpha = 200$	$n_\alpha = 100$ vs. $n_\alpha = 400$	$n_\alpha = 200$ vs. $n_\alpha = 400$
4.00E-01	9.44E-01	4.96E-01

G Ablation Impact Graph Type in GCondNet

Table G.7: We evaluate the impact of the graph type of GCondNet. We present mean$\pm$std validation and test balanced accuracy of GCondNet with three different graph types, averaged over 25 runs. In general, GCondNet has fairly stable performance across different types of graphs.

Dataset	GCondNet with varying graphs					
	Random graphs		SRD graphs		KNN graphs	
	Validation	Test	Validation	Test	Validation	Test
lung	96.88 ± 4.96	94.86 ± 4.57	97.27 ± 4.97	95.34 ± 4.48	96.68 ± 5.08	94.68 ± 4.24
toxicity	98.55 ± 3.73	95.06 ± 6.14	98.25 ± 3.83	95.25 ± 4.53	98.25 ± 3.83	95.22 ± 3.92
metabric-p50	98.94 ± 2.77	95.86 ± 4.25	98.56 ± 3.70	96.26 ± 3.79	98.56 ± 3.70	96.37 ± 3.99
metabric-dr	68.90 ± 10.53	57.89 ± 8.76	68.36 ± 10.63	58.24 ± 6.35	69.56 ± 12.55	59.34 ± 8.93
prostate	95.42 ± 7.21	89.56 ± 6.37	93.97 ± 10.22	89.95 ± 6.14	95.00 ± 8.81	90.37 ± 5.59
cll	89.71 ± 8.96	81.36 ± 5.78	88.09 ± 9.34	81.54 ± 7.14	89.00 ± 9.22	80.69 ± 5.47
smk	76.83 ± 12.27	66.13 ± 8.11	76.89 ± 13.10	68.08 ± 7.31	76.03 ± 13.00	65.92 ± 8.68
tcga-survival	67.17 ± 12.08	58.31 ± 7.81	70.6 ± 9.35	56.36 ± 9.41	66.73 ± 11.07	58.61 ± 7.01
tcga-tumor	62.38 ± 15.33	51.57 ± 9.10	62.46 ± 18.05	52.42 ± 7.57	64.4 ± 13.67	51.69 ± 8.81

H Benchmarks, Training Details and Hyper-parameter Tuning

Software implementation. We implemented GCondNet using PyTorch 1.12 [64], an open-source deep learning library with a BSD licence. We implemented the GNN within GCondNet, and the GCN and GATv2 benchmarks using PyTorch-Geometric [65], an open-source library for implementing Graph Neural Networks with an MIT licence. We train using a library `https://github.com/Lightning-AI/lightning`Pytorch-ligthning built on top of PyTorch and released under an Apache Licence 2.0. All numerical plots and graphics have been generated using Matplotlib 3.6, a Python-based plotting library with a BSD licence. The model architecture Figure 1 was generated using `https://github.com/jgraph/drawio`draw.io, a free drawing software under Apache License 2.0.

As for the other benchmarks, we implement MLP, CAE and DietNetworks using PyTorch 1.12 [64], Random Forest and ElasticNet using scikit-learn [66] (BSD license), LightGBM using the lightgbm library [37] (MIT licence) and TabNet [31] using the `https://github.com/dreamquark-ai/tabnet`implementation (MIT licence) from Dreamquark AI. We use the MIT-licensed `https://github.com/andreimargeloiu/WPFS`implementation of WPFS made public by [30]. We re-implement FsNet [29] in PyTorch 1.12 [64] because the official code implementation contains differences from the paper, and they used a different evaluation setup from ours (they evaluate using unbalanced accuracy, while we run multiple data splits and evaluate using balanced accuracy). We use the `https://github.com/lasso-net/lassonet`official implementation of LassoNet (MIT licence), and the `https://github.com/jjfeng/spinn`official implementation of SPINN (no licence).

We attach our code to this submission, and we will release it under the MIT licence upon publication.

Computing Resources. All our experiments are run on a single machine from an internal cluster with a GPU Nvidia Quadro RTX 8000 with 48GB memory and an Intel(R) Xeon(R) Gold 5218 CPU with 16 cores (at 2.30GHz). The operating system was Ubuntu 20.4.4 LTS. We estimate that to carry out the full range of experiments, comprising both prototyping and initial experimental phases, we needed to train around 11000 distinct models, which we estimate required 1000 to 1200 GPU hours.

GCondNet architecture and settings. The predictor MLP is a 3-layer feed-forward neural network with $100, 100, 10$ neurons. After each linear layer we add LeakyReLU non-linearity with slope 0.01, batch normalisation [67] and dropout [68] with $p = 0.2$. The last layer has softmax activation. The layers following the first one are initialized using a standard Kaiming method [25], which considers the activation. The GNN within GCondNet is a Graph Convolutional Network (GCN) [27] with two layers of size 200 and 100. After the first GCN layer, we use a ReLU non-linearity and dropout with $p = 0.5$. The permutation invariant function f_{agg} for computing graph embeddings is global average pooling.[2]

We train for 10000 steps with a batch size of 8 and optimise using AdamW [71] with a fixed learning rate of $1e-4$. We decay the mixing coefficient α over $n_\alpha = 200$ training steps, although we found that GCondNet is robust to the number of steps n_α, as supported by the statistical tests in Appendix F. We use early stopping with patience 200 steps on the validation loss across all experiments.

For GCondNet, we choose the graph construction method (between KNN and SRD) using the validation balanced accuracy, although this approach sometimes yields suboptimal test performance (Appendix G presents the validation and test accuracy for different graph types).

Training details for all benchmark methods. Here, we present the training settings for all benchmark models, and we discuss hyper-parameter tuning in the next paragraph. We train using 5-fold cross-validation with 5 repeats (training 25 models each run). For each run, we select 10% of the training data for validation. We perform a fair comparison whenever possible: for instance, we train all models using a weighted loss (e.g., weighted cross-entropy loss for neural networks), evaluate using balanced accuracy, and use the same classification network architecture for GCondNet, MLP, WPFS, FsNet, CAE and DietNetworks.

- **WPFS, DietNetworks, CAE, FsNet** have three hidden layers of size $100, 100, 10$. The Weight Predictor Network and the Sparsity Network have four hidden layers $100, 100, 100, 100$. They are trained for $10{,}000$ steps using early stopping with patience 200 steps on the validation

[2] Using hierarchical pooling methods [69, 70] resulted in unstable training and significantly poor performance.

cross-entropy and gradient clipping at 2.5. For **CAE** and **FsNet** we use the suggested annealing schedule for the concrete nodes: exponential annealing from temperature 10 to 0.01. On all datasets, **DietNetworks** performed best with not decoder. For **WPFS** we use the NMF embeddings suggested in 30.

- For **GCN**, we used two graph convolutional layers with 200 and 100 neurons, followed by a linear layer with softmax activation for class label computation. ReLU and dropout with $p = 0.5$ follow each convolutional layer. We train using AdamW [71], tune the learning rate in $[1e-3, 3e-3, 1e-4]$, and select the best model on the validation accuracy.
- For **GATv2**, we used two graph attentional layers with dropout $p = 0.5$. The first attention layer used 4 attention heads of size 100, and the second used one head of size 400. ReLU and dropout with $p = 0.5$ follow each attention layer. A linear layer with softmax activation for class label computation follows the attention layers. We train using AdamW [71], tune the learning rate in $[1e-3, 3e-3, 1e-4]$, and select the best model on the validation accuracy.
- **LassoNet** has three hidden layers of size $100, 100, 10$. We use dropout 0.2, and train using AdamW (with betas $0.9, 0.98$) and a batch size of 8. We train using a weighted loss. We perform early stopping on the validation set.
- For **Random Forest**, we used 500 estimators, feature bagging with the square root of the number of features, and used balanced weights associated with the classes.
- For **LightGBM** we used 200 estimators, feature bagging with 30% of the features, a minimum of two instances in a leaf, and trained for $10{,}000$ steps to minimise the balanced cross-entropy. We perform early stopping on the validation set.
- For **TabNet**, we use width 8 for the decision prediction layer and the attention embedding for each mask (larger values lead to severe overfitting) and 1.5 for the feature re-usage coefficient in the masks. We use three steps in the architecture, with two independent and two shared Gated Linear Units layers at each step. We train using Adam [72] with momentum 0.3 and gradient clipping at 2.
- For **TabTransformer**, we use only the head for continuous features, as our datasets do not contain categorical features. For a fair comparison, we use the same architecture as the MLP following the initial layer normalization, and train the model with the optimal settings of the MLP.
- For **ElasticNet**, we train for 10000 iterations using the 'saga' solver and weighted loss.
- For **SPINN**, we used $\alpha = 0.9$ for the group lasso in the sparse group lasso, the sparse group lasso hyper-parameter $\lambda = 0.0032$, ridge-param $\lambda_0 = 0.0001$, and train for at most $1{,}000$ steps.
- For **DNP** we did not find any suitable implementation, and used SPINN with different settings as a proxy for DNP (because DNP is a greedy approximation to optimizing the group lasso, and SPINN optimises directly a group lasso). Specifically, our proxy for DNP results is SPINN with $\alpha = 1$ for the group lasso in the sparse group lasso, the sparse group lasso hyper-parameter $\lambda = 0.0032$, ridge-param $\lambda_0 = 0.0001$, and train for at most 1000 iterations.

Hyper-parameter tuning. For each model, we use random search and previous experience to find a good range of hyper-parameter values that we can investigate in detail. We then performed a grid search and ran 25 runs for each hyper-parameter configuration. We selected the best hyper-parameter based on the average validation accuracy across the 25 runs.

For the MLP and DietNetworks we individually grid-searched learning rate $\in \{0.003, 0.001, 0.0003, 0.0001\}$, batch size $\in \{8, 12, 16, 20, 24, 32\}$, dropout rate $\in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. We found that learning rate 0.003, batch size 8 and dropout rate 0.2 work well across datasets for both models, and we used them in the presented experiments. In addition, for DietNetworks we also tuned the reconstruction hyper-parameter $\lambda \in \{0, 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30\}$ and found that across dataset having $\lambda = 0$ performed best. For WPFS we used the best hyper-parameters for the MLP and tuned only the sparsity hyper-parameter $\lambda \in \{0, 3e-6, 3e-5, 3e-4, 3e-3, 1e-2\}$ and the size of the feature embedding $\in \{20, 50, 70\}$. For FsNet we grid-search the reconstruction parameter in $\lambda \in \{0, 0.2, 1, 5\}$ and learning rate in $\{0.001, 0.003\}$. For LightGBM we performed grid-search for the learning rate in $\{0.1, 0.01\}$ and maximum depth in $\{1, 2\}$. For Random Forest, we performed a grid search for the maximum depth in $\{3, 5, 7\}$ and the minimum number of samples in a leaf in $\{2, 3\}$. For TabNet we searched the learning rate in $\{0.01, 0.02, 0.03\}$ and the λ sparsity hyper-parameter in $\{0.1, 0.01, 0.001, 0.0001\}$, as motivated by [73]. For ElasticNet, we searched the L1 ratio $\in \{0, 0.25, 0.5, 0.75, 1\}$ and the

sparsity hyper-parameter $C \in \{10, 100, 1000\}$. For GCN and GATv2 we searched the learning rate in $\{1e-3, 3e-3, 1e-4\}$. We selected the best hyper-parameters (Table H.8) on the weighted cross-entropy (except Random Forest, for which we used weighted balanced accuracy).

Table H.8: Best performing hyper-parameters for each benchmark model across datasets.

	Random Forest		LightGBM		ElasticNet		TabNet		GCN	GATv2	FsNet		CAE	WPFS	
	max depth	min samples leaf	learning rate	max depth	L1 ratio	C	learning rate	λ sparsity	learning rate	learning rate	learning rate	λ reconstruction	annealing iterations	λ sparsity	embedding size
lung	3	2	0.1	1	0.5	10	0.02	0.1	0.003	0.001	0.001	0	1000	$3e-5$	20
toxicity	5	3	0.1	2	1	10	0.03	0.1	0.003	0.0001	0.001	0.2	1000	$3e-5$	50
metabric-p50	7	2	0.01	2	1	10	0.02	0.001	0.003	0.003	0.003	0	1000	$3e-6$	50
metabric-dr	7	2	0.1	1	0.75	10	0.03	0.1	0.003	0.003	0.003	0	300	0	50
prostate	5	2	0.1	2	1	10	0.02	0.01	0.0001	0.0001	0.003	0	1000	$3e-3$	50
cll	3	3	0.1	2	0.5	10	0.03	0.001	0.003	0.003	0.003	0	1000	$3e-4$	70
smk	5	2	0.1	2	0.5	10	0.03	0.001	0.001	0.0001	0.003	0	1000	$3e-5$	50
tcga-survival	3	3	0.1	1	0.5	10	0.02	0.01	0.003	0.003	0.003	0	300	$3e-5$	50
tcga-tumor	3	3	0.1	1	0.75	10	0.02	0.01	0.003	0.001	0.003	0	300	$3e-5$	50

LassoNet unstable training. We used the official implementation of LassoNet (https://github.com/lasso-net/lassonet), and we successfully replicated some of the results in the LassoNet paper [34]. However, LassoNet was unstable on all the nine datasets we trained on. We included a Jupyter notebook in the attached codebase that demostrates that LassoNet cannot even fit the training data on our datasets. In our experiments, we grid-searched the L_1 penalty coefficient $\lambda \in \{0.001, 0.01, 0.1, 1, 10, `auto'\}$ and the hierarchy coefficient $M \in \{0.1, 1, 3, 10\}$. These values are suggested in the paper and used in the examples from the official codebase. For all hyper-parameter combinations, LassoNet's performance was equivalent to a random classifier (e.g., 25% balanced accuracy for a 4-class problem).

I Computational Complexity

I.1 Training Time

Table I.9: Average training time (in minutes) of a model with optimal hyper-parameters. In general, GCondNet trains slower than the benchmarks: GCondNet takes 8.5 minutes to train across datasets, other competitive "diet" networks, such as WPFS, take 7.7 minutes, and an MLP takes 5.4 minutes.

Dataset	GCondNet	MLP	WPFS	DietNetworks	FsNet
lung	11	6.1	9.9	10.6	6.1
toxicity	13.7	7.3	10	8.2	5.9
metabric-p50	9.8	6.6	8.9	8.7	4.3
metabric-dr	3.6	3	4.6	4.9	4.8
prostate	9.6	7.2	9.9	7.9	3.5
cll	12	6.1	7.7	8.1	5.4
smk	9	6.2	8.9	5.9	4.3
tcga-survival	3.8	3	4.2	4.7	5.3
tcga-tumor	4	3.5	5	4.4	4.5
Average minutes	8.5	5.4	7.7	7	4.9

I.2 Number of Steps for Convergence

J Influence of the MLP Initialisation

Random Graphs from Sec. 3.2. The proposed SRD method creates graphs that contain, on average, 8% of the edges of a fully connected graph (as shown in Appendix B). To create random graphs with similar statistics, we sample a proportion p from all possible edges in the graph, where $p \sim \mathcal{N}(\mu = 0.08, \sigma = 0.03)$ is sampled for each of the D graphs. We used the same initial node embeddings from Sec. 2.1. We sample each graph five times and train each of the 25 models on all graphs – resulting in 125 trained models on random graphs.

Specialised initialisations from Sec. G. These schemes generate feature embeddings $\boldsymbol{e}^{(i)}$, which are then utilised to initialise the MLP's first layer $\boldsymbol{W}^{[1]}_{\text{MLP}} = [\boldsymbol{e}^{(1)}, \boldsymbol{e}^{(2)}, ..., \boldsymbol{e}^{(D)}]$. We first compute $\boldsymbol{W}^{[1]}_{\text{MLP}}$ using any of the three initialisation methods that we propose below. To mitigate the risk of

Table I.10: Number of steps for convergence.

	MLP	GCondNet with different graph types		
		KNN graphs	SRD graphs	Random graphs
lung	2836	6927	6777	6935
toxicity	2807	6522	6809	6772
metabric-p50	2952	5504	5365	5242
metabric-dr	1331	1657	1577	1598
prostate	2347	4307	4540	4535
cll	1697	4020	4526	4298
smk	1379	2175	2102	2218
tcga-survival	1320	1541	1606	1588
tcga-tumor	1316	1805	1790	1805
Average	1998	3829	3899	3887

exploding gradients, we adopt the method proposed by He et al. [25] to rescale the weights. After computing $\boldsymbol{W}_{\text{MLP}}^{[1]}$, we then perform zero-centring on each row of $\boldsymbol{W}_{\text{MLP}}^{[1]}$ and subsequently rescale it to match the standard deviation of the Kaiming initialisation [25]. The resulting matrix is used to initialise the first layer of the predictor MLP.

1. **Principal Component Analysis (PCA) initialisation.** We use PCA to compute feature embeddings $\boldsymbol{e}_{\text{PCA}}^{(j)}$ for all features j. These embeddings are then concatenated horizontally to form the weight matrix of the first layer of the MLP predictor $\boldsymbol{W}_{\text{MLP}}^{[1]} = [\boldsymbol{e}_{\text{PCA}}^{(1)}, \boldsymbol{e}_{\text{PCA}}^{(2)}, ..., \boldsymbol{e}_{\text{PCA}}^{(D)}]$.
2. **Non-negative matrix factorisation (NMF) initialisation.** NMF has been applied in bioinformatics to cluster gene expression [74, 75] and identify common cancer mutations [76]. It approximates $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$, with the intuition that the column space of $\boldsymbol{W}$ represents "eigengenes", and the column $\boldsymbol{H}_{:,j}$ represents coordinates of gene j in the space spanned by the eigengenes. The feature embedding is $\boldsymbol{e}_{\text{NMF}}^{(j)} := \boldsymbol{H}_{:,j}$. These embeddings are then concatenated horizontally to form the weight matrix of the first layer of the MLP predictor $\boldsymbol{W}_{\text{MLP}}^{[1]} = [\boldsymbol{e}_{\text{NMF}}^{(1)}, \boldsymbol{e}_{\text{NMF}}^{(2)}, ..., \boldsymbol{e}_{\text{NMF}}^{(D)}]$.
3. **Weisfeiler-Lehman (WL) initialisation.** The WL algorithm [77], often used in graph theory, is a method to check whether two given graphs are isomorphic, i.e., identical up to a renaming of the vertices. The algorithm creates a graph embedding of size N. For our use-case, the embeddings must have size K, the size of the first hidden layer of the predictor MLP; thus, we obtain the feature embeddings $\boldsymbol{e}_{\text{WL}}^{(j)}$ by computing the histogram with K bins of the WL-computed graph embedding, which is then normalised to be a probability density. We apply the WL algorithm on the SRD graphs described in Sec. 2.1, and finally obtain $\boldsymbol{W}_{\text{WL}}^{[1]} = [\boldsymbol{e}_{\text{WL}}^{(1)}, \boldsymbol{e}_{\text{WL}}^{(2)}, ..., \boldsymbol{e}_{\text{WL}}^{(D)}]$.