Tabular Deep Learning when $d \gg n$ by Using an Auxiliary Knowledge Graph

Camilo Ruiz*
Department of Computer Science
Stanford University
Stanford, CA 94305
caruiz@cs.stanford.edu

Hongyu Ren*
Department of Computer Science
Stanford University
Stanford, CA 94305
hyren@cs.stanford.edu

Kexin Huang
Department of Computer Science
Stanford University
Stanford, CA 94305
kexinh@cs.stanford.edu

Jure Leskovec
Department of Computer Science
Stanford University
Stanford, CA 94305
jure@cs.stanford.edu

Abstract

Machine learning models exhibit strong performance on datasets with abundant labeled samples. However, for tabular datasets with extremely high d-dimensional features but limited n samples (*i.e.* $d \gg n$), machine learning models struggle to achieve strong performance. Here, our key insight is that even in tabular datasets with limited labeled data, input features often represent real-world entities about which there is abundant prior information which can be structured as an auxiliary knowledge graph (KG). For example, in a tabular medical dataset where every input feature is the amount of a gene in a patient's tumor and the label is the patient's survival, there is an auxiliary knowledge graph connecting gene names with drug, disease, and human anatomy nodes. We therefore propose PLATO, a machine learning model for tabular data with $d \gg n$ and an auxiliary KG with input features as nodes. PLATO uses a modified multilayer perceptron (MLP) to predict the output labels from the tabular data and the auxiliary KG with two components. First, PLATO predicts the parameters in the first layer of the MLP from the auxiliary KG. PLATO thereby reduces the number of trainable parameters in the MLP and integrates auxiliary information about the input features. Second, PLATO predicts different parameters in the first layer of the MLP for every input sample, thereby increasing the MLP's representational capacity by allowing it to use different prior information for every input sample. Across 10 state-of-the-art baselines and 6 $d \gg n$ datasets, PLATO exceeds or matches the prior state-of-the-art, achieving performance improvements of up to 10.19%. Overall, PLATO uses an auxiliary KG about input features to enable tabular deep learning prediction when $d \gg n$.

1 Introduction

Machine learning models have reached state-of-the-art performance in domains with abundant labeled data like computer vision [1, 2] and language [3, 4, 5]. However, for tabular datasets in which the number d of features vastly exceeds the number n of samples, machine learning models struggle to achieve strong performance [6, 7]. Unfortunately, many high impact domains like chemistry [8], biology [9, 10, 11, 12], and physics [13] produce datasets with high-dimensional features but limited labeled samples due to the high time and labor costs of experiments. In chemistry, for example, mass

NeurIPS 2022 AI for Science Workshop.

spectrometry datasets can have tens of thousands of features but only hundreds of samples [8]. For these tabular datasets with $d \gg n$, the performance of machine learning systems is currently limited.

To date, deep learning approaches for tabular data have focused on data regimes with far more samples than features ($n \gg d$) [14, 15, 16]. In the low-data regime with far more features than samples ($d \gg n$), the dominant approaches are statistical methods [6] that reduce the dimensionality of the input space [17, 7, 18, 19], select features [20, 21, 22, 23], impose regularization penalties on parameter magnitudes [24], or use ensembles of weak tree-based models [25, 26, 27, 28, 29].

Here, we present a novel problem setting and framework for tabular deep learning when $d \gg n$ (Figure 1). Our key insight is that even in tabular settings with limited labeled data, input features often represent real-world entities about which there is abundant prior information which can be structured as an auxiliary knowledge graph (KG). We propose a novel problem setting in which every input feature of a tabular dataset corresponds to a node in an auxiliary KG (Figure 1a). For example, consider a tabular medical dataset in which every row is a cancer patient, every column is a gene, every value is the amount of that gene in the patient's tumor, and the task is to predict the patient's survival. For this tabular dataset, there exists an auxiliary KG which consists of each gene's function, the relationships between genes, how a gene affects a part of human anatomy, and how human anatomy itself is structured. Note that the KG does *not* capture the relationships between *input data samples* but instead captures the relationships between *input features*.

Within our novel problem setting, we propose PLATO, a deep learning method for tabular datasets with $d \gg n$ and an auxiliary KG with input features as nodes (Figure 1(b)-(e)). PLATO uses a modified multilayer perceptron (MLP) to predict the output labels from the input samples and the auxiliary KG with two components. First, the parameters in the first layer of the MLP are predicted from the auxiliary KG and the input sample rather than learned from just the tabular data. PLATO thereby integrates prior information about the input features from the auxiliary KG and drastically reduces the number of trainable parameters in the MLP. Second, the parameters in the first layer of the MLP are predicted differently for every sample by using the auxiliary KG and the sample values. PLATO thereby increases the representational capacity of the MLP and enables effective predictions.

We exhibit PLATO's performance on 6 datasets. We choose biomedicine as it is a rich domain for $d \gg n$ in which we can construct a single knowledge graph to serve as a unified backbone for many distinct tabular datasets with distinct input features. We compare PLATO to 10 state-of-the-art baselines spanning dimensionality reduction, feature selection, classic statistical models, deep tabular learning methods, and parameter-prediction methods. Following a rigorous evaluation protocol from the tabular deep learning literature [14, 15], PLATO achieves or matches the prior state-of-the-art on all 6 datasets, achieving performance improvements of up to 10.19%. Ablation studies further demonstrate the necessity of each methodological component of PLATO. Ultimately, PLATO uses an auxiliary KG about input features to enable tabular deep learning prediction when $d \gg n$.

2 Related Work

Tabular deep learning methods. In contrast to PLATO's setting, tabular deep learning methods have been developed for settings with far more samples than features (*i.e.* $n \gg d$). Indeed, recent tabular benchmarks ignore datasets with high numbers of features and low numbers of samples [14, 15, 16]. In the $n \gg d$ setting, various categories of deep tabular models have been benchmarked. We select the state-of-the-art models to compare against PLATO. First, decision tree models like NODE [30] make decision trees differentiable to enable gradient-based optimization [31, 32, 33]. Second, tabular transformer architectures use an attention mechanism to select and learn interactions among features. These include TabNet [34], TabTransformer [35], FT-Transformer [15], and others [36, 37, 38].

$d \gg n$ methods. For PLATO's setting of $d \gg n$, various tabular learning approaches exist [6]. First, dimensionality reduction methods like PCA [17] reduce the dimensionality of the input data while preserving as much of the data variance as possible [7, 18, 19]. Second, feature selection approaches select a parsimonious set of features, leading to a smaller feature space. Classical feature selection approaches include LASSO [20] and its variants [21, 22, 23]. For feature selection with deep learning, Stochastic Gates [39] are among the best performing of variants [40, 41]. Finally, tree-based models like XGBoost learn ensembles of weak decision trees to make an overall prediction [25, 26, 27, 29].

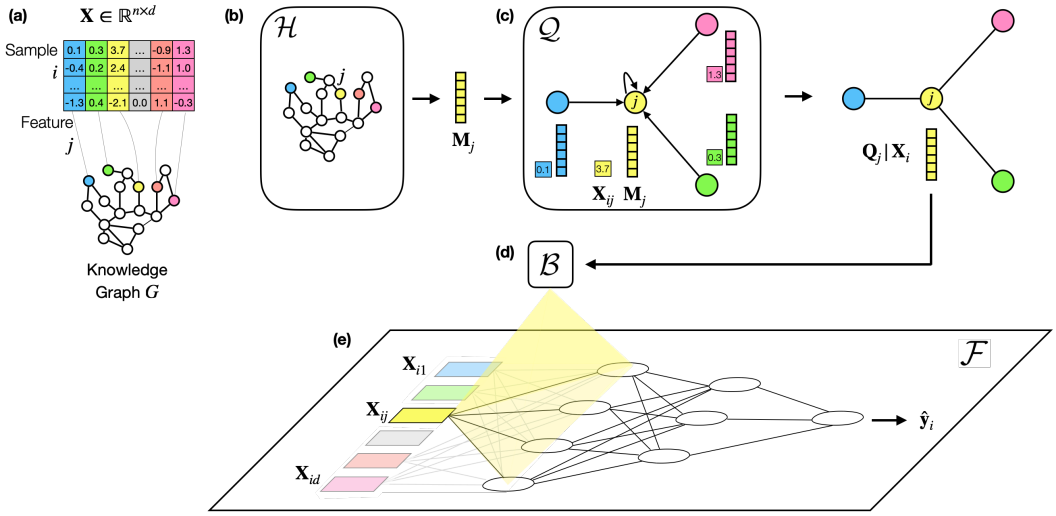


Figure 1: PLATO is a deep learning model for tabular data with $d \gg n$ and an auxiliary knowledge graph (KG) with input features as nodes. Machine learning models struggle to achieve strong performance on tabular data with far more d features than n samples (*i.e.* $d \gg n$). **(a)** The key insight of PLATO is that even in settings with limited labeled samples, input features often represent real-world entities about which there is abundant prior knowledge. We propose a new problem setting in which every feature in the input matrix corresponds to a node in an auxiliary KG G. **(b-e)** PLATO uses G to predict the parameters in the first layer of a modified MLP $\mathcal{F}$. **(b)** First, PLATO pretrains an embedding $\mathbf{M}_j \in \mathbb{R}^c$ for every feature node in G using $\mathcal{H}$, a self-supervised node embedding approach. **(c)** Second, PLATO updates each feature embedding to focus on the feature information that is most relevant to an input sample $\mathbf{X}_i$. PLATO uses a message passing network $\mathcal{Q}$ to produce $\mathbf{Q}_j \in \mathbb{R}^c$. $\mathcal{Q}$ uses an attention mechanism which considers the input sample $\mathbf{X}_i$. $\mathbf{Q}_j$ thus depends on the input sample (*i.e.* $\mathbf{Q}_j | \mathbf{X}_i$). **(d, e)** Finally, PLATO uses a small neural network $\mathcal{B}$ to predict the parameters in the first layer of a MLP $\mathcal{F}$ from $\mathbf{Q}_j$. The parameters in the first layer of $\mathcal{F}$ vary for every input sample $\mathbf{X}_i$.

Knowledge graph approaches. Existing knowledge graph approaches are designed for tasks directly on the graph such as link prediction [42, 43, 44, 45, 46]. By contrast, PLATO does not make any predictions on the knowledge graph. Instead, PLATO makes predictions on a separate, tabular dataset by using the knowledge graph as prior information about the features and domain. Graph classification approaches are similarly not applicable to PLATO's problem setting (Appendix C).

Parameter prediction. Using one network to predict the parameters of another is well studied [47, 48, 49]. For example, hypernetworks predict the weights in all layers of a sequential model (*i.e.* RNN, LSTM) by using information about the structure of the weights [50]. Diet Networks [51] predict parameters by hand-crafting prior information about the features or using random projections. By contrast, PLATO predicts parameters in a network by leveraging prior information about the input features in an auxiliary KG. PLATO systematically constructs a feature embedding which contains the prior information about the feature that is most relevant to a given sample.

3 PLATO

PLATO is a deep learning method for tabular datasets with $d \gg n$ and an auxiliary KG with input features as nodes (Section 3.1). Our key insight is that even in tabular datasets with limited labeled samples, input features often represent real world entities with abundant prior information that can be structured as an auxiliary knowledge graph (KG) (Figure 1(a)). PLATO uses a modified MLP to predict the output labels from the input samples and the KG. PLATO's modified MLP has two parts. First, the parameters in the MLP's first layer are predicted from the KG and the input sample rather than learned from just the tabular data (Figure 1(b-e)). PLATO thus integrates prior information about the input features from the KG and drastically reduces the number of trainable parameters in the MLP. Second, the parameters in the MLP's first layer are predicted differently for every sample. PLATO thus increases the MLP's representational capacity.

3.1 Problem Setting

Consider a tabular dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ with labels $\mathbf{y} \in \mathbb{R}^n$ and far more features than samples such that $d \gg n$. The goal is to train a machine learning model $\mathcal{F}$ to predict labels $\hat{\mathbf{y}}$ from the input $\mathbf{X}$. PLATO assumes the existence of an auxiliary knowledge graph $G = (V, E)$ with $|V|$ nodes and $|E|$ edges such that every input feature j corresponds to a node in G. Formally, $\forall j \in \{1, \ldots, d\}$, $\exists v \in V$ s.t. $j \mapsto v$, as shown in Figure 1(a). G also contains additional nodes which represent broader knowledge about the domain. The edges in G are (head node, relation type, tail node) triplets.

3.2 The PLATO MLP $\mathcal{F}$

Consider a standard MLP $\hat{\mathbf{y}} = \mathcal{T}(\mathbf{X}; \mathbf{\Theta})$ with L layers, h hidden units in the first layer, and trainable parameters $\mathbf{\Theta} = \{\mathbf{\Theta}^{[1]}, \mathbf{\Theta}^{[2]}, \ldots, \mathbf{\Theta}^{[L]}\}$. The PLATO MLP $\mathcal{F}$ differs from $\mathcal{T}$ in two key ways.

First, the parameters in the first layer of PLATO's MLP $\mathcal{F}$ are predicted from prior information rather than learned from the tabular data. We observe that every parameter in the first layer of $\mathcal{T}$ is associated with a input feature j. In particular, $\mathbf{\Theta}^{[1]} \in \mathbb{R}^{d \times h}$ such that $\mathbf{\Theta}^{[1]}_j \in \mathbb{R}^h$ is a vector of parameters connecting input feature j to every hidden unit in the first layer of the MLP (Figure 1e). We observe that if two input features j and k represent real-world entities that are related, then their corresponding parameters $\mathbf{\Theta}^{[1]}_j$ and $\mathbf{\Theta}^{[1]}_k$ are likely to be related too. Typically this intuition is missed since $\mathcal{T}$ learns the parameters $\mathbf{\Theta}^{[1]}_j$ and $\mathbf{\Theta}^{[1]}_k$ associated with two features j and k independently by gradient backpropagation. To capture the intuition that related input features j and k should have related parameters, PLATO's MLP $\mathcal{F}$ instead predicts $\hat{\mathbf{\Theta}}^{[1]}_j$ and $\hat{\mathbf{\Theta}}^{[1]}_k$ from prior information known about j and k. How such parameter prediction occurs is left for Section 3.3. For now, note that the parameters in the first layer of PLATO's MLP $\mathcal{F}$ are predicted such that $\hat{\mathbf{\Theta}}^{[1]} \in \mathbb{R}^{d \times h}$. The parameters $\mathbf{\Theta}^{[2]}, \ldots, \mathbf{\Theta}^{[L]}$ in the remaining layers of PLATO's MLP $\mathcal{F}$ are learned normally.

Second, the parameters in the first layer of PLATO's MLP are allowed to vary for every input sample. In the standard MLP $\mathcal{T}$, all parameters $\mathbf{\Theta}^{[1]}, \ldots, \mathbf{\Theta}^{[L]}$ are the same for every input sample $\mathbf{X}_i$. In the first layer of PLATO's MLP $\mathcal{F}$, however, the parameters $\hat{\mathbf{\Theta}}^{[1]}$ are being predicted from prior information about the input features. We observe that for each input sample $\mathbf{X}_i$, the most relevant prior information about each input feature j might differ. Therefore for each sample $\mathbf{X}_i$, PLATO uses different prior information about each input feature j to predict the parameters $\hat{\mathbf{\Theta}}^{[1]}_j$. As a result, the parameters $\hat{\mathbf{\Theta}}^{[1]}$ in the first layer of $\mathcal{F}$ vary with each input sample $\mathbf{X}_i$, increasing the representational capacity of $\mathcal{F}$. How PLATO uses different prior information for parameter prediction is left to Section 3.3. For now, note that PLATO predicts $\hat{\mathbf{\Theta}}^{[1]}_j$ from prior information about feature j. The prior information about feature j that is used depends on the input sample $\mathbf{X}_i$. As a result, $\hat{\mathbf{\Theta}}^{[1]}$ is conditional on $\mathbf{X}_i$ according to $\hat{\mathbf{\Theta}}^{[1]}|\mathbf{X}_i$.

Formal Notation. Overall, PLATO's MLP $\mathcal{F}$ takes the form

$$\hat{\mathbf{y}}_i = \mathcal{F}(\mathbf{X}_i; \hat{\mathbf{\Theta}}|\mathbf{X}_i). \tag{1}$$

$\mathcal{F}$ has parameters $\hat{\mathbf{\Theta}} = \{\hat{\mathbf{\Theta}}^{[1]}|\mathbf{X}_i\} \cup \{\mathbf{\Theta}^{[2]}, \ldots, \mathbf{\Theta}^{[L]}\}$ where L is the number of layers in $\mathcal{F}$. The parameters $\hat{\mathbf{\Theta}}^{[1]}|\mathbf{X}_i$ in the first layer of $\mathcal{F}$ are predicted from the input sample $\mathbf{X}_i$ via message-passing on the KG according to Section 3.3. For every sample i, a new $\hat{\mathbf{\Theta}}^{[1]}$ is predicted such that $\hat{\mathbf{\Theta}}^{[1]}$ is conditional on $\mathbf{X}_i$ at both training and inference time. The dimensionality of $\hat{\mathbf{\Theta}}^{[1]} \in \mathbb{R}^{d \times h}$ is the same as in a normal MLP where h is the number of hidden units in the first layer of $\mathcal{F}$. The parameters in the remaining layers of PLATO's MLP $\mathbf{\Theta}^{[2]}, \ldots, \mathbf{\Theta}^{[L]}$ are the same as in a standard MLP: they are learned normally, are the same for every sample, and are thus not conditional on $\mathbf{X}_i$.

3.3 Predicting the parameters in the first layer of PLATO's MLP $\mathcal{F}$

PLATO uses prior information about the input features to predict the parameters in the first layer of PLATO's MLP $\mathcal{F}$. PLATO predicts these parameters in three steps. First, PLATO uses self-supervision on the auxiliary KG to pretrain an embedding for every input feature (Section 3.3.1, Figure 1(b)).

Second, since different input samples might rely on different prior information about each input feature, PLATO updates each feature embedding to contain the most relevant prior information about the input feature for the given input sample (Section 3.3.2, Figure 1(c)). Finally, PLATO predicts the parameters in the first layer of $\mathcal{F}$ from the updated feature embeddings with a small neural network that is shared across input features (Section 3.3.3, Figure 1(d)(e)).

3.3.1 Pretraining feature embeddings with self-supervision

First, PLATO learns general prior information about each input feature j from the auxiliary KG G (Figure 1b). PLATO represents the general prior information about each input feature j as a low-dimensional embedding $\mathbf{M}_j \in \mathbb{R}^c$. Since every input feature j corresponds to a specific node in G, PLATO can learn $\mathbf{M}_j$ by learning an embedding for the corresponding feature node in G. Any self-supervised node embedding method on G can be used within PLATO's framework.

Formal notation. Formally, PLATO uses self-supervision on G to pretrain an embedding for every input feature according to

$$\mathbf{M} = \mathcal{H}(G). \tag{2}$$

$\mathbf{M} \in \mathbb{R}^{d \times c}$ is the matrix of all feature embeddings. $\mathcal{H}$ is a self-supervised node embedding method.

We refer to Eq. (2) as pretraining since only the auxiliary KG G is used but the tabular data $\mathbf{X}$, $\mathbf{y}$ is ignored. After pretraining, the feature embeddings $\mathbf{M}$ are fixed.

Implementation. For $\mathcal{H}$, we choose ComplEx, a KG node embedding method [43]. ComplEx uses a self-supervised objective which learn an embedding for every node in G by classifying whether a proposed edge exists in G. ComplEx's proposed edges include both feature nodes and other nodes in G, thereby integrating prior information about the input features and the broader domain.

3.3.2 Updating feature embeddings to contain the most relevant information for an input sample

Since different input samples might rely on different prior information about each input feature, PLATO next updates each feature embedding $\mathbf{M}_j \in \mathbb{R}^c$ to $\mathbf{Q}_j \in \mathbb{R}^c$, a feature embedding which contain the most relevant prior information about feature j for a given input sample $\mathbf{X}_i$ (Figure 1(c)). PLATO uses a message-passing network $\mathcal{Q}$ on the KG to update the feature embeddings in a way that minimizes the number of additional trainable parameters.

$$\mathbf{Q} = \mathcal{Q}(\mathbf{X}_i, \mathbf{M}, G; \mathbf{\Pi}). \tag{3}$$

$\mathcal{Q}$ uses an attention mechanism which considers the sample values $\mathbf{X}_i$ to update the feature embeddings. The attention mechanism has a small number of trainable parameters $\mathbf{\Pi}$.

The message passing network $\mathcal{Q}$. Let $\mathbf{Q}_j^{[r]}$ be the embedding of input feature j after round $r \in \{1, ..., R\}$ of message passing. For every input feature j, $\mathcal{Q}$ first initializes the updated feature embedding to the pretrained feature embedding.

$$\mathbf{Q}_j^{[0]} = \mathbf{M}_j. \tag{3a}$$

$\mathcal{Q}$ then conducts R rounds of message passing. In each round of message passing, the feature embedding $\mathbf{Q}_j^{[r]}$ is updated from the feature embedding of each neighbor k in the prior round $\mathbf{Q}_k^{[r-1]}$ and its own feature embedding in the prior round $\mathbf{Q}_j^{[r-1]}$. The "message" being passed is the embedding of each feature from the prior round.

$$\mathbf{Q}_j^{[r]} = \sigma\Bigg[\overbrace{\beta(\sum_{k \in N_j} \mathbf{A}_{ijk} \mathbf{Q}_k^{[r-1]})}^{\text{Weighted messages from neighbors}} + \underbrace{(1-\beta)\mathbf{Q}_j^{[r-1]}}_{\text{Weighted message from self}} \Bigg]. \tag{3b}$$

σ is an optional nonlinearity. N_j are the neighbors of feature node j in G.

During message-passing, $\mathcal{Q}$ uses two scalar values $\beta \in \mathbb{R}$ and $\mathbf{A}_{ijk} \in \mathbb{R}$ to control the weights of messages. First, $\mathcal{Q}$ uses hyperparameter β to control the weight of the messages aggregated from all

neighbors vs. the message from the feature node j itself. Second, $\mathcal{Q}$ calculates an attention score $\mathbf{A}_{ijk}$ to control the weight of the specific message between feature j and neighbor k. The attention score is different for every sample i and is calculated by a shallow neural network $\mathcal{A}$ with a small number of trainable parameters $\mathbf{\Pi}$. The attention score A_{ijk} thus enables $\mathcal{Q}$ to update the information in the feature embedding in a way that is most relevant for the input sample i. Formally:

$$\mathbf{A}_{ijk} = \frac{\exp\left(\mathcal{A}(\mathbf{X}_{ij}, \mathbf{X}_{ik}; \mathbf{\Pi})\right)}{\sum_{t \in N_j} \exp\left(\mathcal{A}(\mathbf{X}_{ij}, \mathbf{X}_{it}; \mathbf{\Pi})\right)}. \tag{3c}$$

The number of trainable parameters in $\mathbf{\Pi}$ is small since the input of $\mathcal{A}$ is $\mathbb{R}^2$ and the output of $\mathcal{A}$ is a scalar $\mathbb{R}$. $\mathcal{A}$ and its parameters $\mathbf{\Pi}$ are shared for all samples and features.

Finally, the updated feature embeddings $\mathbf{Q}_j$ are set after R rounds of message-passing.

$$\mathbf{Q}_j = \mathbf{Q}_j^{[R]}. \tag{3d}$$

3.3.3 Predicting the first layer of parameters in $\mathcal{F}$ from the updated feature embeddings

Finally, PLATO predicts the parameters in the first layer of $\mathcal{F}$ from each updated feature embedding (Figure 1(d)(e)). Every parameter in the first layer of $\mathcal{F}$ is associated with a feature j. PLATO thus predicts $\hat{\mathbf{\Theta}}_j$, the parameters associated with the feature j, from $\mathbf{Q}_j$, the prior information about j.

Formal notation. PLATO predicts the parameters associated with every input feature j in the first layer of $\mathcal{F}$ according to

$$\hat{\mathbf{\Theta}}_j^{[1]} = \mathcal{B}(\mathbf{Q}_j | \mathbf{X}_i; \mathbf{\Phi}). \tag{4}$$

$\mathcal{B}$ is a shallow neural network parameterized by $\mathbf{\Phi}$. $\mathbf{Q}_j$ is the updated feature embedding of j which is conditional on the input sample $\mathbf{X}_i$. $\mathcal{B}$ and $\mathbf{\Phi}$ are shared for every feature $j \in \{1, \ldots, d\}$.

PLATO drastically reduces the number of trainable parameters vs. a standard MLP. Sharing $\mathcal{B}$ and $\mathbf{\Phi}$ across all input features drastically reduces the number of trainable parameters compared to a standard MLP. For a high-dimensional dataset (*i.e.* $d \gg n$), a standard MLP $\mathcal{T}$ with h hidden units has a large number of trainable parameters in the first layer since $\mathbf{\Theta}^{[1]} \in \mathbb{R}^{d \times h}$. A standard MLP $\mathcal{T}$ must learn these dh trainable parameters independently. By contrast, $\mathcal{B}$ uses a shared set of trainable parameters $\mathbf{\Phi}$ to predict $\hat{\mathbf{\Theta}}_j$ from $\mathbf{Q}_j$ for every $j \in \{1, \ldots, d\}$. The number of trainable parameters in Φ is small compared to dh since $\mathcal{B}$ need only transform every $\mathbf{Q}_j \in \mathbb{R}^c$ to $\hat{\mathbf{\Theta}}^{[\mathbf{1}]} \in \mathbb{R}^h$. Thus, $|\Phi| = ch$ (assuming that $\mathcal{B}$ is a single layer neural network). c, the dimensionality of the feature embedding, is much less than d, the number of input features. As a result, $|\Phi| = ch \ll dh$ and PLATO reduces the number of trainable parameters in the MLP's first layer.

3.4 The PLATO Algorithm

Pseudocode for the PLATO algorithm is in Appendix A.

4 Experiments

Datasets and KG. We evaluate PLATO on 6 tabular datasets with $d \gg n$ and 4 with $d \sim n$ [12, 11, 9, 10]. We construct a general biomedical KG [52, 53, 54, 55, 56, 57, 58] with 108,447 nodes, 3,066,156 edges, and 99 relation types. The same KG is used for all datasets even though the datasets have distinct feature sets with distinct cardinalities. PLATO thus allows a single KG to serve as a unified knowledge backbone for different datasets in a domain. Details are in Appendix F, G.

Fair Comparison of PLATO with Baselines. We compare PLATO to 10 state-of-the art baselines. We consider Ridge Regression [24], dimensionality reduction with PCA [17], feature selection with LASSO [20], deep feature selection with Stochastic Gates [39], and gradient boosted decision trees with XGBoost [26]. We also consider deep tabular methods including a standard MLP, self-attention-based methods with TabTransformer [35] and TabNet [34], differentiable decision trees with NODE [30], and parameter-prediction with Diet Networks [51]. We also tried FT-Transformer [15], but it experienced out of memory issues on all datasets due to the large number of features.

Table 1: PLATO outperforms baselines when $d \gg n$. Best model in **bold**. Second best model underlined.

Dataset		MNSCLC	CM	PDAC	BRCA	CRC	CH
D		15,390	13,183	12,932	12,693	18,206	19,902
N		295	286	321	476	562	924
D/N		52.2	46.1	40.3	28.2	22.6	19.7
Classic Stat ML	Ridge	0.153±0.000	0.390±0.000	0.344±0.000	0.538±0.000	0.376±0.000	0.546±0.000
Dim. Reduct.	PCA	0.156±0.113	0.070±0.000	0.232±0.121	0.452±0.000	0.193±0.163	0.237±0.232
Feat. Select.	LASSO	0.168±0.000	0.431±0.000	0.346±0.000	0.470±0.000	0.400±0.000	0.547±0.000
	STG	0.132±0.130	0.366±0.043	0.258±0.055	0.485±0.037	0.301±0.010	0.262±0.076
Decision Tree	XGBoost	-0.02±0.000	0.225±0.000	0.363±0.000	0.347±0.000	0.354±0.000	0.728±0.000
Param. Pred.	Diet	-0.04±0.205	0.054±0.149	0.309±0.096	0.213±0.036	0.087±0.112	0.148±0.008
Tabular DL	MLP	0.128±0.126	0.322±0.043	0.289±0.047	0.240±0.067	0.355±0.022	0.044±0.039
	NODE	0.003±0.000	0.150±0.000	0.190±0.000	0.512±0.000	0.344±0.000	0.181±0.000
	TabTransformer	0.265±0.000	0.072±0.000	0.029±0.000	0.202±0.000	0.238±0.000	0.020±0.000
	TabNet	0.085±0.028	0.010±0.068	0.088±0.037	0.055±0.037	0.018±0.016	0.039±0.026
Ours	PLATO	**0.272±0.130**	**0.435±0.022**	**0.400±0.021**	**0.583±0.019**	**0.401±0.019**	**0.770±0.003**

Table 2: PLATO depends on feature embeddings that contain information specific to a given sample.

Parameter Pred. $\mathcal{B}$ Input	Feature Information	Sample Specific	PearsonR
Updated feat. embed. $\mathbf{Q}$	✔	✔	0.583±0.019
General feat. embed $\mathbf{M}$	✔	✗	0.522±0.030
None	✗	✗	0.240±0.067

Table 3: PLATO depends on both feature nodes and other nodes representing domain knowledge.

Auxiliary KG	Feature-only Knowledge	Broader Knowledge	PearsonR
Full KG	✔	✔	0.583±0.019
Feature-only KG	✔	✗	0.539±0.038
No KG	✗	✗	0.240±0.067

To ensure a fair comparison, we follow evaluation protocols in tabular benchmarks [14, 15]. We conduct a random search with 500 configurations of every model (including PLATO) on every dataset across a broad range of hyperparameters (Appendix B). We split data with a 60/20/20 training, validation, test split. All results are computed across 3 data splits and 3 runs of each model in each data split. We report the mean and standard deviation of the Pearson correlation (PearsonR) between $\mathbf{y}$ and $\hat{\mathbf{y}}$ across runs and splits on the test set. Each model is run on a GeForce RTX 2080 TI GPU.

4.1 Results

PLATO outperforms statistical and deep baselines when $d \gg n$. PLATO outperforms all baselines across all 6 datasets with $d \gg n$ (Table 1). PLATO achieves the largest improvement on the PDAC dataset, improving by 10.19% vs. XGBoost, the best baseline for PDAC (0.400 vs. 0.363). While PLATO achieves the strongest performance across all 6 datasets, the best performing baseline varies with different datasets. Ridge Regression is the strongest baseline for BRCA, LASSO for CM and CRC, XGBoost for PDAC and CH, and TabTransformer for MNSCLC. The remaining baselines (PCA, STG, Diet Networks, MLP, NODE, and TabNet) are not the strongest baseline for any dataset. We also find that the performance of a specific baseline varies with the dataset. TabTransformer, for example, is the best baseline for MNSCLC but the worst for CH (model ranking in Appendix D).

PLATO's performance depends on updating feature embeddings to contain information relevant to a sample. PLATO predicts the parameters $\hat{\mathbf{\Theta}}^{[1]}$ in the first layer of a modified MLP $\mathcal{F}$ by using feature embeddings which contain prior information about the input features. We conduct an ablation study to compare PLATO's performance when it uses feature embeddings which contain the relevant information for a given sample $\mathbf{X}_i$ (*i.e.* $\hat{\mathbf{\Theta}}^{[1]} = \mathcal{B}(\mathbf{Q}|\mathbf{X}_i)$) vs. the pretrained feature embeddings $\mathbf{M}$ which contain general information about the input features that is not specific to a given sample (*i.e.* $\hat{\mathbf{\Theta}}^{[1]} = \mathcal{B}(\mathbf{M})$). We also compare to a PLATO ablation that does not use any feature embeddings and is thus a standard MLP. Using general feature embeddings $\mathbf{M}$ improves over not using feature embeddings (0.522 vs. 0.240). Using feature embeddings $\mathbf{Q}$ that are specific to a given input sample further improves performance (0.583 vs. 0.522). Therefore, updating the feature embeddings to $\mathbf{Q}$ so they contain the information specific to a given sample is critical to PLATO's performance.

Table 4: PLATO is competitive with baselines when $d \sim n$. Best model in **bold**. Second best model underlined.

Dataset		ME	BC	SCLC	NSCLC
D		19,902	18,261	18,437	18,308
N		10,064	10,101	10,712	16,730
D/N		2.0	1.8	1.7	1.1
Classic Stat ML	Ridge	0.566±0.008	0.483±0.008	0.604±0.057	0.679±0.008
Dim. Reduct.	PCA	0.239±0.310	0.233±0.294	0.284±0.274	0.645±0.000
Feat. Select.	LASSO	0.667±0.000	0.633±0.000	0.669±0.000	0.637±0.000
	STG	0.676±0.000	0.643±0.000	0.668±0.000	0.646±0.000
Decision Tree	XGBoost	**0.875±0.000**	0.826±0.000	0.878±0.000	**0.843±0.000**
Param. Pred.	Diet	0.105±0.000	0.037±0.000	-0.05±0.000	0.002±0.000
Tabular DL	MLP	0.487±0.131	0.508±0.061	0.537±0.061	0.573±0.005
	NODE	0.870±0.000	0.420±0.169	0.801±0.102	0.487±0.197
	TabTransformer	0.305±0.028	0.010±0.000	0.288±0.203	0.503±0.187
	TabNet	0.667±0.002	0.624±0.001	0.657±0.004	0.647±0.000
Ours	PLATO	**0.875±0.004**	**0.844±0.003**	**0.883±0.002**	0.839±0.000

PLATO's performance depends on both feature nodes and broader knowledge nodes in the auxiliary KG. PLATO relies on an auxiliary KG G which contains information about input features and information about the broader domain. Information about input features is represented as feature nodes while information about the broader domain is represented as other nodes in G (Methods 3.1). To test the relative importance of the feature information in G vs. the broader domain information, we measured the performance of PLATO on the BRCA dataset in two KG configurations: PLATO with the full KG (*i.e.* both the feature nodes and the broader domain nodes) and PLATO with a "feature-only KG" (*i.e.* an induced subgraph on only the feature nodes) (Table 3). We also compare to a "No KG" configuration in which PLATO does not have access to the KG and is thus ablated to a standard MLP.

We find that both the feature nodes and the broader knowledge nodes are important for PLATO's performance. Using the "feature-only KG" configuration of PLATO improves performance vs the "no KG" configuration of PLATO (0.539 vs 0.240). Using the "full KG" configuration further improves performance vs the "feature-only KG" configuration (0.583 vs 0.539). PLATO's performance thus relies on both the feature information and the broader domain information in the KG.

For datasets with $d \sim n$, PLATO is competitive with baselines. Finally, we test PLATO's performance for datasets with $d \sim n$. We test 4 datasets with $d \sim n$ ranging from $\frac{d}{n} = 1.1$ to $\frac{d}{n} = 2.0$ (Table 4). We find that on 4 datasets with $d \sim n$, PLATO is competitive with the best performing baseline, XGBoost, but does not improve performance substantially. PLATO's stronger performance for datasets with $d \gg n$ than for datasets with $d \sim n$ is justified. PLATO's key idea is to include auxiliary information about the input features. Auxiliary information is likely to help performance the most in settings with the least labeled data (*i.e.* $d \gg n$). When $d \sim n$, auxiliary information is less helpful since the tabular dataset may already have enough information to train a strong predictive model. We further find that XGBoost is consistently the strongest baseline for datasets with $d \sim n$, in contrast to the varied performance of XGBoost on the datasets with $d \gg n$ (Table 1).

5 Discussion

PLATO is a deep learning model for tabular data with $d \gg n$ and an auxiliary KG with input features as nodes. Across 6 datasets, PLATO achieves state-of-the-art, including improvements up to 10.19%. Ablations validate each component of PLATO. PLATO has several limitations. First, PLATO matches but does not improve the performance of baselines for high-dimensional datasets with more samples (*i.e.* $d \sim n$). Second, PLATO relies on the coverage of prior information. Datasets with input features that have less prior information in the KG are less likely to benefit from PLATO. Overall, PLATO uses an auxiliary KG about input features to enable tabular deep learning when $d \gg n$.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? Code and data will be released with the archival version of the manuscript.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 under "Fair Comparison of PLATO with Baselines" and Appendix B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See figures and tables.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4 under "Fair Comparison of PLATO with Baselines" header.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] All datasets and baseline models are cited.
   (b) Did you mention the license of the assets? [Yes] All licenses are present in the cited work.
   (c) Did you include any new assets either in the supplemental material or as a URL? Code and data will be released with the archival version of the manuscript.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Acknowledgments. We thank Maria Brbic, Michael Moor, Kaidi Cao, Weihua Hu, and Rajas Bansal for discussions and for providing feedback on our manuscript. C.R. is supported by a Siebel Fellowship, a National Science Foundation Graduate Research Fellowship under Grant No. DGE-1656518, and a Stanford Enhancing Diversity in Graduate Education (EDGE) Fellowship. H.R. is supported by a Masason Foundation, an Apple Fellowship, and a Baidu Scholarship. We also gratefully acknowledge the support of DARPA under Nos. HR00112190039 (TAMI), N660011924033 (MCS); ARO under Nos. W911NF-16-1-0342 (MURI), W911NF-16-1-0171 (DURIP); NSF under Nos. OAC-1835598 (CINES), OAC-1934578 (HDR), CCF-1918940 (Expeditions), NIH under No. 3U54HG010426-04S1 (HuBMAP), Stanford Data Science Initiative, Wu Tsai Neurosciences Institute, Amazon, Docomo, GSK, Hitachi, Intel, JPMorgan Chase, Juniper Networks, KDDI, NEC, and Toshiba. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding entities.

References

[1] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[3] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[5] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv:2204.06125*, 2022.

[6] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[7] 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 Organization*, pages 2287–2293, 2017.

[8] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems*, volume 17, pages 545–552, 2004.

[9] 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.

[10] 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.

[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] 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.

[13] Gregor Kasieczka, Benjamin Nachman, David Shih, Oz Amram, Anders Andreassen, Kees Benkendorder, Blaz Bortolato, Gustaaf Broojimans, Florencia Canelli, Jack Collins, et al. The LHC Olympics 2020: a community challenge for anomaly detection in high energy physics. *Reports on Progress in Physics*, 84:124201, 2021.

[14] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2022.

[15] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, volume 34, pages 18932–18943, 2021.

[16] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[17] Hervé Abdi and Lynne J Williams. Principal Component Analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[18] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11):2579–2605, 2008.

[19] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. Dimensionality reduction: a comparative. *Journal of Machine Learning Research*, 10(66-71):13, 2009.

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

[21] Héctor Climente-González, Chloé-Agathe Azencott, Samuel Kaski, and Makoto Yamada. Block HSIC lasso: model-free biomarker detection for ultra-high dimensional data. *Bioinformatics*, 35(14):i427–i435, 2019.

[22] Tobias Freidling, Benjamin Poignard, Héctor Climente-González, and Makoto Yamada. Post-selection inference with HSIC-Lasso. In *International Conference on Machine Learning*, pages 3439–3448. PMLR, 2021.

[23] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group LASSO for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[24] Donald W Marquardt and Ronald D Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, 1975.

[25] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[26] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[27] 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. In *Advances in Neural Information Processing Systems*, volume 30, pages 3146–3154, 2017.

[28] Yin Lou and Mikhail Obukhov. BDT: Gradient boosted decision tables for high accuracy and scoring efficiency. In *Proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1893–1901, 2017.

[29] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31, pages 6639–6649, 2018.

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

[31] 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.

[32] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1475, 2015.

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

[34] Sercan Ö Arik and Tomas Pfister. TabNet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

[35] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[36] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1161–1170, 2019.

[37] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[38] Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 28742–28756, 2021.

[39] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In *International Conference on Machine Learning*, pages 10648–10659. PMLR, 2020.

[40] Muhammed Fatih Balın, Abubakar Abid, and James Zou. Concrete autoencoders: Differentiable feature selection and reconstruction. In *International Conference on Machine Learning*, pages 444–453. Proceedings of Machine Learning Research, 2019.

[41] Yang Lu, Yingying Fan, Jinchi Lv, and William Stafford Noble. DeepPINK: reproducible feature selection in deep neural networks. In *Advances in Neural Information Processing Systems*, volume 31, pages 8690–8700, 2018.

[42] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[43] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080. PMLR, 2016.

[44] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, pages 1112–1119, 2014.

[45] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*, 2015.

[46] Claudia d'Amato, Nicola Flavio Quatraro, and Nicola Fanizzi. Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In *European Semantic Web Conference*, pages 441–457. Springer, 2021.

[47] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, volume 26, pages 2148–2156, 2013.

[48] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[49] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. Learning a synaptic learning rule. In *International Joint Conference on Neural Networks*, volume 2, pages 969–975. IEEE, 1991.

[50] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2016.

[51] 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.

[52] Katja Luck et al. A reference map of the human binary protein interactome. *Nature*, 580(7803):402–408, April 2020.

[53] Sebastian Köhler et al. The Human Phenotype Ontology in 2017. *Nucleic Acids Research*, 45(D1):D865–D876, November 2016.

[54] Michael Kuhn, Ivica Letunic, Lars Juhl Jensen, and Peer Bork. The SIDER database of drugs and side effects. *Nucleic Acids Research*, 44(D1):D1075–D1079, October 2015.

[55] Camilo Ruiz, Marinka Zitnik, and Jure Leskovec. Identification of disease treatment mechanisms through the multiscale interactome. *Nature Communications*, 12(1):1–15, 2021.

[56] Damian Szklarczyk, Annika L Gable, Katerina C Nastou, David Lyon, Rebecca Kirsch, Sampo Pyysalo, Nadezhda T Doncheva, Marc Legeay, Tao Fang, Peer Bork, Lars J Jensen, and Christian von Mering. The STRING database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic Acids Research*, 49(D1):D605–D612, November 2020.

[57] David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, Nazanin Assempour, Ithayavani Iynkkaran, Yifeng Liu, Adam Maciejewski, Nicola Gale, Alex Wilson, Lucy Chin, Ryan Cummings, Diana Le, Allison Pon, Craig Knox, and Michael Wilson. DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Research*, 46(D1):D1074–D1082, November 2017.

[58] David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, et al. DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Research*, 46(D1):D1074–D1082, 2017.

[59] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems*, volume 34, pages 28877–28888, 2021.

[60] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

[61] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133, 2020.

[62] Soufiane MC Mourragui, Marco Loog, Daniel J Vis, Kat Moore, Anna G Manjon, Mark A van de Wiel, Marcel JT Reinders, and Lodewyk FA Wessels. Predicting patient response with models trained on cell lines and patient-derived xenografts by nonlinear transfer learning. *Proceedings of the National Academy of Sciences*, 118(49):e2106682118, 2021.

[63] Anna Bauer-Mehren, Michael Rautschka, Ferran Sanz, and Laura I Furlong. DisGeNET: a cytoscape plugin to visualize, integrate, search and analyze gene–disease networks. *Bioinformatics*, 26(22):2924–2926, 2010.

[64] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreutz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O'Donnell, Genie Leung, Rochelle McAdam, et al. The BioGRID interaction database: 2019 update. *Nucleic Acids Research*, 47(D1):D529–D541, 2019.

[65] Jean-François Rual, Kavitha Venkatesan, Tong Hao, Tomoko Hirozane-Kishikawa, Amélie Dricot, Ning Li, Gabriel F Berriz, Francis D Gibbons, Matija Dreze, Nono Ayivi-Guedehoussou, et al. Towards a proteome-scale map of the human protein–protein interaction network. *Nature*, 437(7062):1173–1178, 2005.

[66] Lukasz Salwinski, Christopher S Miller, Adam J Smith, Frank K Pettit, James U Bowie, and David Eisenberg. The Database of Interacting Proteins: 2004 update. *Nucleic Acids Research*, 32(suppl_1):D449–D451, 2004.

[67] Jörg Menche, Amitabh Sharma, Maksim Kitsak, Susan Dina Ghiassian, Marc Vidal, Joseph Loscalzo, and Albert-László Barabási. Uncovering disease-disease relationships through the incomplete interactome. *Science*, 347(6224):1257601, 2015.

[68] Thomas Rolland, Murat Taşan, Benoit Charloteaux, Samuel J Pevzner, Quan Zhong, Nidhi Sahni, Song Yi, Irma Lemmens, Celia Fontanillo, Roberto Mosca, et al. A proteome-scale map of the human interactome network. *Cell*, 159(5):1212–1226, 2014.

[69] Haiyuan Yu, Leah Tardivo, Stanley Tam, Evan Weiner, Fana Gebreab, Changyu Fan, Nenad Svrzikapa, Tomoko Hirozane-Kishikawa, Edward Rietman, Xinping Yang, et al. Next-generation sequencing to generate interactome datasets. *Nature Methods*, 8(6):478–480, 2011.

[70] Kavitha Venkatesan, Jean-Francois Rual, Alexei Vazquez, Ulrich Stelzl, Irma Lemmens, Tomoko Hirozane-Kishikawa, Tong Hao, Martina Zenkner, Xiaofeng Xin, Kwang-Il Goh, et al. An empirical framework for binary interactome mapping. *Nature Methods*, 6(1):83–90, 2009.

[71] Gene Ontology Consortium. The Gene Ontology resource: 20 years and still GOing strong. *Nucleic Acids Research*, 47(D1):D330–D338, 2018.

Appendix

A The PLATO Algorithm

Algorithm 1: The PLATO Algorithm.

Input: a data sample $\mathbf{X}_i \in \mathbb{R}^d$, a knowledge graph (KG) $G = (V, E)$
Output: predicted label $\hat{\mathbf{y}}_i \in \mathbb{R}$

Pretrain KG embedding for every feature: $\mathbf{M} = \mathcal{H}(G)$

Initialize feature embedding for feature j: $\mathbf{Q}_j^{[0]} = \mathbf{M}_j$

Compute sample i-specific attention weight: $\mathbf{A}_{ijk} = \frac{\exp\left(\mathcal{A}(\mathbf{X}_{ij}, \mathbf{X}_{ik}; \mathbf{\Pi})\right)}{\sum_{t \in N_j} \exp\left(\mathcal{A}(\mathbf{X}_{ij}, \mathbf{X}_{it}; \mathbf{\Pi})\right)}$, $\forall$ features j, k, where $\mathcal{A}$ is a NN parameterized by $\mathbf{\Pi}$

for $r = 1;\ r \leq R$ **do**

  Update feature embedding with message passing neural network at layer r:

  $$\mathbf{Q}_j^{[r]} = \sigma\left[\beta(\textstyle\sum_{k \in N_j} \mathbf{A}_{ijk} \mathbf{Q}_k^{[r-1]}) + (1-\beta)\mathbf{Q}_j^{[r-1]}\right]$$

end

Obtain feature j embedding from GNN last layer R: $\mathbf{Q}_j = \mathbf{Q}_j^{[R]}$

Predict the parameter of first layer of a MLP from the feature embedding: $\hat{\mathbf{\Theta}}^{[1]} = \mathcal{B}(\mathbf{Q}|\mathbf{X}_i; \mathbf{\Phi})$, where $\mathcal{B}$ is a NN parameterized by $\mathbf{\Phi}$

Concatenate the first layer predicted embedding with the parameters from the rest of layers: $\hat{\mathbf{\Theta}} = \{\hat{\mathbf{\Theta}}^{[1]}\} \cup \{\mathbf{\Theta}^{[2]}, \ldots, \mathbf{\Theta}^{[L]}\}$

Predict label: $\hat{\mathbf{y}}_i = \mathcal{F}(\mathbf{X}; \hat{\mathbf{\Theta}}|\mathbf{X}_i)$, where $\mathcal{F}$ is an MLP parameterized by $\hat{\mathbf{\Theta}}$

B Evaluation Protocol and Hyperparameter Ranges

To ensure a fair comparison with baselines, we follow evaluation protocols outlined in tabular benchmarks [14, 15]. We conduct a random search with 500 configurations of every model (including PLATO) on every dataset across a broad range of hyperparameters. We base the hyperparameter ranges on the ranges used in prior tabular learning benchmarks [14, 15] and the ranges mentioned in the original papers of the methods. Hyperparameter ranges for PLATO are given in Supplementary Table 1. Hyperparameter ranges for baseline methods are given in Supplementary Table 2.

Module in PLATO	Hyperparameter	Range
General	Learning rate	LogUniform(1e-4, 5e-3)
	Batch size	[16, 32, 64]
	L2	0, LogUniform(1e-5, 1e-2)
KG $\mathcal{H}$	Embedding dimension c	200
	Embedding model	ComplEx
Message Passing (MP) $\mathcal{Q}$	# MP layers R	2
	Beta	LogUniform(1e-4, 1e-1)
	Hidden dimension in $\mathcal{A}$	UniformInt(16, 512)
Param Prediction $\mathcal{B}$	# Layers R	UniformInt(2, 6)
	Hidden dimension in h	UniformInt(16, 512)

Supplementary Table 1: Hyperparameter ranges used for PLATO.

Model	Hyperparameter	Range
LASSO	L1	LogUniform(1E-4, 10)
Ridge	L2	LogUniform(1E-4, 10)
XGBoost	n-estimators	UniformInt(1,2000)
	Max depth	UniformInt(3, 10)
	Min weight	LogUniform(1E-8,1E5)
	Subsample	Uniform(0.5, 1)
	Learning rate	LogUniform(1E-5,1)
	Col sample by level	Uniform(0.5, 1)
	Col sample by tree	Uniform(0.5, 1)
	Gamma	0, LogUniform(1E-8, 1E2)
	Lambda	0, LogUniform(1E-8, 1E2)
	Alpha	0, LogUniform(1E-8, 1E2)
	Booster	"gbtree"
	Early-stopping-rounds	50
	Iterations	100
PCA	Number of PCA Components	UniformInt(2,1000)
STG	Hidden dimension	UniformInt(10, 500)
	Number of layers	UniformInt(1, 5)
	Activation	[Tanh, Relu, Sigmoid]
	Learning rate	LogUniform(1e-4, 1e-1)
	Sigma	Uniform(0.001, 2)
	Lambda	LogUniform(1e-3, 10)
MLP	Number of layers	UniformInt(1, 8)
	Hidden dimension	UniformInt(1, 512)
	Dropout	0, Uniform([0,0.5])
	Learning rate	LogUniform(1e-5, 1e-2)
	L2	0, LogUniform(1e-6, 1e-3)
TabNet	Decision Steps	UniformInt(3, 10)
	Layer size	2, 4, 8, 16, 32, 64
	Relaxation factor	Uniform[1, 2]
	Sparsity loss weight	LogUniform[1e-6, 1e-1]
	Decay rate	Uniform[0.4, 0.95]
	Decay steps	100, 500, 2000
	Learning rate	Uniform(1e-3, 1e-2)
	Iterations	100
TabTransformer	Embedding dimension	4, 8, 16, 32, 64, 128
	Number of heads	UniformInt(1, 10)
	Number of attention blocks	UniformInt(1, 12)
	Attention dropout rate	Uniform(0, 0.5)
	Add norm dropout	Uniform(0, 0.5)
	Transformation activation	[Tanh, Relu, LeakyReLU]
	L2	LogUniform(1e-6, 1e-1)
	Learning rate	LogUniform(1e-6, 1e-3)
	FF dropout	Uniform(0, 0.5)
	FF hidden multiplier	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
	Out FF activation	[Tanh, Relu, LeakyReLU]
	Out FF dropout	Uniform(0, 0.5)
NODE	Learning rate	LogUniform(1e-5, 1)
	Number of layers	UniformInt(1, 10)
	Number of trees	UniformInt(2, 2048)
	Depth	UniformInt(1, 10)
Diet Network	Embedding choice	X^T, random
	Number of layers	UniformInt(1, 8)
	Hidden dimension	UniformInt(1, 512)
	Dropout	0, Uniform([0,0.5])
	Learning rate	LogUniform(1e-5, 1e-2)
	L2	0, LogUniform(1e-6, 1e-3)

Supplementary Table 2: Hyperparameter range for all baselines.

C Graph classification approaches

Graph classification models are not applicable for PLATO's setting. In graph classification models, every input sample is a graph with node attributes, and a model must make a prediction for that graph. The PLATO problem setting breaks fundamental assumptions made by graph classification models, rendering them not applicable. First, graph classification models assume that different samples correspond to different graphs [59, 60, 61]. However, in PLATO every sample corresponds to the exact same graph. There is a single background knowledge graph for all samples, and every sample has input features that correspond to the exact same nodes within the knowledge graph. Second, graph classification approaches typically assume that every node in an input graph has a node attribute [59, 60, 61]. However, in PLATO only a small subset of the nodes in the knowledge graph have measured feature values. Finally, graph classification approaches typically assume small graphs: the largest graph classification task in the Open Graph Benchmark has only 244 nodes [61]. However in PLATO, the knowledge graph contains 108,447 nodes and the smallest dataset has 12,693 features corresponding to nodes.

D Rank ordering of methods for datasets with $d \gg n$

In Supplementary Table 3, we show the rank order performance of all models on all $d \gg n$ datasets. We find that PLATO exhibits consistent and strong performance while the performance of the baselines depends on the specific $d \gg n$ dataset. For example, TabTransformer is the second best performing of all models on the MNSCLC dataset but the worst performing of all models on the PDAC and CH datasets. Similarly, XGBoost is the second best performing of all models on PDAC but only the seventh best performing of all models on BRCA. The baselines with the most stable performance are LASSO and Ridge Regression which rank consistently between the second and fifth best of all models.

Supplementary Table 3: For datasets with $d \gg n$, PLATO exhibits consistent and strong performance. By contrast, the performance of the baselines varies with each dataset. For every dataset, the rank order of performance from Table 1 is shown. The best overall model is in **bold** and the second best model is underlined.

Dataset		MNSCLC	CM	PDAC	BRCA	CRC	CH
D/N		52.2	46.1	40.3	28.2	22.6	19.7
Classic Stat ML	Ridge	5	3	4	<u>2</u>	3	4
Dim. Reduct.	PCA	4	9	8	6	9	6
Feat. Select.	LASSO	3	<u>2</u>	3	5	<u>2</u>	3
	STG	6	4	7	4	7	5
Decision Tree	XGBoost	11	6	<u>2</u>	7	5	<u>2</u>
Param. Pred.	Diet	10	10	5	9	10	8
Tabular DL	MLP	7	5	6	8	4	9
	NODE	9	7	9	3	6	7
	TabTransformer	<u>2</u>	8	11	10	8	11
	TabNet	8	11	10	11	11	10
Ours	PLATO	**1**	**1**	**1**	**1**	**1**	**1**

Supplementary Table 4: For datasets with $d \sim n$, PLATO is competitive with baselines. For XGBoost is consistently the strongest baseline. For every dataset, the rank order of performance from Table 4 is shown. The best overall model is in **bold** and the second best model is underlined.

Dataset		ME	BC	SCLC	NSCLC
D/N		2.0	1.8	1.7	1.1
Classic Stat ML	Ridge	7	7	7	3
Dim. Reduct.	PCA	10	9	10	6
Feat. Select.	LASSO	6	4	4	7
	STG	4	3	5	5
Decision Tree	XGBoost	2	2	2	**1**
Param. Pred.	Diet	11	10	11	11
Tabular DL	MLP	8	6	8	8
	NODE	3	8	3	10
	TabTransformer	9	11	9	9
	TabNet	5	5	6	4
Ours	PLATO	**1**	**1**	**1**	2

E Code Details

Code to run PLATO will be included as a supplementary file in the archival version of the manuscript.

F Dataset Details

We compiled 6 datasets with $d \gg n$ and 4 datasets with $d \sim n$. In all datasets, a machine learning model must predict the response of a cell or mouse to a drug. In the tabular data, every row corresponds to a specific cell or mouse. Every column corresponds to a gene name. Every value corresponds to the amount of that gene in the tumor of the specific cell or mouse. The label is the response of the cell or mouse. All genes are nodes in the knowledge graph. In practice, the number of genes is large for all tasks and the number of samples is comparatively small making the drug response setting appropriate for $d \gg n$.

Dataset statistics, names, and sources are given below. Dataset pre-processing followed a standard process described in [62]. Briefly, gene expression values underwent TMM normalization and log transformation (*i.e.* $\log(x+1)$). Values were made to have zero mean and unit standard deviation. As labels, we used ln-ic50 for datasets from [9, 10, 11] and the minimum average percent tumor growth (*i.e.* "min-avg-pct-tumor-growth") for datasets from [12]. For all datasets, we use a 200-dimensional ComplEx [43] embedding of the drug as the input feature vector. All datasets will be released in the archival version of the manuscript.

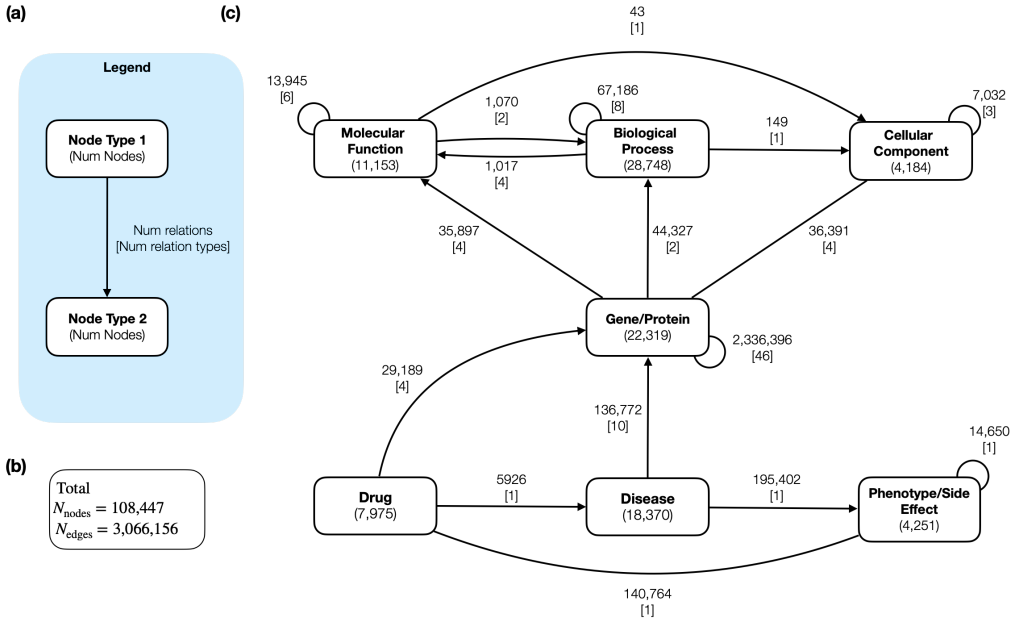
Dataset	Name	Source
MNSCLC	Non-small cell lung carcinoma	[12]
CM	Cutaneous melanoma	[12]
PDAC	Pancreatic ductal carcinoma	[12]
BRCA	Breast carcinoma	[12]
CRC	Colorectal cancer	[12]
CH	Chondrosarcoma	[9, 10, 11]
ME	Melanoma	[9, 10, 11]
BC	Breast carcinoma	[9, 10, 11]
SCLC	Small cell lung carcinoma	[9, 10, 11]
NSCLC	Non-small cell lung carcinoma	[9, 10, 11]

G Knowledge Graph Details

As a unified knowledge backbone for the datasets, we compile a general biomedical knowledge graph from prior studies [52, 53, 54, 55, 56, 57, 58]. A schematic of the KG is in Supplementary Figure 2. A detailed breakdown of relation types is in Supplementary Table 5.

Our knowledge graph contains 108,447 total nodes, including 7,975 drugs, 18,370 diseases, 11,447 phenotypes, 22,319 genes, 11,153 molecular functions, 28,748 biological processes, and 4,184 cellular components. Our knowledge graph contains 3,066,156 edges with 99 distinct relation types. All datasets include features which map to a subset of nodes in the knowledge graph, primarily genes and drugs. The remaining node types and their relationships serve as broader domain knowledge.

Edges between drug nodes and gene/protein nodes were derived from Drugbank [58], Gao [12], and the Genomics of Drug Sensitivity in Cancer [10, 9, 11]. Edges between diseases and genes/proteins were derived from DisGeNet [63]. Edges between diseases and phenotypes were derived from the Human Phenotype Ontology [53]. Edges between drugs and diseases were derived from the Multiscale Interactome [55]. Edges between drugs and side effects were derived from SIDER [54]. Edges between genes/proteins and other genes/proteins were derived from BioGRID [64], [65], the Database of Interacting Proteins [66], [52], [67], [68], [69], [70], and STRING [56]. Finally, edges from genes/proteins to molecular functions, biological processes, and cellular components as well as edges between molecular functions, biological processes, and cellular components were derived from the Gene Ontology [71]. The full knowledge graph will be included as a supplementary file in the archival version of the manuscript.



Supplementary Figure 2: Knowledge graph as a unified knowledge backbone. We constructed a knowledge graph as a unified knowledge backbone across all 6 datasets. (a) Legend. For each node type, the number of nodes is given in parentheses. Between node types, the number of edges and the number of relation types are given. (b) Number of total nodes and edges across entire knowledge graph. (c) Visual schematic of knowledge graph across each node type.

Head type	Relation	Tail type	# edges
BiologicalProcess	EndsDuring	BiologicalProcess	1
BiologicalProcess	HappensDuring	BiologicalProcess	8
BiologicalProcess	HasPart	BiologicalProcess	229
BiologicalProcess	IsA	BiologicalProcess	53015
BiologicalProcess	NegativelyRegulates	BiologicalProcess	2768
BiologicalProcess	PartOf	BiologicalProcess	5193
BiologicalProcess	PositivelyRegulates	BiologicalProcess	2756
BiologicalProcess	Regulates	BiologicalProcess	3216
BiologicalProcess	OccursIn	CellularComponent	149
BiologicalProcess	HasPart	MolecularFunction	173
BiologicalProcess	NegativelyRegulates	MolecularFunction	269
BiologicalProcess	PositivelyRegulates	MolecularFunction	274
BiologicalProcess	Regulates	MolecularFunction	301
CellularComponent	HasPart	CellularComponent	179
CellularComponent	IsA	CellularComponent	4863
CellularComponent	PartOf	CellularComponent	1990
Disease	AlteredExpression	Gene	7157
Disease	Biomarker	Gene	107160
Disease	ChromosomalRearrangement	Gene	162
Disease	FusionGene	Gene	166
Disease	GeneticVariation	Gene	15076
Disease	GermlineCausalMutation	Gene	4677
Disease	ModifyingMutation	Gene	10
Disease	SomaticCausalMutation	Gene	130
Disease	SusceptibilityMutation	Gene	441
Disease	Therapeutic	Gene	1793
Disease	Has	Phenotype	195402
Drug	Treats	Disease	5926
Drug	Carries	Gene	866
Drug	Enzymes	Gene	5382
Drug	Targets	Gene	19817
Drug	Transports	Gene	3124
Drug	Has	Phenotype	140764
Gene	Associates	BiologicalProcess	43857
Gene	NotAssociates	BiologicalProcess	470
Gene	Associates	CellularComponent	35306
Gene	Colocalizes	CellularComponent	914
Gene	NotAssociates	CellularComponent	160
Gene	NotColocalizes	CellularComponent	11
Gene	Acetylation	Gene	9
Gene	Activation	Gene	58502
Gene	AdpRibosylation	Gene	2
Gene	Ampylation	Gene	5
Gene	Association	Gene	18
Gene	Binary	Gene	56565
Gene	Binding	Gene	287641
Gene	Catalysis	Gene	344801
Gene	Cleavage	Gene	22
Gene	Complexes	Gene	62552
Gene	CovalentBinding	Gene	52
Gene	Deacetylation	Gene	8
Gene	Demethylation	Gene	6
Gene	Dephosphorylation	Gene	26
Gene	Deubiquitination	Gene	18

Gene	DirectInteraction	Gene	2904
Gene	DisulfideBond	Gene	5
Gene	DosageGrowthDefect	Gene	9
Gene	DosageLethality	Gene	112
Gene	DosageRescue	Gene	63
Gene	Enzymatic	Gene	2
Gene	Expression	Gene	188
Gene	GeneticInterference	Gene	32
Gene	Hydroxylation	Gene	26
Gene	Inhibition	Gene	20108
Gene	Kinase	Gene	11960
Gene	Literature	Gene	174162
Gene	Metabolic	Gene	10646
Gene	Methylation	Gene	25
Gene	NegativeGenetic	Gene	3449
Gene	OxidoreductaseActivityElectronTransferAssay	Gene	2
Gene	PhenotypicEnhancement	Gene	209
Gene	PhenotypicSuppression	Gene	214
Gene	Phosphorylation	Gene	166
Gene	Phosphotransfer	Gene	1
Gene	PhysicalAssociation	Gene	824164
Gene	PositiveGenetic	Gene	2331
Gene	PostTranslationalModification	Gene	5306
Gene	ProteinCleavage	Gene	48
Gene	PutativeSelfInteraction	Gene	3
Gene	Reaction	Gene	400658
Gene	Regulation	Gene	2650
Gene	Signaling	Gene	65412
Gene	SyntheticGrowthDefect	Gene	407
Gene	SyntheticLethality	Gene	816
Gene	SyntheticRescue	Gene	91
Gene	Associates	MolecularFunction	35012
Gene	Contributes	MolecularFunction	596
Gene	NotAssociates	MolecularFunction	285
Gene	NotContributes	MolecularFunction	4
MolecularFunction	PartOf	BiologicalProcess	1068
MolecularFunction	Regulates	BiologicalProcess	2
MolecularFunction	OccursIn	CellularComponent	43
MolecularFunction	HasPart	MolecularFunction	204
MolecularFunction	IsA	MolecularFunction	13631
MolecularFunction	NegativelyRegulates	MolecularFunction	42
MolecularFunction	PartOf	MolecularFunction	11
MolecularFunction	PositivelyRegulates	MolecularFunction	27
MolecularFunction	Regulates	MolecularFunction	30
Phenotype	IsA	Phenotype	14650

Supplementary Table 5: Knowledge graph relations between node types.